DESIGN AND IMPLEMENTATION OF A REAL-TIME SYSTEM TO CHARACTERIZE FUNCTIONAL CONNECTIVITY BETWEEN CORTICAL AREAS

by

Mohammadbagher Parsa Gharamaleki

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

April 2017

DEDICATION

To my parents and sisters for their unconditional love, support and encouragement.

ACKNOWLEDGEMENTS

I would like to thank Dr. Brendan Mumey, Dr. Behrad Noudoost, and Dr. Clem Izurieta. I had this opportunity to work under their supervision and receive constant support from them. I also would like to thank Dr. Neda Nategh and Dr. Kelsey Clark for their support and advice during this work.

I extend my acknowledgement to Casey Stengel and Alex Lear from Neuralynx, Inc. for their support and help during my internship in Summer of 2016 and Spring of 2017.

I thank Gianforte School of Computing and Department of Cell Biology and Neuroscience for supporting my research and graduate studies.

This work is supported by Montana State University startup fund, Whitehall 2014-5-18, NSF BCS143221, NSF EPSCoR1632738 and NIH R01EY026924 grants to Behrad Noudoost.

TABLE OF CONTENTS

1. INTRODUCTION 1

Background and Motivation 1
Collision Technique 1
Chief Challenges in Utilizing Collision Techniques 4
Definition of Terms 7

2. METHODS 10

Requirement and Specification 10
Dependencies 11
Access, Management, and Security 14
Portability 15
Scalability 15
System Features 15
Latency 15
Design Constraints 16
Integration of a third-party Application Program Interface 16
Operating Environment Design 17
User Interface 17
Selecting Compute Platform 17
Real-Time Systems 19
Soft Real-Time and Hard Real-Time 19
Real-Time Operating System 20
The Algorithm 20
Spike Match and Stimulation Triggering Algorithm 20
Run-Time Complexity 22

3. HARDWARE PROCESSING PLATFORM 24

The HPP Processing Subsystem 24
The HPP Programmable Logic 25
PS/PL Intercommunication 25
Spike Match and Stimulation Triggering on the HPP 26
Application Development Environment 27
System Schematic 28

4. SOFTWARE PROCESSING PLATFORM 30

Kernel Preemption 31
Real-Time Preempt Patch of Linux 32

TABLE OF CONTENTS — CONTINUED

Real Time Scheduling Algorithms....32
Memory Management....33
Application Development Environment....33
System Schematic....34

5. RESULTS....36

Temporal Precision....36
Channel Specificity....38
Waveform Identification....40
Threats to Validity....42

6. CONCLUSIONS....43

Future Directions....45

REFERENCES CITED....47

LIST OF TABLES

Table Page

1. The format and data type of “NRD” raw data record 13

2. Minnow-Board technical specification 30

LIST OF FIGURES

Figure Page

1. Orthodromically activated neurons shows variable-latency evoked spikes and fails the collision test (a,b). Antidromically activated neurons shows fixed-latency evoked spikes and passes the collision test (c,d) 3

2. DLSX converts the analog signal recorded by the electrode to A/D values and sends two copies of data to the hosts through UDP packets 11

3. Spike Match and Stimulation Triggering Algorithm: If the spike waveform lies between the upper and lower bands of the template, then it is a match....22

4. SMST has the highest priority between tasks, and at of the end of each tick, the kernel puts the SMST at Running state....27

5. Hardware Processing Platform (HPP) and the stages of signal processing for experimenter-guided neuronal isolations followed by real-time automatic spike detection and sub-millisecond stimulation triggering 28

6. Software Processing Platform (SPP) and the stages of signal processing for experimenter-guided neuronal isolations followed by real-time automatic spike detection and low-latency stimulation triggering. 35

7. An artificially generated waveform (top) sent to the HPP and the SPP, is matched to a template and generates a TTL pulse (middle) in ~ 0.8ms for the HPP and ~1.5ms for the SPP....36

8. Across many waveform presentations, the system reliably matches the waveform to the template in the HPP, with triggering latencies consistently between 750 and 850μs 37

LIST OF FIGURES — CONTINUED

Figure Page

9. FIFO and RR policies show more deterministic latencies in the SPP, which are always lower than 1.6ms regardless of CPU usage load 38

10. The HPP and SPP only matches incoming waveforms to templates on specified channels 39

11. Comparing multiple examples of two different spike waveforms to the full 64-point template shown will trigger spikes for the red waveform but not the black....40

12. Template-matching criteria can be modified to optimize processing time and waveform specificity 41

LIST OF ALGORITHMS

Algorithm Page

1. Spike Triggering Match for UDP Packets 21

LIST OF EQUATIONS

Equation Page

1. Run-time complexity for the algorithm. The worst-case complexity of the algorithm is $\theta(n3)$.....23

2. The template values obtained in 16-bit are converted to 24-bit A/D count.....30

ABSTRACT

Despite a thorough mapping of the anatomical connectivity between brain regions and decades of neurophysiological studies of neuronal activity within the various areas, our understanding of the nature of the neural signals sent from one area to another remains rudimentary. Orthodromic and antidromic activation of neurons via electrical stimulation ("collision testing") has been used in the peripheral nervous system and in subcortical structures to identify signals propagating along specific neural pathways. However, low yield makes this method prohibitively slow for characterizing cortico-cortical connections. We employed recent advances in electrophysiological methods to improve the efficiency of the collision technique between cortical areas. There are three key challenges: 1) maintaining neuronal isolations following stimulation, 2) increasing the number of neurons being screened, and 3) ensuring low-latency triggering of stimulation after spontaneous action potentials. We have developed a software-hardware solution for online isolations and stimulation triggering, which operates in conjunction with two hardware options, Hardware Processing Platform (HPP) or a Software Processing Platform (SPP). The HPP is a “system on a chip” solution enabling real-time processing in a re-programmable hardware platform, whereas the SPP is a small Intel Atom processor that allows soft real-time computing on a CPU. Employing these solutions for template matching both accelerates spike sorting and provides the low-latency triggering of stimulation required to produce collision trials. Recording with a linear tetrode array electrode allows simultaneous screening of multiple neurons, while the software package coordinates efficient collision testing of multiple user-selected units across channels. This real-time connectivity screening system enables researchers working with a variety of animal models and brain regions to identify the functional properties of specific projections between cortical areas in behaving animals.

CHAPTER ONE — INTRODUCTION

Background and Motivation

Most brain areas display a mix of different response properties[1]; this is especially true for the prefrontal and parietal cortical areas involved in the control of complex behaviors. We know a great deal about the anatomical connectivity between brain regions[2], [3], and also about the types of activity within each area, but few methods exist for combining the anatomical and functional data to understand what information travels between specific areas. One method for identifying the anatomical connectivity of neurons in behaving animals is the collision technique [4]. This method is very reliable but has proven too laborious to be commonly used in cortical neurophysiology studies[5], [6]. With the goal of making it easy and efficient to collect paired anatomical and functional data, we developed a semi-automated system for the high-throughput screening of connectivity between brain areas using a combination of hardware and software solutions.

Collision Technique

The collision method uses electrical stimulation to identify neurons in one area which either project to, or receive input from, a second area. A stimulating electrode is placed in the receiving area, and a recording electrode in the projecting area, where the anatomical connectivity of neurons will be screened.

First, the stimulation electrode delivers stimulation, and screening starts by looking for neurons in the projecting area with spikes reliably occurring shortly after stimulation. This includes both neurons receiving input from the receiving area, in which the action

potential travels away from the cell body down the axon, known as orthodromic activation, and it will also include neurons which project to the receiving area, in which the evoked spike travels backwards up the axon to the cell body, known as antidromic activation. Evoked spikes that travel orthodromically must then cross a synapse before driving activity in the projecting area, causing variability in the spike arrival time. Figure 1shows an orthodromically activated neuron on five different trials, when stimulation is delivered at time zero the timing of the evoked spike ranges from about 5 milliseconds (ms) to 11 ms after stimulation. Spikes that travel antidromically have no synaptic delay, and therefore arrive more quickly and with less variability. So, looking at this antidromically activated neuron, for five different stimulation events the evoked spike always arrives at the same time after stimulation (Figure-1c).

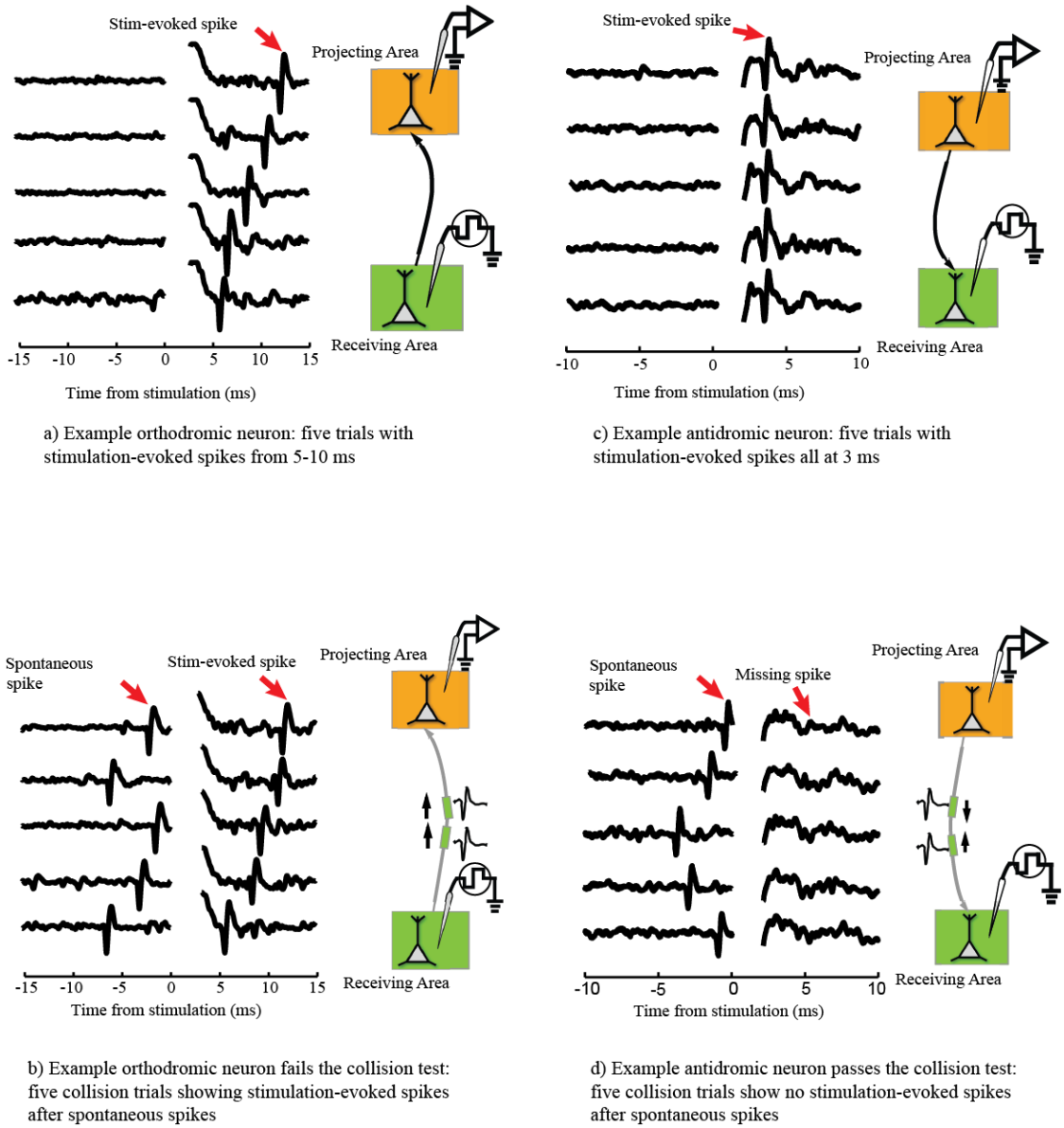


Figure 1. Orthodromically activated neurons show variable-latency evoked spikes and fails the collision test (a,b). Antidromically activated neurons show fixed-latency evoked spikes and passes the collision test (c,d).

To verify whether a neuron is antidromically activated we use the collision test. Stimulation is delivered just after a spontaneous spike in the neuron being recorded: if the evoked spike is traveling orthodromically down the axon, it follows behind the spontaneous

spike and evokes a spike as before. For our orthodromic example neuron, Figure 1b shows that there are still evoked spikes even when stimulation occurs after a spontaneous action potential. If the evoked spike is traveling antidromically up the axon (Figure 1d), it will "collide" with the spontaneous spike and the accompanying refractory period in the axon, and no evoked spike will be seen following stimulation. As a result, for the antidromically activated neuron, there are no evoked spikes detected in cases where stimulation happened just after a spontaneous action potential.

Chief Challenges in Utilizing Collision Techniques

Orthodromic and antidromic electrical stimulation through microelectrodes has been used to determines whether a neuron projects to or receives input from a second area. This technique can be used to reliably identify neurons that project to or receive input from the receiving area, the basics of this technique were developed decades ago[4]. In our experience, there are three chief challenges preventing more widespread use of this method between cortical areas: isolation quality, yield, and the ability to perform low-latency spike matching and stimulation triggering.

Isolation quality: One difficulty of performing collision experiments with a single recording electrode is that electrical stimulation of a connected brain region is likely to activate many neurons around the recording site with low latency. The firing of many neurons whose activity can be seen in the recording electrode so close in time can make the isolation of individual neurons during this time period quite difficult. The use of tetrode electrodes which contain units of four bundled electrodes can enable us to much more

accurately isolate multiple neurons, providing a three-fold improvement in isolation over single electrodes [7].

Recording the same action potential from multiple electrical contacts with slightly different positions relative to the neuron allows more accurate assignment of spikes to individual neurons, and the ability to separate spikes from multiple neurons even when they occur simultaneously [7], [8].

Yield: Because only a small fraction of neurons project from one cortical area to another, a large number of neurons must be screened to identify neurons of the desired anatomical connectivity. This makes isolating and screening individual neurons using single electrodes extremely time consuming. However, utilizing array electrodes can dramatically increase the number of neurons screened in a single recording session.

Low-latency spike matching and stimulation triggering: We need a way to trigger stimulation for collision testing. Randomly delivering stimulation and sorting the data post-hoc to find collision events requires many more stimulation trials. By developing a method for low-latency triggering of stimulation after spontaneous action potentials, we can efficiently generate collision trials and determine the connectivity of neurons during the experiment. Of these three chief challenges, low-latency spike matching and stimulation triggering has the higher priority, since it directly affects the efficiency of experiments. Therefore, we will not further discuss isolation and yield here. There is no doubt streamlining technologies and other recent advances in the hardware aspects of data acquisition systems, combined with our software solution will provide a powerful new tool for dissecting functional circuitry in behaving animals. Thus, the truly innovative aspect of

this project is not incremental improvements in implementing these methods, but the multiplicative gains of uniting these technologies and providing a software solution with the aim of making this technique easy to use for the research community.

Short-latency template matching and stimulation triggering have classically [4] been achieved by randomly delivering stimulation and post-hoc sorting for stimulation events delivered after spikes. This method has a very low fraction of useful stimulation events. For example, if the recorded neuron has a mean firing rate of 20 Hz, then only 8% of randomly delivered stimulation trials will provide useful collision data for neurons with axonal delays of 4 ms or less. The ability to detect a spike and immediately trigger stimulation would therefore dramatically reduce the number of stimulation trials required, and significantly expedite the screening throughput. To perform the collision test, and thus establish whether an activated neuron projects to or receives input from the stimulated area, stimulation must be delivered very soon after an action potential in the recorded neuron (less than the axonal delay of a given neuron; we have observed V4-projecting FEF neurons with axonal delays as low as 2 ms). By developing a method for low-latency triggering of stimulation after spontaneous action potentials, we can efficiently generate collision trials and determine the connectivity of neurons during the experiment. We have developed a software solution to simultaneously screen many neurons for connectivity with a particular region and trigger stimulation within the time constraint as mentioned earlier.

Definition of Terms

The current work is considered an interdisciplinary study. We are addressing a computational issue that exists in neuroscience by using techniques and methods from computer science. Therefore, having a chapter for the definition of the most frequently used terms is helpful.

Action potentials

An event in which a neuron's electrical membrane potential changes quickly, and it follows a particular pattern of rises and falls.

Antidromic activation

Neuron projecting to the stimulation area, in which the evoked spike travels backward up the axon to the cell body.

Axon

Part of a nerve cell that transfer information to different neurons and conducts electrical signals away from the cell body.

Cheetah

Neuralynx's software for electrophysiology recording and experiment control.

Dendroid

Part of a nerve cell that propagates received electrical stimulation to the cell body.

Digital Lynx SX (DLSX)

Neuralynx's digital data acquisition system which supports up to 512 channels analog to digital conversion.

Electrical Brain Stimulation (EBS)

A technique used in neuroscience research to stimulate a neuron by excitation of the cell membrane using a small electric current.

Hardware Description Language (HDL)

A specific language to define structure and behavior of a digital logic circuit.

Linear tetrode array electrode

A microelectrode that contains units of four bundled electrodes. A tetrode electrode can be used to classify extra-cellular action potentials into a group of neurons.

NetCom API

Neuralynx's application programming interface (API) that allows the user to build an application to monitor and control Cheetah remotely.

Offline sorting

The act of post-hoc processing of recorded data in order to classify and cluster recorded neuronal activity during the experiment.

Orthodromic activation

Neuron receiving input from stimulation area, in which the action potential travels away from the cell body down the axon.

Peripheral nervous system

The nervous system outside the brain and spinal cord.

Raw Data File (NRD)

Neuralynx's file format for a raw data A/D record. These files end in the NRD extension.

Real-time processing

A hardware-software solution with real-time constraints that guarantees satisfying defined time constraints.

Simulink

An interactive programming environment for modeling, simulating and testing embedded systems.

Single Electrode Spike Record (NSE)

Neuralynx's file format for a single electrode spike record. These files end in the NSE extension.

Spike Sort 3D

Neuralynx's software that performs online and offline waveform-to-cell sorting and classification.

Subcortical structures

Refers to a region of the brain below the cortex.

Synaptic delay

A time delay between the release of a neurotransmitter from a presynaptic membrane, and diffusion into the synaptic cleft to binding to a receptor site on the post-synaptic membrane.

System Generator

A Xilinx high-level programming tool for designing high-performance FPGA solutions.

System on Chip (SoC)

An integration of multiple computing units on a single chip.

CHAPTER TWO — METHODS

Requirements and Specifications

The common way of measuring the electrophysiological activity of a single neuron is to place a single electrode near a neuron and then record the changes in voltage produced by action potentials. The electrode is placed inside the brain of a behaving animal, and while the animal is performing a particular task, the electrode can record any change in the voltage. The ability to place the electrode close to the cell membrane and record the intracellular or extracellular activity provides a unique single neuron resolution. The analog signal recorded by the electrode is transferred to an analog to digital (A/D) converter. In this study, we used Neuralynx's Digital Lynx SX (DLSX) data acquisition system to convert neuronal activity to A/D values. This system is a DC-coupled wide band recording solution which utilizes an amplifier and a 24-bit A/D converter. The data is sampled at 32 kHz with 32-bit resolution and stored in UDP datagrams. The size of a packet depends on the number of input boards available in DLSX.

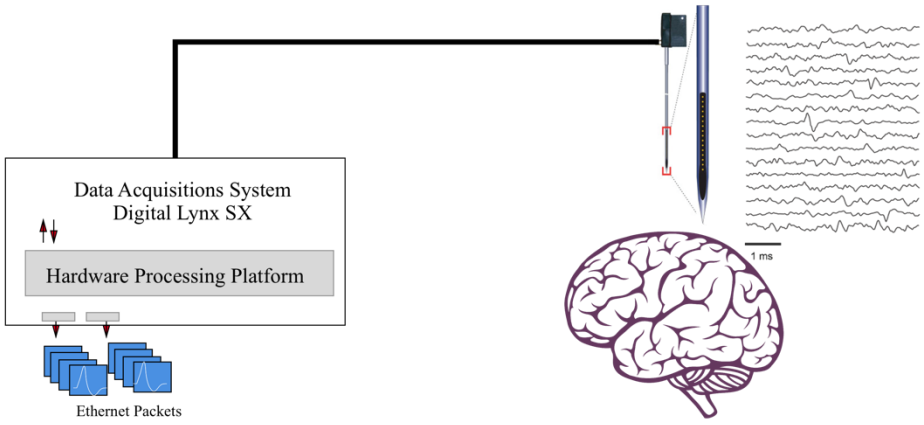


Figure 2. DLSX converts the analog signal recorded by the electrode to A/D values and sends two copies of data to the hosts through UDP packets.

The DLSX is used in conjunction with Neuralynx's Cheetah software. Cheetah supports recording from multi-channel electrodes with up to 512 channels. Software requirements were gathered from the end-users who are members of the Department of Cell Biology and Neuroscience at Montana State University. After formulating features, specification, and requirements, the requirements were validated with by the help of engineers at Neuralynx, Inc.

Dependencies

The software has a direct dependency on the output data stream of the DLSX and Cheetah configuration. Spike detection type is required to be triggered using the electrode voltage slope detection method. Filter settings need to be configured with no active filtering. Activating any low/high pass filter causes a non-deterministic delay in DLSX. Additionally, the user is required to use either Neuralynx's raw data file format "NRD" or

spike files format “NSE” to extract the template from a recorded session. Neuralynx’s SpikeSort3D (SS3D) can be employed to generate the template in either offline or online mode. Another factor to consider is the sampling rate of DLSX, this value can be set up to 40 KHz, and in a standard recording session, this frequency is between 30 KHz to 32 KHz.

Finally, the DLSX uses Ethernet packets to communicate with the host computer (Cheetah). There are two types of protocols utilized by DLSX, Transition Control Protocol (TCP) for Cheetah commands to control DLSX, and User Datagram Protocol (UDP) for transferring sampled data. The Ethernet packets that output from the second data port are duplicates of the first port on the DLSX. Once the acquisition is started on Cheetah, the DLSX starts sending out packets on both ports. The TCP command packets are only using the first data port. Therefore, if using the second data port, the user won’t be able to control or adjust the functionality of the DLSX, which include starting acquisition and changing references. There are three sections to the UDP packet: header, data, and footer. The header and footer sections are fixed in length, and the data section is variable depending on loaded input boards on the DLSX system.

Storage format for a raw data A/D record is identical to the UDP packet format. These files end in the NRD extension. Table 1 shows the format and data type for a raw data record.

Table 1. The format and data type of “NRD” raw data record.

Int32	STX	Start of transmission identifier for the A/D record. This value is always 2048.
Int32	Packet ID	Value to identify the type of packet. There is only one type of packet for the NRD file format. This value is always 1.
Int32	Packet Size	This value is equal to the number of AD channels plus the number of Extra values in the packet.
UInt32	Timestamp High	The 32 high order bits that make up the 64-bit timestamp value. This value must be combined with the Timestamp Low value to create the full 64-bit Cheetah timestamp whose value is in microseconds
UInt32	Timestamp Low	The 32 low order bits that make up the 64-bit timestamp value. This value must be combined with the Timestamp High value to create the full 64-bit Cheetah timestamp whose value is in microseconds.
Int32	Status	Reserved
UInt32	Parallel Input Port	The current value of the Parallel Input Port.

Table 1 Continued

Int32[]	Extras	Reserved. This is currently an array ten (10) values containing reserved information from the hardware. While actual value of this array is not needed for data analysis, the number of values in this array needs to be considered when calculating the size of the Data block from the Packet Size value detailed above.
Int32[]	Data	Data values for the current record. The number of data values is based on the packet size value. The number of values in this array depends on the number of channels in your acquisition system. The size of this array is calculated by subtracting the number of Extra values from the Packet Size value detailed above.
Int32	CRC	The CRC value for the current record only.

Access, Management, and Security

The solution should be designed for research purposes and does not need to follow medical regulation. Thus, security and access management of the device does not have a high priority for design consideration. However, for the accurate functioning of the solution, any unwanted access to the DLSX configuration should be prevented.

Portability

The user should be able to move the implementation from one architecture to another; the software should not limit the user to a particular platform. Thus, hardware-dependent code such as platform dependent I/O handling should cover less than 10% of the line of code (LOC) in the source code.

Scalability

The user should be able to quickly scale up the solution to process more channels and templates simultaneously. Thus, the solution should be scalable in design and implementation, allowing the user to add more processing units to cover more channels and templates.

System Features

The user should be able to upload the extracted template to the solution in binary or ASCII format. Also, the user should be able to select up to 32 channels for screening and comparing each channel to a user-defined target template. Furthermore, the user should receive a Transistor-Transistor Logic (TTL) signal upon the occurrence of any template match as output. This TTL should be between 3-5 volts so that it can be used as an input for the stimulator.

Latency

Latency is defined as the time difference from when a task gets inserted into a ready queue and its effective execution time. This application should meet the deadline for each

individual template match and trigger. Based on the experimental observation, V4-projecting FEF neurons have axonal delays as low as 2 ms, which means the application should be able to provide triggering with latency less than 2 ms. Although a missed deadline is not considered a total system failure, it may reduce the yield of an experiment, thus requiring more trials to run.

Design Constraints

The implantation should be in the C programming language, because of the need to control the hardware in lower level. The C programming language has many advantages for this solution since supports many machine-related features such as bit, byte manipulation, and memory management capabilities. It also provides a tool to access the CPU's register and other memory mapped hardware[9]. Further, several other rich characteristics of this language such as pointers and types are essential for runtime optimizing. The C programming language is universally available for all existing platforms, and most of the architecture come with at least one C compiler. The C language does not rely on the garbage collector; it has deterministic use of the resource, and has a very small runtime memory footprint.

Integration of a Third-Party Application Program Interface (API)

To control the overhead and optimize the software for smallest possible runtime and performance, the application should be implemented with zero to minimum dependency on any API or third party libraries. Also, utilizing a third-party package may result in an unmaintainable software product.

Operating Environment Design

The solution should be able to operate in an electrophysiology lab environment. The solution should be able to function as expected in the temperate range of 10 degrees to 35 degrees centigrade (50 degrees to 90 degrees Fahrenheit). It should be noted that high-frequency and low-frequency noises have an adverse impact on the DLSX which may directly alter the system's functionality based on its dependency to the DLSX. Noise source reduction is a common practice for all experiments.

User Interface

The user should be able to communicate with the system through a Command Line Interface and/or through a graphical user interface.

Selecting Compute Platform

We can assume that all available programmable platforms are Turing complete [10], so theoretically we should be able to address any task on a programmable platform. However, considering our low time-latency requirement that we are dealing with here, we could choose a different compute platform among more feasible options, such as Central Processing Unit (CPU), Graphic Processing Unit (GPU) and Field Programmable Gate Arrays (FPGA). One option is a processor. CPUs are optimized for latency, and they are easy to program, and can deliver multiple complex tasks. On the other hand, GPUs are optimized for throughput, so they are well-designed to handle programs that fit with multiple data single instruction (SIMD) models; it is exactly for this reason, that they perform very well on tasks like image processing where same instructions are being

executed on millions of pixels. GPUs are also capable of handling floating point operations. However, software with rich flow control and branches can be handled very easily on CPUs as compared to GPUs [11]. The last platform to consider is FPGAs. FPGAs are digital chips that can be configured to perform a particular computation task [10]. While FPGAs usually have a low clock frequency, such as 100 MHz, they are capable of parallel processing, and therefore are the ideal choice for running multiple independent processes in parallel. Moreover, FPGAs are extremely fast in manipulating registers compared to CPUs. For example, it might take hundreds of instructions for a CPU to perform a shift operation, while a FPGA can accomplish the same goal only in a few clock cycles. CPUs and GPUs are mostly rely on memory access to execute instructions, plus FPGAs can take advantage of data flow streaming [12]. FPGAs are more flexible, providing more customization ability compared to the other predefined System on Chip (SoC) solutions, such as the multi-core processor or GPUs. However, FPGAs are relatively hard to program and therefore lead to a higher cost for development and maintenance. Clearly, every platform has some advantages and disadvantages for any given set of requirements. However, considering our requirements for this project, we selected two computing platforms to implement our solution: CPUs and FPGAs. Having more than one platform provides us with the opportunity to compare performance, ease of use, and scalability of the platforms. We used Neuralynx's Hardware Processing Platform (HPP), a Xilinx FPGA of Zynq 7000 series with a dual ARM Cortex A9 processor for our FPGA based implementation, and the Software Processing Platform (SPP), a MinnowBoard Dual Turbo with a dual Atom E3800 series processor (more details in Chapter 3 and 4).

Real-Time Systems

Real-time systems are systems with a timeline as their most critical requirement. The real-time system must meet defined time constraints [9], with a guaranteed upper-bound for a specific process [13]. Real-time systems are primarily embedded systems that are optimized for a particular real-time application. Real-time systems are normally categorized into two groups, soft real-time and hard real-time systems.

Soft Real-Time and Hard Real-Time

A soft real-time system is a system in which missing a deadline would degrade its performance while not being considered as a total system failure. However, failing to meet the time constraint in a hard real-time system is rated as a system failure. In our case, failing to meet the time constraint reduces the efficiency of an experiment and depending on the scale of the failure it might require more trials during the experiment or necessitate a post-hoc analysis. We can classify our application at a soft real-time level since missing some number of deadlines degrades the performance but does not result in system failure. A real-time system usually requires an operating system to make the development process manageable by providing an abstract layer on top of the hardware system and handling several system level tasks, such as resource sharing, memory management, and scheduling. For a real-time system, full-blown operating systems are typically not desired because of their complexity and overhead that comes with multiple unnecessary features [9].

Real-Time Operating System

The solutions require being dependent on an operating system for resource management, scheduling, device driver, network stack, and I/O operations. The operating system for our solution should include a kernel with the features previously mentioned, and a command line interface or graphical user interface. Moreover, the operating system must include a real-time scheduler in order to meet the response-time requirements.

Commercial real-time operating systems (RTOSs) are more popular for aerospace, military, and medical applications, compared to the open-source options. The commercial real-time operating systems are more stable, and usually come with a constant security update and support [9]. However, many of these operating systems does not come with a benchmark of their determinism data, so it is difficult to decide which real-time operating system is suitable for a given time constraint [14]. For the current application, the real-time operating system should be easy to maintain, update, and administrate. In addition, providing more than one real-time scheduling algorithm is considered an advantage.

The Algorithm

Spike Match and Stimulation Triggering Algorithm

To process sampled data, we use a sliding window over a vector of buffered values. Let L and H be vectors to store the lower band and higher band values of templates, typically a template has 32 lower-band values and 32 upper-band values, so depending on the loaded number of templates, the length of L, and H will vary. As soon as a sample arrives, depending on the form of samples (either in an IP packet or a memory location

with an interrupt), an extra step may be required to process the header of each packet. (Algorithm 1). The samples get collected in a first in first out buffer with a fixed length called "Windows size" which is equal to the length of a template (32 values).

```
1.  B ← Number of loaded input boards
2.  I ← Number of loaded templates
3.  J ← Number of selected channels
4.  K ← Length of the template vector
5.  Li ← Low-band values of template 'i'
6.  Hi ← High-band values of template 'i'
7.  Mij ← Set match indicator for template 'i' and target channel 'j' to zero
8.  for every Ethernet packets do
9.      S ← Packet Header
10.     P ← Extract Packet Size from S
11.     if P = S+B then
12.         Dkj ← Store the payload to a buffer of size K
13.         if Dkj is Full then
14.             for every i ∈ I do
15.                 for every j ∈ J do
16.                     for |K| do
17.                         if Li < Dkj < Hj then
18.                             Mij ← Mij+1
19.                             if Mij >= Threshold then
20.                                 GPIOij ← 1
21.                                 GPIOij ← 0
22.                                 Break
23.             Delete D1j and shift the window, K elements to right
```

Algorithm 1. Spike Triggering Match for UDP packets

The next step is to compare each of the 32 buffered values of a channel against a user-defined template. If the number of A/D values between the upper band and the lower band of the target template exceeds a user-defined threshold, the channel is considered a match (Figure 3). As soon as a match occurs, the system sends a TTL pulse as an output (Spike Match and Stimulation Trigger (SMST)).

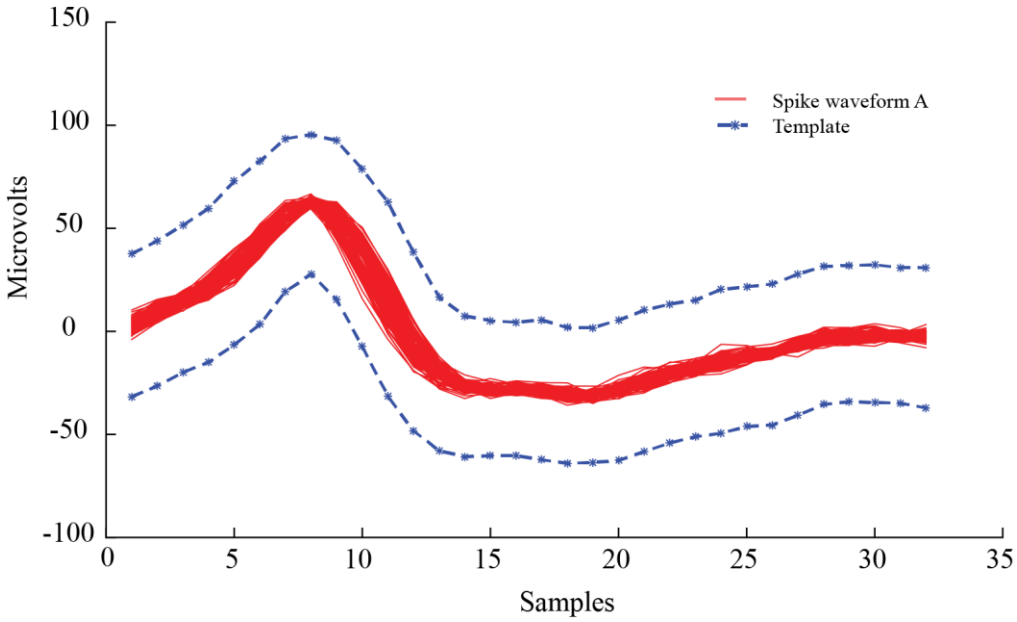


Figure 3. Spike Match and Stimulation Trigger Algorithm: If the spike waveform lies between the upper and lower bands of the template, then it is a match.

Run-Time Complexity

We are interested in analyzing the algorithm and understanding its worst case running time complexity. In theory, and only considering the requirement of the current setup, the system should be able to handle a 32 channels comparison against 32 templates (by limiting factor of input boards on the DLSX). Analyzing run-time complexity for the algorithm shows the worst-case complexity of $\theta(n^3)$ (Equation 1). It should be noted that we are looking at running time of processing each data packet in its arrival time. Complexity class does not explain the actual running time. However, running-time analysis shows how an algorithm scales by changing the input size.

$$T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} c$$

$$T(n) = \theta(n^3)$$

Equation 1. Run-time complexity for the algorithm. The worst-case complexity of the algorithm is $\theta(n^3)$.

In practice, real-time response constraint and system resources such as the number of General Purpose Digital I/O (GPIO) for triggering, are the potential limits to the number of possible channel versus template comparisons. Furthermore, each sample arrives at the processor normally with a sampling rate of 32 kHz (32 microseconds), so buffering the incoming values into a buffer of size 32 adds a small delay of 1000 microseconds for the first comparison. Once the buffer is full, comparison begins, and the rest of the process is carried by the arrival of each new sample about every 32 microseconds. Thus, the inner loop of the algorithm (comparison and triggering the TTL) should be able to run faster than 32 microseconds to prevent any loss of data on a single thread process.

CHAPTER THREE — HARDWARE PROCESSING PLATFORM

The Hardware Processing Platform (HPP) is a System on Chip (SoC) solution which provides a real-time capability at the FPGA level as well as the CPU level. This solution utilizes a Xilinx Zynq 7000 all programmable SoC which comes with a dual ARM Cortex A9 processor. The HPP provides the user with access to a C programming language to implement any software solution and run it on the CPU. Also, the user can use either a Hardware Description Language (HDL), such as VHDL, or Verilog, or System Generator co-simulation, for the development of FPGA fabric.

The HPP Processing Subsystem

The operating system installed on the HPP Processing Subsystem (PS) is FreeRTOS. This operating system is hosted as an application in Xilinx's standalone OS board support package (BSP). HPP's FreeRTOS is a version of officially released ZC702 FreeRTOS. FreeRTOS is designed for real-time embedded systems with a microcontroller or a small microprocessor. A real-time application with hard or soft real-time requirements can be built on top of its scheduler. FreeRTOS also provides multi-threading functionality. Therefore, multiple independent processes can run on the HPP processor. The user assigns the priority of the application, thus providing a base for the kernel scheduler to decide in which order the process or task should be executed.

The HPP Programmable Logic

The FPGA fabric, known as programmable logic (PL), provides logic-intensive processing and communication with the DLSX systems. This programmable logic is designed to communicate with the DLSX using any of the HDL, C or System Generator methods.

PS/PL Intercommunication

To provide real-time manipulation of incoming data, the DLSX sends a copy of the data which is provided by the Cheetah software- to the HPP board. The data in HPP is accessed in the FPGA fabric and then routed to the Zynq Processing subsystem in a Dynamic Data Rate (DDR) as a circular buffer named "HPP_Data". The PL alerts the PS by sending an interrupt once a sample data packet is available.

"vHPP_DataInterruptHandler" is an Interrupt Service that handles this interrupt by freeing the binary semaphore "xHPP_Data_Sem" and incrementing the number of samples loaded into DDR (num_data_buffers_loaded). The "xHPP_Data_Sem" semaphore is used by the real-time algorithm to process each sample as it arrives.

There are two main ways to communicate with the DLSX: the first is with the logic; and the second is with the Cheetah emulated command. Users can perform a function call to communicate with the DLSX; the commands get buffered in a FIFO buffer.

The PL sends back an acknowledgment to PS in the same way it does for the sample data, giving the user Direct Memory Access (DMA) to the acknowledgments in DDR (Double Data Rate) RAM. The acknowledgment comes with an interrupt to the PS. HPP has

"vHPP_AckInterruptHandler()" handler which handles this interrupt, and toggles the acknowledgment GPIO line which allows the PL to de-assert the interrupt.

Spike Match and Stimulation Trigger on the HPP

The Zynq Processing subsystem consists of two ARM Cortex A9 cores. However, only the first processor is utilized by the HPP applications. The Spike Match and Stimulation Trigger task is implemented using C language on top of FreeRTOS. SMST on FreeRTOS has an entry point and runs continually within an infinite loop looking for incoming samples and comparing each value in target channels against given templates. When the SMST is running, FreeRTOS puts this process in the "Running" state, and as soon as it gets deleted from the task stack, FreeRTOS moves the SMST to "Ready" state. The FreeRTOS scheduler is the only part of FreeRTOS that is capable of moving tasks between these two states. FreeRTOS provides an API function called "xTaskCreate()" to create a task. This API is used to create the SMST and set its priority in FreeRTOS. A task in FreeRTOS can obtain a priority between 0, the lowest priority, up to a maximum priority of 1 unit less than a user defined integer. FreeRTOS is designed in a way that only one task can run in the Running state at any given time. However, FreeRTOS scheduler always ensures the highest priority task enters first into the Running state. If there is more than one task with the same priority, the scheduler uses a time slicing technique to move tasks in and out between the Running and Ready states. At the end of each time slice or "tick," the scheduler decides which task to run, and since we are running only one task, the "SMST" with the highest priority gets selected at the end of each tick (Figure 4).

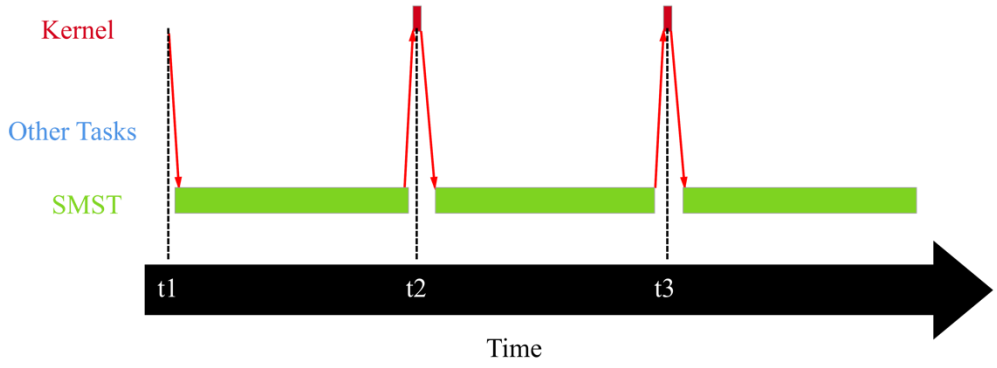


Figure 4. SMST has the highest priority between tasks, and at of the end of each tick, the kernel puts the SMST at Running state.

Selecting the highest priority task depends on the task being ready to execute. If the task is not ready to run, the kernel must choose a task with lower priority. This situation happens when the SMST is waiting for samples or the buffering operation. A task waiting for an event or data is in "Blocked" state. In this state, SMST is not available to the scheduler. Once the SMST receives an alert regarding new data arrival, it moves to the Ready state and is now accessible to the scheduler.

Application Development Environment

The Xilinx software development and Xilinx System Debugger are used for developing, testing, and running the application in HPP. This SDK can process the custom embedded hardware design definition of HPP, and auto-configure memory maps, peripheral registers, library path, and flash memory settings.

System Schematic

The solution for the spike template matching and trigger (SMST) on HPP with its dependencies is presented in

Figure 5. This diagram is divided into two sections: real-time and non-real-time. First, signals from the electrode placed in the brain are converted to A/D values using the DLSX system. The acquired data on the DLSX are available on both Cheetah and HPP, which is housed alongside the DLSX.

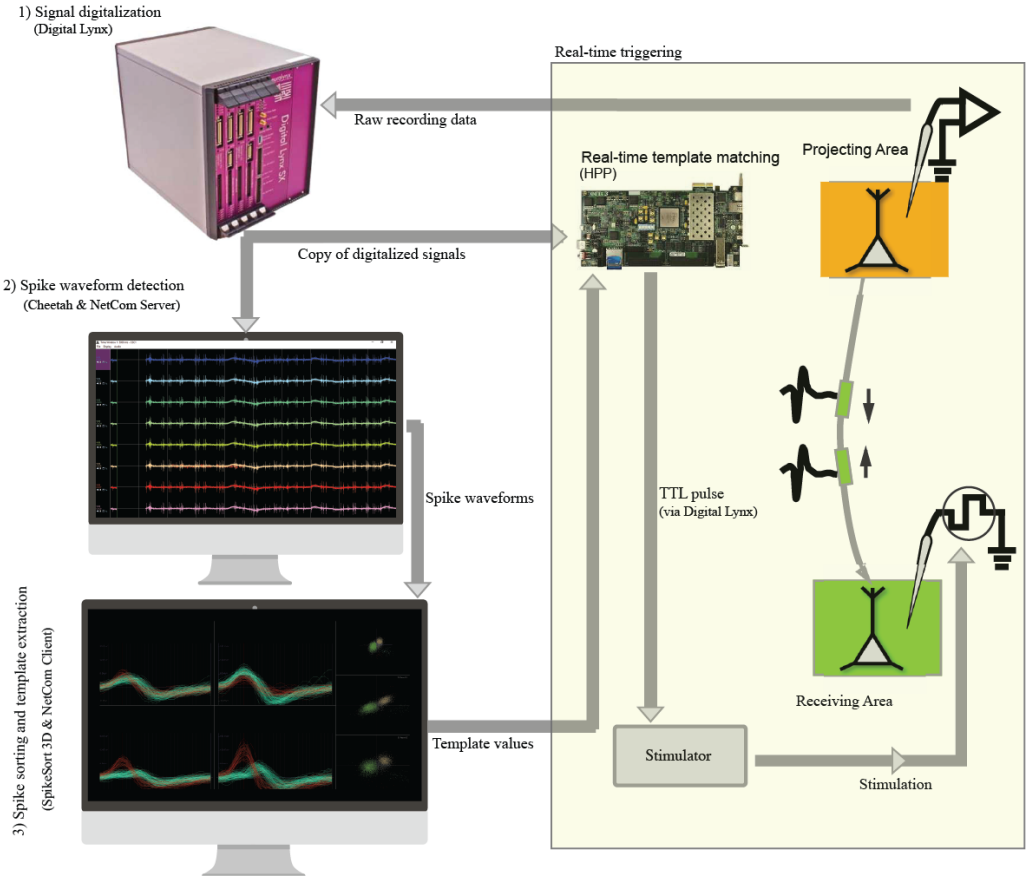


Figure 5. Hardware Processing Platform (HPP) and the stages of signal processing for experimenter-guided neuronal isolations followed by real-time automatic spike detection and sub-millisecond stimulation triggering

Cheetah, running on a Windows machine, performs waveform and spike detection, and then sends the spike waveforms through NetCom to a second desktop computer which is running Spike Sort 3D software, allowing the user to interact with this software to get the best isolation, and then generate one or more spike templates.

The template values obtained in 16-bit are converted to 24-bit A/D counts by reversing the conversion used in Cheetah (Equation 2).

$$V_{24bit} = V_{16bit} \times \left(\frac{2^8}{\frac{131072}{Input\ Range(\mu v)}} \right)$$

Equation 2. The template values obtained in 16-bit are converted to 24-bit A/D counts.

Once the user is ready to switch to real-time triggering mode, the spike templates are sent to HPP. The HPP gets a copy of the A/D counts from the DLSX and performs template matching in real-time, generating a TTL pulse when a matching waveform is identified. This pulse is sent to the stimulator which delivers stimulation to the brain via a second electrode.

CHAPTER FOUR — SOFTWARE PROCESSING PLATFORM (SPP)

The Software Processing Platform (SPP) provides a real-time capability at the CPU level. This solution utilizes a "Minnow-Board Turbot" with a 64-bit Intel Atom E3800 processor series. Table 2 shows the key technical specifications of the Minnow-Board [15]. The Minnow-Board comes with 2GB of memory and 1Gb Ethernet interface.

We used Wind River Pulsar Linux 8.0 [16], a small footprint embedded RTOS that uses relatively few resources, making it ideal for our development of SMST on the Minnow-Board. The Pulsar Linux is supported and tested on the Minnow-Board, and comes with a binary image for easy installation. Also, this operating system removes the traditional cross compile and build complexity which makes development faster and easier for developers as well as end-users.

Table 2. Minnow-Board technical specification

Hardware	Specification
CPU	Dual core intel E3800 (clock speed1.46 GHz) and 1 MB of cache size
DRAM	2GB DDR3L
Storage	SATA2 and Micro SD
I/O Connectors	USB 2.0 and USB 3.0, 8x buffered GPIO, and1Gb Ethernet
Boot loader	UEFI
Power	5VDC coaxial power jack

Additionally, the Pulsar Linux comes with a real-time preempt kernel which can accommodate a soft real-time capability for our application.

Kernel Preemption

The Linux operating system is widely used for many of scientific applications. Applications in Linux run in the userspace or the kernel space. For instance, when an application is executing a system call, the kernel, on behalf of the application executes instructions in the kernel space. An interrupt can preempt the userspace programs; this typically happens when the kernel wants to switch between threads. However, until the release of Linux 2.6, the kernel itself was not preemptible, so, the running thread could not be preempted to run a new thread, causing a non-determinism and jitter to any program running on Linux [17]. Moreover, Linux, as a time-sharing operating system, splits its time between multiple applications, switching between running multiple processes at any given time. To perform such a task, it uses its default class of scheduler called "Completely Fair Scheduler" or CFS. This scheduler monitors all the running processes and adjusts their priority dynamically by increasing priority of those processes that have been waiting and decreasing priority of tasks that have been running for a long time [18]. This scheduler performs very well in situations where the operating system is dealing with multiple types of applications, such as batch processes or multimedia applications [18]. However, a real-time application requires predictable and more deterministic scheduling. Thus, a real-time application must be scheduled by a specialized class of scheduler. The latest Linux kernel version provides two categories of scheduling for real-time applications, including Round Robin Scheduler (SCHED_RR), and First In First Out Scheduler (SCHED_FIFO). The choice of scheduler depends on the application, and should be extensively tested to select the most optimal scheduler for any given application[19].

Real-Time Preempt Patch of Linux

As discussed above, the preemptible kernel moved Linux closer to Real-time operating system. However unbounded priority inversion in the Linux kernel, which puts a hold on running a higher priority application until a lower priority application finishes its running in the critical section [13], can cause non-deterministic latency. In addition, the critical sections in Linux are managed by spin locks, which are a potential source for a large amount of latency. Many of these similar problems for running real-time applications are solved by the introduction of PREEMPT_RT patch (RT Patch) [13]. The RT patch converts spin locks to mutexes, and by doing so, reduces the latency caused by spin locks. Furthermore, RT patch uses Priority Inheritance (PI) in which a thread temporarily inherits highest priority for using a specific resource and no other thread can preempt it. Pulsar Linux 8.0 comes with the RT patch, allowing the user to have more control on timing and scheduling, and making it possible to run a real-time application on the SPP platform.

Real Time Scheduling Algorithms

The real-time scheduling provided in Linux are First In First Out (also known as First Come and First Serve) and Round Robin Scheduling policy. The FIFO policy is not limited to time-slices, unlike the RR scheduling policy [20].

A process from FIFO-class always runs if it has the highest priority, and it preempts all the lower priority tasks. However, there are two cases that a FIFO-class process stops executing: 1) A higher priority task becomes available. In this case, the task with higher priority runs until it voluntarily leaves or finishes; and 2) The process blocks for an event

or an I/O resource. Once the blocked process becomes runnable again, it is available for the scheduler and gets inserted at the end of the priority list [21]. The Round Robin policy is similar to FIFO-classed tasks, with the exception of time-slice limitation. The RR-classed process additionally ceases execution once it reaches its time-slice, and then it moves to the end of the list of executable processes with the same priority [21]. To select a scheduling algorithm that performs better and provides a deterministic latency for the SMST application running on the SPP, we tested all three-scheduling algorithms.

Memory Management

Memory management is a critical part of any real-time application development. Page-faults are normal in a generic operating system, but they introduce a non-deterministic latency which is not acceptable for any real-time system. In a real-time environment, any page-fault should be avoided [22]. Linux provides a system call that locks the process's virtual access space into RAM, actually preventing any page fault caused by getting memory space into the swap space [21]. We used "mlockall" to avoid any page fault during the runtime of the process.

Application Development Environment

Pulsar Linux provides multiple options for fast application development. The user can develop application natively using the built-in compiler and shell or a text editor. Also, the user can use cloud development and run or debug the application remotely on the target device. We used Wind River Helix App Cloud [23] for the development environment, a

cloud-based development environment that provides a unique location for development, as well as testing and debugging.

System Schematic

The solution for spike template matching and trigger on the SPP with its dependencies is provided in Figure 6. (This diagram is the same as the HPP diagram in

Figure 5, with the exception of using the SPP instead of the HPP for processing waveform and outputting TTL pulses.) The diagram is divided into a real-time and a non-real-time section. First, signals from the electrode positioned in the brain arrive in the DLSX and get converted to A/D counts. Next, the DLSX sends a copy of the recorded data to Cheetah and the SPP using UDP/IP stack. Cheetah is running on a Windows machine, performing waveform and spike detection.

Upon receiving a request from the SpikeSort3D for spike waveforms, Cheetah sends the spike waveforms, and spikes become available in SS3D, allowing the user to interact with this software to get the best isolation, and then generate one or more spike templates. Once the user is ready to switch to real-time triggering mode, the spike templates are sent to the SPP. The SPP receives A/D values from the DLSX through UDP protocol, and then performs template matching in real-time, generating a TTL pulse when a matching waveform is identified. This pulse is sent to the stimulator, which delivers stimulation to the brain via a second electrode.

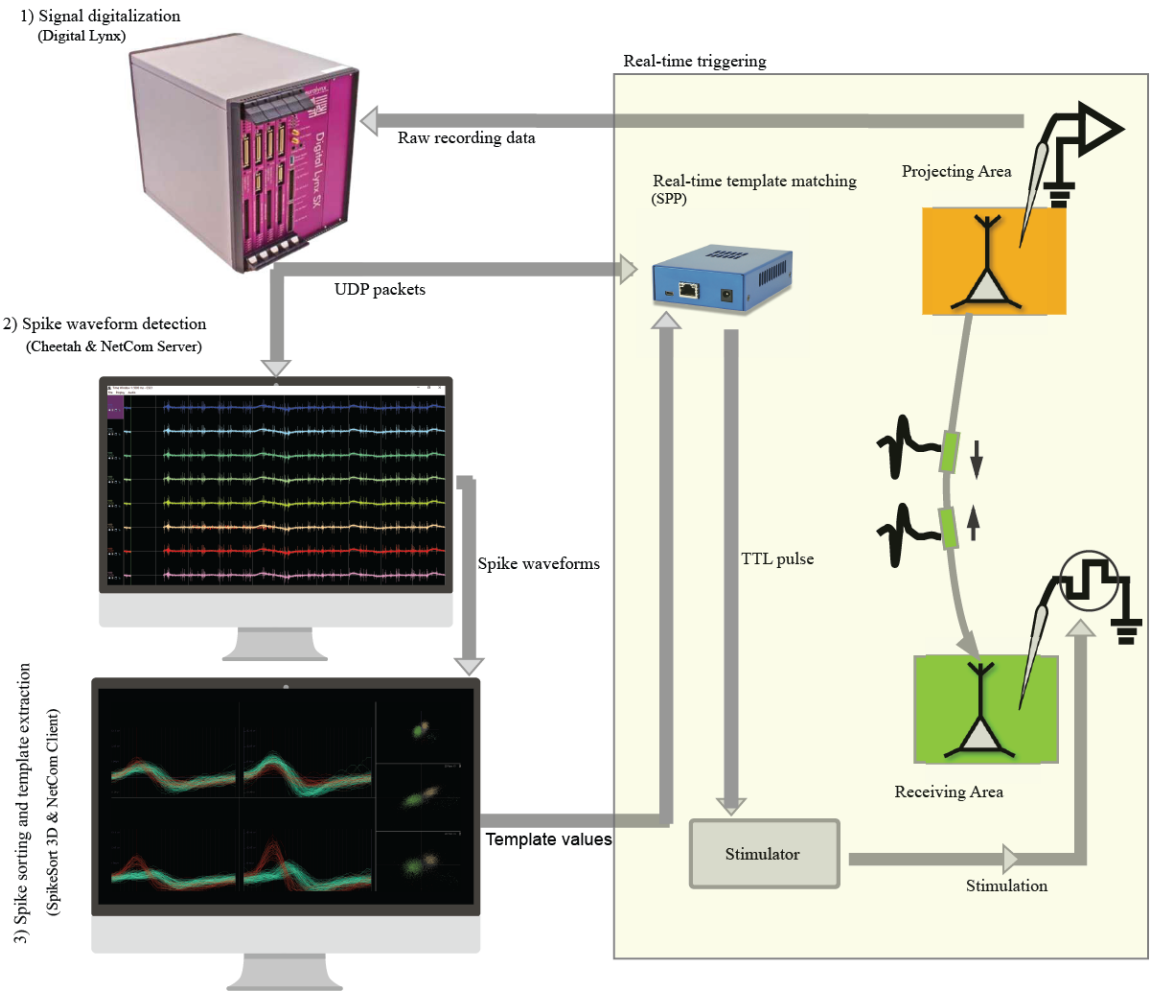


Figure 6. Software Processing Platform (SPP) and the stages of signal processing for experimenter-guided neuronal isolations followed by real-time automatic spike detection and low-latency stimulation triggering.

CHAPTER FIVE — RESULTS

To test the latency of triggering, we utilized an artificially generated spike waveform (extracted from previously generated recording) as input into the system. We measured the latency of a TTL pulse after performing a template matching in the SPP as well as the HPP.

Temporal Precision

The HPP and the SPP were able to match waveforms to a template and generate a TTL output in under two milliseconds.

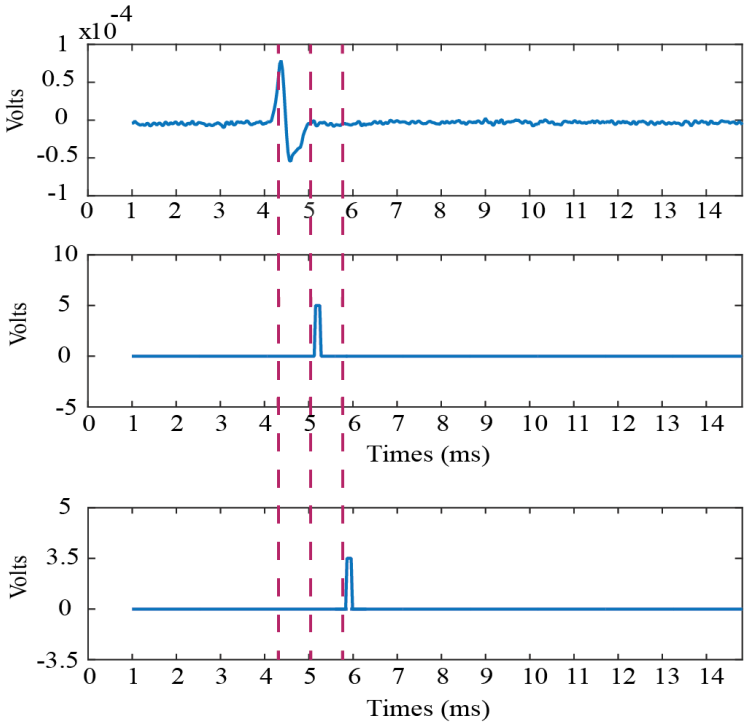


Figure 7. An artificially generated waveform (top) sent to the HPP and the SPP is matched to a template and generates a TTL pulse (middle) in ~ 0.8 ms for the HPP and ~1.5 ms for the SPP.

The delay is measured between the peak of the waveform and the TTL pulse which is about 0.8ms in the HPP and 1.5ms in the SPP. Figure 7 shows the delay between the peak of the waveform and the TTL pulse generated by the HPP and the SPP.

Additionally, the HPP provides more deterministic latency (Figure 8) and consistent delay in triggering which is always between 750 and 825 microseconds with standard division 13.5 microseconds.

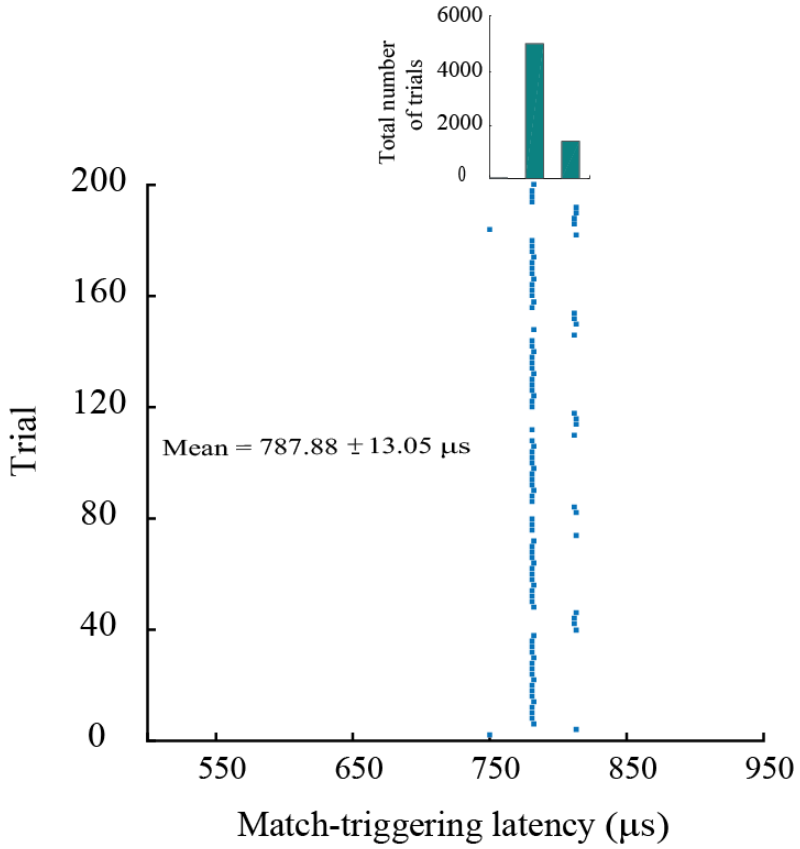


Figure 8. Across many waveform presentations, the system reliably matches the waveform to the template in the HPP, with triggering latencies consistently between 750 and 850 μs.

To measure delay in the SPP, we used both FIFO and RR as real-time scheduling, and CFS policy to run the application in the SPP. The result indicates more deterministic

latency using FIFO and RR policy (Figure 11). Looking at the maximum delay between the peak of the waveform and the TTL pulse when we are running the application as a FIFO-classed and RR-classed shows a deterministic latency, always under 1.6ms regardless of CPU load. However, the Linux default scheduler is not able to satisfy the real-time requirement of the application (2 ms), and by increasing, CPU usage load, it drastically fails.

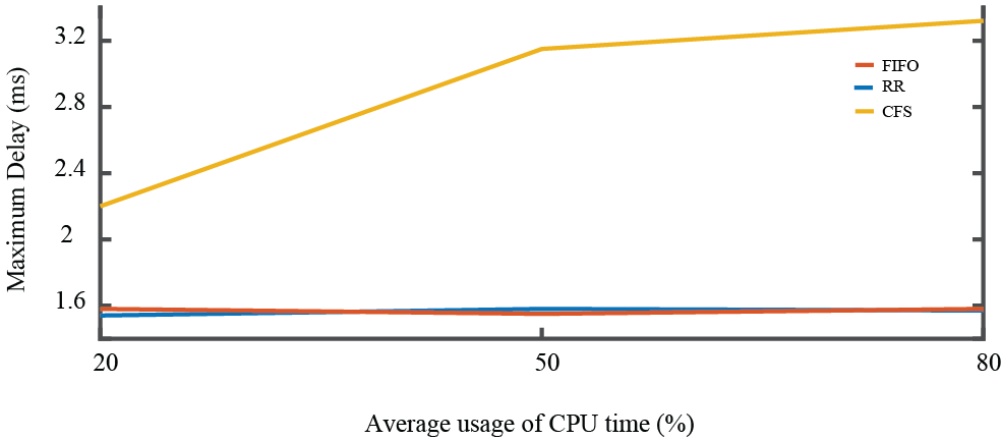
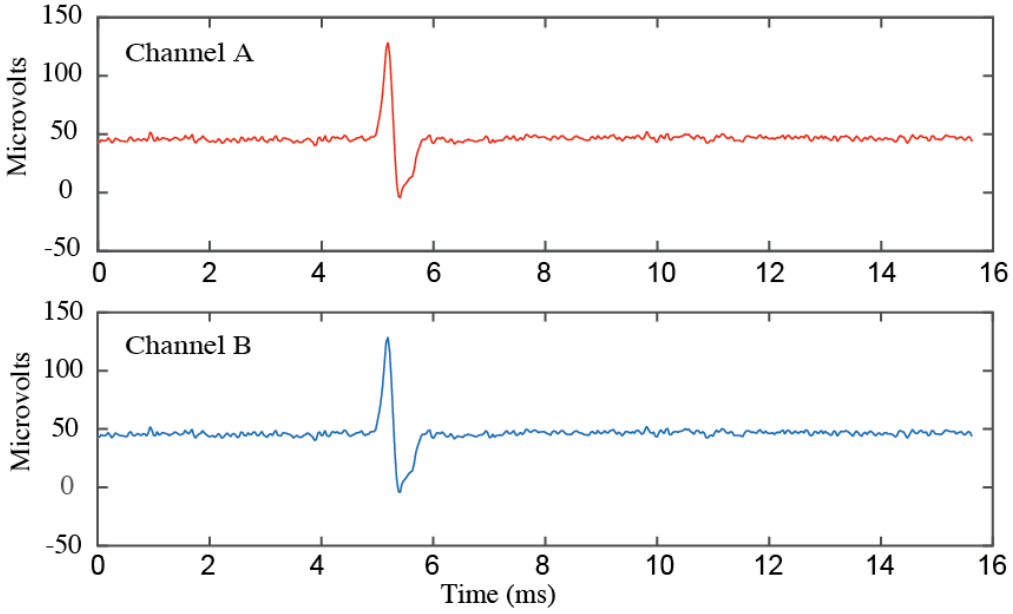


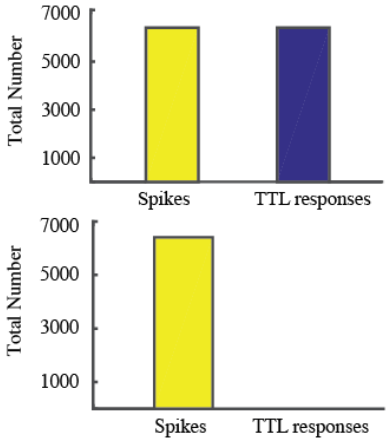
Figure 9. FIFO and RR policies show more deterministic latencies in the SPP, which are always lower than 1.6ms regardless of CPU usage load. On the other hand, Linux default scheduler (CFS) is not able to satisfy the real-time requirement of the application.

Channel Specificity

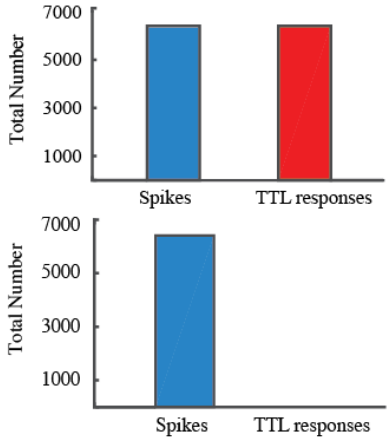
To make sure that the triggering is unique to any given channel where the waveform was identified, we count the number of TTL outputs for each channel in both HPP and SPP. Since the user sets the template for comparison separately on each channel, and thereby feeding the same waveform into two channels and setting a template for one of them, will only trigger the stimulation for that particular channel (Figure 10).



a) The same artificial waveform was fed into the system on both Channel A and Channel B, but the template was only applied to Channel A



b) HPP: On the channel selected for template matching, all instances of the waveform input triggered a TTL pulse, where on the channel not selected for template matching, the same waveform input never triggered a TTL pulse



c) SPP: On the channel selected for template matching, all instances of the waveform input triggered a TTL pulse, where on the channel not selected for template matching, the same waveform input never triggered a TTL pulse

Figure 10. The HPP and SPP only match incoming waveforms to templates on specified channels.

Waveform Identification

One of the main challenges is waveform specificity; the triggering should be specific to the selected waveform.

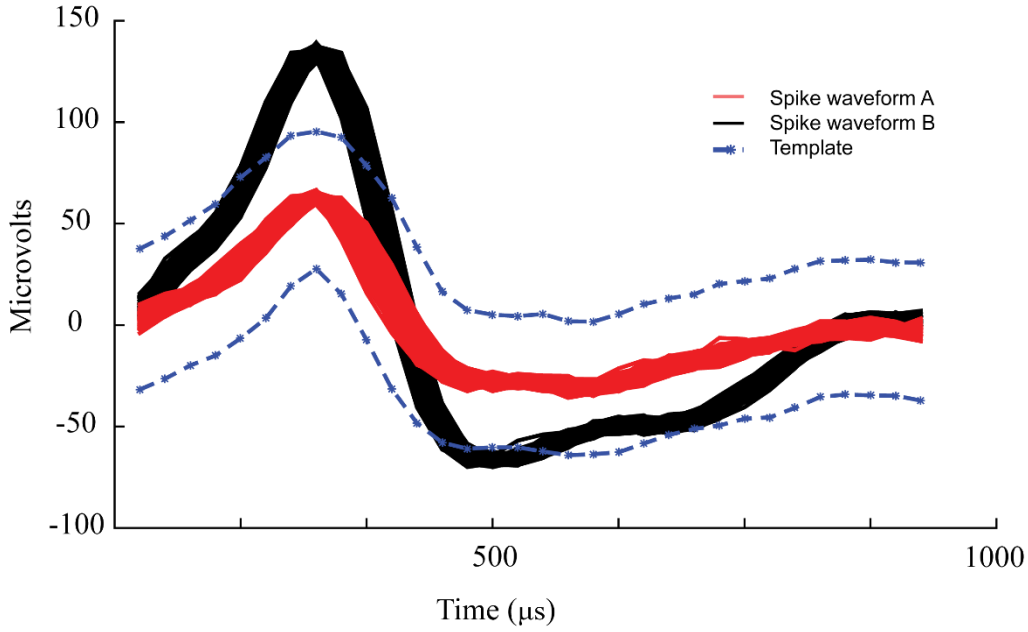


Figure 11. Comparing multiple examples of two different spike waveforms to the full 64-point template shown will trigger spikes for the red waveform but not the black.

One of test we ran was streaming multiple traces of two waveforms (Figure 11: Shown in red and black), and comparing the waveforms against a template for one of the waveforms (Figure 11: red waveform (in blue)). Typical templates consist of 64 values, but you can adjust the threshold at which a waveform is considered to match the template by changing the number of values that are required to match.

Changing the number of template values to be matched will slightly change the processing time (Figure 12a). However, even with all 64 values, the average is under 0.08 ms for the HPP and 1.5ms for the SPP. Furthermore, in the SPP, the width of a TTL pulse

is about 5 microseconds, meaning each individual channel-template match requires an extra 5 microseconds to output a TTL pulse. Therefore, the number of possible channel-template comparisons is not only limited to the hardware configuration, such as number of available GPIO on the board (~22), but is also limited by the time it takes to respond for each match. The typical sampling rate of the DLSX is about 32 kHz, with both platforms receiving samples every 32 microseconds, and thus increasing the number of comparisons which may lead to loss of data in the SPP. The number of possible comparisons in HPP is not limited by this factor, since HPP can output any number of TTL pulses in parallel.

However, changing the number of values to be matched will also change how well the system differentiates between similar waveforms (Figure 12b).

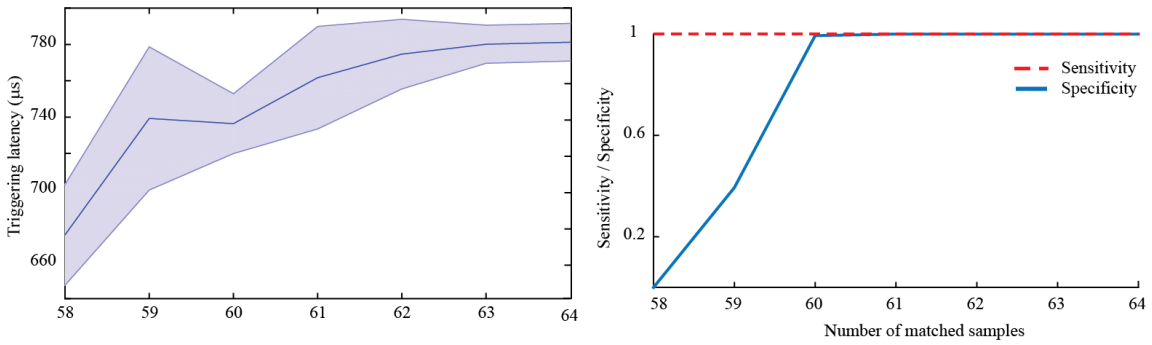


Figure 12. Template-matching criteria can be modified to optimize processing time and waveform specificity.

For example, when comparing the two waveforms shown above, when comparing only 58 values of both waveforms will match (as indicated by the low specificity value in blue), but when matching 60 or more template values, only the red waveforms will match.

In an experimental setting, this value could be adjusted based on noise or the need to differentiate between similar waveforms.

Threats to Validity

The user is required to interact with SS3D in order to extract the best template out of the spike waveforms; in some cases, the user might encounter trouble when trying to isolate the waveforms of two different neurons that have similar waveforms. This is common when a single electrode is used for recording. Also, since delivering a stimulus can cause unusual changes in spike waveforms, the user should constantly monitor the system for capturing changes and updating the templates. In addition, any high-frequency and/or low-frequency noise may directly affect the application's performance. The HPP and SPP are both dependent on the DLSX and its signal’s quality and noise filtering features.

CHAPTER SIX — CONCLUSIONS

We have cleared one of the major technical hurdles on the path to developing a high-throughput connectivity screening system: we can reliably match spike waveforms and trigger stimulation in under two milliseconds.

One of the chief technical challenges in developing this system is the ability to detect spikes and trigger electrical stimulation at low latencies. In our experience using the collision technique to identify neurons projecting from the Frontal Eye Field in prefrontal cortex to visual area V4, axonal delays are typically 2-4 ms. Our current system can match a spike template and trigger stimulation in ~2 ms, allowing for targeted collision testing of selected neurons. Future work will incorporate this triggering mechanism into a complete hardware-software package for efficiently screening neural connectivity. A major distinction appears while examining the latency result in the SPP and HPP. The HPP is a hardware platform that provides a real-time capability at both fabric level and CPU level, whereas the SPP relies on real-time capabilities of the operating system running on its CPU. Although both platforms can satisfy real-time constraints defined in the requirements section, the HPP can provide more deterministic latency for a real-time application; this makes HPP more suitable for hard real-time as well as soft-real-time usage. On the other hand, the SPP, by relying on real-time capabilities at the software-level, is a good candidate for soft real-time applications. When it comes to selecting one of the platforms for our application, many other factors should be considered before making a conclusion.

Developing on HPP requires longer lifecycles because this platform is difficult to program, debug and test. Besides, the HPP uses much more expensive hardware compared

to the price of the SPPs hardware. This could be a limiting factor for widely adopting HPP for use in research labs. Also, the SPP provides cloud-based development and debugging environments, while HPP is limited to the traditional cross-compile development and debugging environment. The cloud-based development and testing is of particular importance for our end-users because most of the signal processing devices in neurophysiology are physically located inside a vet lab environment with biosafety level of 2 or higher, requiring the experimenter or developer to wear eye and facial protection while working in the lab: conducting an experiment or testing a device on the cloud is considered an advantage for this platform. Furthermore, the SPP is using Pulsar Linux as its operating system which allows the developer to use "containers" as the most modern tools for software deployment. A container is a light-weighted virtual environment which isolates a set of hardware resources and processes for an application. Thus, the software package can be delivered to the end-user as a container. Delivering this solution in container guarantees security in runtime by limiting system calls and controlling resources and portability with a single dependency to the kernel application binary interface. Access management and preventing unwanted changes to the DLSX are other issues that should be addressed. Although there are no controlled ways to prevent alternations in the DLSX for the HPP user, the SPP is entirely isolated. The SPP user does not have the capability to make any changes in the DLSX's functionality since SPP uses the second data port of the DLSX which is designed only to stream data from the DLSX to a host. Thus, the functionality of the DLSX cannot be affected by SPP. Finally, scaling the solution to cover more channels and more templates in the HPP is limited to its hardware resource, sampling

rate and timing requirement of the applications, yet the SPP can quickly scale to process more channels or templates. The SPP is using a cloned MAC address, and a switch can be used to multicast the packets to multiple destinations.

Clearly, there is not a single winner platform, and each one has some advantages and disadvantages, but selecting a platform over the other platform should be done carefully after examining all the features provided by the platforms and considering the requirements of the application.

Future Directions

Although we have achieved our primary goal of being able to quickly identify a waveforms and trigger stimulation in about two milliseconds with either platforms, making a commercially available product that can easily be incorporated into their neurophysiology experiments will requires further improvements. First, the spike match and stimulation triggering algorithm can be improved, potentially by adding an extra step to remove noise (i.e., high pass and low pass filter). Any new feature, such as filtering, should be carefully tested before deployment, since filtering is naturally considered a process with a significant overhead, and more importantly, it can introduce a non-deterministic delay to the application. The HPP is the more proper choice for implementing a filter since it can use FPGA as a hardware accelerator. Second, using tetrode arrays to improve isolation quality and expand simultaneous screening capabilities, will make this product ready to be used for neurophysiology experiments. And lastly, the current solution comes with multiple command line interfaces and building a single graphical user interface that communicates

with all the components to allow the experimenter to easily isolate neurons, create templates during data acquisition, test selected neurons for connectivity, and perform direction-identification test (Collision Test) will dramatically improve user experience with the solution.

REFERENCES CITED

[1] A. Szücs, F. Berton, T. Nowotny, P. Sanna, and W. Francesconi, “Consistency and Diversity of Spike Dynamics in the Neurons of Bed Nucleus of Stria Terminalis of the Rat: A Dynamic Clamp Study,” *PLOS ONE*, vol. 5, no. 8, p. e11920, Aug. 2010.

[2] K. S. Rockland and D. N. Pandya, “Laminar origins and terminations of cortical connections of the occipital lobe in the rhesus monkey,” *Brain Res.*, vol. 179, no. 1, pp. 3–20, Dec. 1979.

[3] D. J. Felleman and D. C. Van Essen, “Distributed hierarchical processing in the primate cerebral cortex,” *Cereb. Cortex N. Y. N 1991*, vol. 1, no. 1, pp. 1–47, Feb. 1991.

[4] R. Lemon and A. Prochazka, *Methods for neuronal recording in conscious animals*. Wiley, 1984.

[5] S. N. Baker *et al.*, “Multiple single unit recording in the cortex of monkeys using independently moveable microelectrodes,” *J. Neurosci. Methods*, vol. 94, no. 1, pp. 5–17, Dec. 1999.

[6] M. Paré and R. H. Wurtz, “Monkey Posterior Parietal Cortex Neurons Antidromically Activated From Superior Colliculus,” *J. Neurophysiol.*, vol. 78, no. 6, pp. 3493–3497, Dec. 1997.

[7] C. M. Gray, P. E. Maldonado, M. Wilson, and B. McNaughton, “Tetrodes markedly improve the reliability and yield of multiple single-unit isolation from multi-unit recordings in cat striate cortex,” *J. Neurosci. Methods*, vol. 63, no. 1–2, pp. 43–54, Dec. 1995.

[8] S. Takahashi, Y. Anzai, and Y. Sakurai, “Automatic sorting for multi-neuronal activity recorded with tetrodes in the presence of overlapping spikes,” *J. Neurophysiol.*, vol. 89, no. 4, pp. 2245–2258, Apr. 2003.

[9] P. A. Laplante and others, *Real-time systems design and analysis*. Wiley New York, 2004.

[10] D. Koch, F. Hannig, and D. Ziener, *FPGAs for Software Programmers*. Springer, 2016.

[11] J. D. Owens *et al.*, “A survey of general-purpose computation on graphics hardware,” in *Computer graphics forum*, 2007, vol. 26, pp. 80–113.

[12] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, “Accelerating Compute-Intensive Applications with GPUs and FPGAs,” in *2008 Symposium on Application Specific Processors*, 2008, pp. 101–107.

[13] S. Rostedt and D. V. Hart, “Internals of the RT Patch,” in *Proceedings of the Linux symposium*, 2007, vol. 2, pp. 161–172.

[14] T. N. B. Anh and S.-L. Tan, “Real-time operating systems for small microcontrollers,” *IEEE Micro*, vol. 29, no. 5, 2009.

[15] “MinnowBoard.org | MinnowBoard Turbot Technical Specifications.” [Online]. Available: https://www.minnowboard.org/. [Accessed: 01-Mar-2017].

[16] “Wind River Pulsar Linux.” [Online]. Available: https://www.windriver.com/products/operating-systems/pulsar/. [Accessed: 12-Mar-2017].

[17] S.-T. Dietrich and D. Walker, “The evolution of real-time linux,” in *7th RTL Workshop*, 2005.

[18] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel: from I/O ports to process management*. O’Reilly Media, Inc., 2005.

[19] J.-P. Stauffert, F. Niebling, and M. E. Latoschik, “Reducing application-stage latencies for real-time interactive systems,” in *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2016 IEEE 9th Workshop on*, 2016, pp. 1–7.

[20] A. Garg, “Real-time Linux Kernel Scheduler,” *Linux J*, vol. 2009, no. 184, Aug. 2009.

[21] R. Love, *Linux system programming: talking directly to the kernel and C library*. O’Reilly Media, Inc., 2013.

[22] H. Härtig, M. Hohmuth, J. Liedtke, J. Wolter, and S. Schönberg, “The Performance of u-kernel-based Systems,” in *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, New York, NY, USA, 1997, pp. 66–77.

[23] “Wind River Helix App Cloud.” [Online]. Available: https://www.windriver.com/products/helix/app-cloud/?appref=pulsar. [Accessed: 12-Mar-2017].