HYBRIDIZING CELLULAR AND BEHAVIORAL NEUROBIOLOGY WITH MODERN ENGINEERING TOOLS: MICROELECTRONICS, MICROFABRICATED DEVICES, AND SOFTWARE SOLUTIONS FOR PHYSIOLOGY

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

In Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Gus Kaderly Lott

May 2007

HYBRIDIZING CELLULAR AND BEHAVIORAL NEUROBIOLOGY WITH MODERN ENGINEERING TOOLS: MICROELECTRONICS, MICROFABRICATED DEVICES, AND SOFTWARE SOLUTIONS FOR PHYSIOLOGY

Gus Kaderly Lott, Ph. D.

Cornell University 2007

Trained as an electrical engineer and physicist, I have learned the language of the modern neurobiologist and analyzed the state of experimental metrology (measurement tools) and laboratory based undergraduate education in the field (Neurobiology & Behavior). As a result of this examination, I develop solutions that offer a new kind of resolution and accessibility to physiological preparations from the sub-cellular to organismal level. My goal is to create an instrumentation toolset that trivializes the solutions to many of the current questions in neurophysiology and enable the experimenter to access higher resolution and new kinds of data about a system. My goal is to allow for new kinds of questions to be asked and answered. This represents the essence of mission of biophysics: Bringing the tools and methods of modern physics to revolutionize studies of biological systems.

I present three distinct projects which illustrate my findings. These projects bring the majority of modern electrical engineering tools and methods to focus on physiological questions in neural systems. The first project illustrates how a time critical microcontroller driven data acquisition and instrumentation system revolutionizes behavioral analysis in model organisms. The second projects brings the tools of the worlds foremost biophysical nanotechnology center (Cornell's

Nanobiotechnology Center, NBTC) to the design of a new kind of polymer substrate microelectrode structure capable of implantation and extracellular recording from sub-millimeter processes in intact animals. The third project describes a software data acquisition and analysis program capable of acquiring data and analyzing events in a variety of useful ways.

These projects should be viewed as responses to questions about the method, itself, of making observations and analysis in neurophysiological research and education environments. These projects describe tools that are currently being applied at a variety of institutions including several affiliates of Cornell to produce dramatic results. It is my goal to bring the experimental mind of yesterday's behavioral and cellular neurobiologist into the realm of today's biophysical engineering tools.

BIOGRAPHICAL SKETCH

Gus Kaderly Lott, III was born August 17th, 1979 in Corpus Christi, Texas to Mary Christenberry and Gus Kaderly Lott Jr. Gus has one sister, Melissa, born December 7th, 1982. As the son of a US Navy intelligence officer, Gus traveled the continental United States attending K-12 educational institutions located in the Southeast, the East Coast, and the West Coast. After graduating from Monterey High School in Monterey, California at the age of 16, Gus attended Auburn University in Auburn, Alabama. He achieved two, separate, bachelor's degrees at Auburn in Electrical Engineering and Physics. He graduated from Auburn University, in 2001, Cum Laude and with the Deans Medal for most outstanding Physics Senior in the College of Science and Mathematics. In 2000, Gus Jr. and Mary founded Yarcom Inc., an information security and communications engineering firm and Gus III was retained as an employee for consulting in instrumentation and measurement.

In the fall of 2001, Gus chose to attend Cornell University to pursue his doctorate in the field of Biophysics.

ACKNOWLEDGMENTS

I first must acknowledge my family. My mother and father and sister have been constants in my life through an existence consisting of much change. They have always represented a "home base" where I could find unwavering support in any endeavor I put my mind to.

I would also like to thank my Special Committee at Cornell University consisting of Dr. Harold Craighead, Dr. Manfred Lindau, and Minor Field Member Dr. Ron Hoy. Dr. Craighead and Dr. Lindau provided an invaluable springboard from which I launched my PhD path. Through Dr. Craighead's research group, I met many important figures in my doctoral experience and formed a foundation of experience with microfabrication techniques that facilitated my future work. Particular influences in the Craighead Group were Andrea Turner and Conrad James. They exposed me to world of electrically active biology and launched me into my research project in the department of Neurobiology & Behavior in the laboratory of Dr. Ron Hoy.

In the Hoy Lab, I found an environment directed by a visionary biologist (Hoy) who understood the value of enabling people that could be driving figures in this period of biophysics integration. My main compatriot in the Hoy Lab who helped me found my work and find direction was Andrew Spence for my first few months as part of the group. I still have the blue toolbox that I saw him carrying when he was featured in a Cornell Engineering publication my first months on campus.

Many professors at Cornell have molded my experience and helped to train me in the language of the neurobiologist. These professors include Dr. Bruce Land, Dr. Bruce Johnson, and Dr. David Deitcher. Courses taught by these professors opened, to my mind, the fundamental nature of what we are as humans. These professors have a mastery of the material they present to student and provide education unparalleled in educational institutions in the world.

I thank the Nanobiotechnology center for making me their first funded student in 2001 and for the support of Dr. Barbara Baird as both the directory of the Nanobiotechnology Center (NBTC) and administrator of the Molecular Biophysics Training Grant which funded 3 years of my graduate experience. I would also like to thank Anna Waldron and the Educational Outreach arm of the NBTC for providing me with the opportunities to share the sciences of biology and physics with those in the Central New York area.

Finally, I would like to acknowledge existence (Brahman or whatever) of which we are all expressions as flowers from a stem. I acknowledge the illusory nature of consciousness and choice and giggle at my ego as I continue to enjoy and experience my role in the cosmos, however small it may be. I find continuous amazement in the fact that something exists instead the alternative and I marvel at how the field of modern neuroscience and biophysics are combining to further connect us to more of the truth about the nature of being.

Satchitananda – True Being, Pure Consciousness, and Bliss

Tat Tvan Asi – Thou art That, Thou art God

Aham Brahmasmi – I am He

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1:
INTRODUCTION

1.1 Analyzing Neurophysiology and Constructing an Approach

The explosion of the information and communications industries has driven sensors and engineering tools to scales comparable to those of individual subcomponents of biological systems. The application of these tools to biological systems increases as mathematical and electronic tools of the modern engineer evolve into more complex and smaller packages. Both public and private funding of hybrid engineering/biology programs and centers and institutes, such as the Cornell Nanobiotechnology Center (NBTC), facilitates interdisciplinary communication between biologists (about their systems of interest) and engineers (about their tools and methods).

The same mathematical tools applied to fundamental physics, communications and control system design and analysis are now part of biological metrology. Specifically when analyzing the structure and function of a biological neural network, the mathematical tools of convolution and basis function analysis offer a means by which the underlying structure and physical circuitry may be inferred. The tools of the modern engineer offer resolution on a spatial and temporal level comparable to individual subcomponents and discrete processes of biological systems (such as action potentials) from a cellular to organismal level. But the implementation of these new biophysical technologies into the fields associated with neurobiology is slow coming. Education systems for new biologists are slowly but surely adapting to the challenge of training new experimental biologist in engineering methods. It is into this

adaptation process to which I direct the elements of the biophysicist's engineering arsenal.

This dissertation presents three distinct applications of engineering tools to current problems in cellular, behavioral, and educational physiology. The first project involves the introduction of a variety of micro-electronics to a study of the phonotactic behavior of the parasitoid fly, *Ormia Ochracea*. This work provides measurement resolution greater than time scales of propagation through individual units in the underlying physical neural network (sub-millisecond resolution). The second application involves an implantable polyimide electrode array designed to monitor extracellular action potentials propagating along individual axons in a peripheral nerve bundle in the red swamp crayfish, *Procambarus Clarkii*. The third project is a software system designed for physiological data acquisition, stimulation, and event identification and analysis in a variety of target systems.

These new applications are designed to demonstrate a bridge between engineering mathematical methods, modern electro-mechanical devices, and neurobiological systems of interest. These distinct projects present tools that offer glimpses of neural architecture at the organismal and cellular levels respectively through input space construction, system response analysis, and basis transform.

1.2 RISC Instrumentation in Network Level Neurobiology

The objective of this project is to build, from the ground up, an instrumentation system capable of spectral analysis of the response of a biological system to a broad input space and thus infer the underlying equivalent structure (as in a Thevenin equivalent circuit model) of the information processing network. *Ormia Ochracea* is a model system for a unique form of directionally sensitive hearing (Robert 1992, Mason 2001). *Ormia*'s ear and associated neural system respond specifically to cricket chirps in the wild (target of their parasitoid behavior) (Robert 1998 & 1999).

Much is known about the underlying architecture of the afferent auditory and motor neurons in the peripheral nervous system of the fly (Oshinsky 2002), but the majority of decision making and high level signal analysis resides amongst the massive information process in the fly's brain.

I present a time critical, calibrated metrology driven by a cycle accurate microcontroller processing core. From stimulus generation and presentation to high throughput response recording, this project demonstrates a total experiment system capable of automated control of all factors of behavioral response analysis. My work includes an instrumentation system with a temporal resolution exceeding that of individual units within the neural architecture. This system is capable of crafting and presenting an input space of signals with complexity and resolution approaching that of possible internal neural pathways.

My work includes a complicated signal generation, presentation, and data acquisition program written in MATLAB (Mathworks Inc.) that interfaces with a custom programmed "reduced instruction set computer" (RISC) processor. This RISC core provides a computation center capable of complete experimental control. The microcontroller core provides nanosecond accurate device synchronization and protocol bridges between experimental hardware and the MATLAB interface.

This project automates the construction of an input space of synthetic cricket chirps varying in a multitude of dimensions such as amplitude (sound pressure level), center frequency, pulse duty cycle, signal duration, and presentation angle. This system presents signals to the female fly through a calibrated audio system either as individual signals, or as choice experiments. My custom-designed motion tracking sensor system, based on optical computer mouse technology, monitors the resulting taxation response. My work provides an observational environment presenting large volumes of data for individual points in the input space in random or controlled order

with random inter-stimulus delays to counter predictive effects of the fly's nervous system.

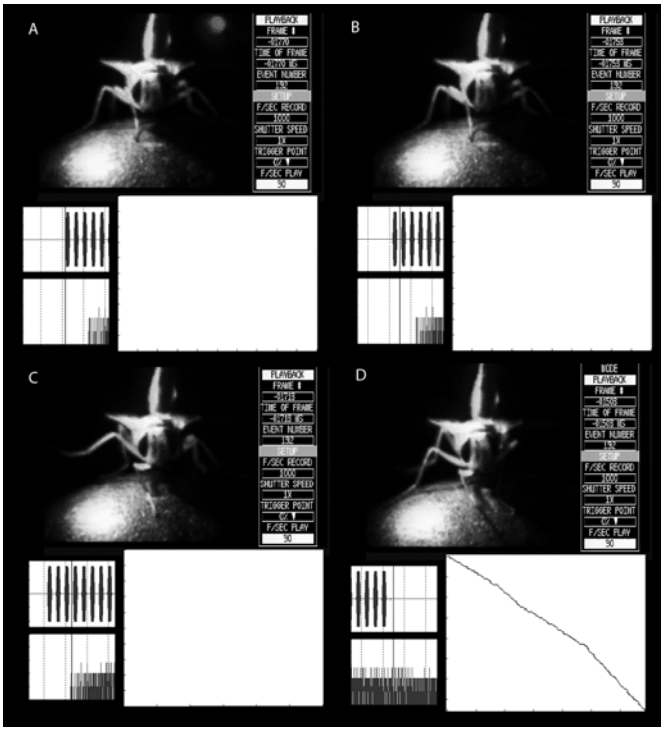

Figure 1.1 – A series of high speed video frames of the taxation response of *Ormia Ochracea* to a computer generated cricket chirp with characteristics taken from a specifically constructed input space. The associated instrumentation data indicates the fly's response with temporal and spatial resolution beyond that of the animal's actual response capabilities.

This project presents several factors to characterize the array of input space responses to the resulting taxation information. We track the onset latency, track angle, velocity, and duration of the walking response of the fly. Given this highly accurate data presentation and acquisition system, this project demonstrates the new capability to correlate these taxation "output" parameters with the signal "input" parameters and generate an equivalent circuit model for the underlying neural network of the fly.

1.3 Polyimide Micro-Electrode Array in Cellular Neurobiology

CMOS electronic device fabrication often utilizes the photo-patterned and curable polyimides as an insulating material. Polyimide applications include biologically inert (Haggerty 1989, Richardson 1993, Seo 2004) shunts and coatings for other materials in macroscopic medical science. A broad spectrum of researchers and medical professionals incorporate the polymer itself into modern medical microdevices. The most commercially successful micro-application to date is as a substrate material in some versions of the cochlear implant (Shamma-Donoghue 1982, van der Puije 1989). Polyimides are an essential part of the first truly successful and widely clinically applied neural prosthetic device.

This dissertation presents a method for rapid prototype design of a multi-channel cuff microelectrode array capable of connecting to small diameter neural fibers in peripheral nervous systems. My polyimide device acts as a self contained electrophysiology tool offering integrated recording and reference electrodes along with micron-scale features capable of constraining tissue elements in an intact system similar to the way a classical cuff electrode device constrains larger tissue bundles (Veraart 1998, Naples 1988, Popovic 2000, Rodriguez 2000, Tarler 2004, Yoo 2005). Along the same theme as the total instrumentation system developed for *Ormia*, this project encompasses a site of interest in an arbitrary neural process and offers propagation information about action potentials at several locations along individual neurons. My design allows for multiple recording elements to monitor action potentials as they propagate into a site of interest and to track the resulting electrical activity as they propagate out of a target environment.

Photosensitive polyimide allows for rapid production of novel device geometries for applications to arbitrary systems of interest. My specific application is to the third root offshoot of a tail segment ganglion in the red swamp crayfish,

Procambarus Clarkii. The nerve bundle consists of six distinct neural units with varying physical geometries and functionalities (Kennedy & Takeda 1965). The nerve innervates a thin sheet of muscles responsible for posture control on the ventral surface of the crayfish tail and offers several accessible recruitment and suppression schemes for specific neurons while a few member neurons maintain tonic activity. This signal rich environment offers an ideal testing ground for a sub-millimeter cuff micro-device.

I designed customized analog signal conditioning circuits coupled with a suite of software tools written in MATLAB. My project demonstrates the acquisition of simultaneous action potential recordings at multiple electrode sites spaced along a two millimeter segment of tissue in the crayfish. This novel electrode system for sub-millimeter neural processes presents a state of the art contribution to the field of neural prosthetic devices and acts as a demonstration of engineering input response analysis methods to an electrophysiology problem at the cellular level.

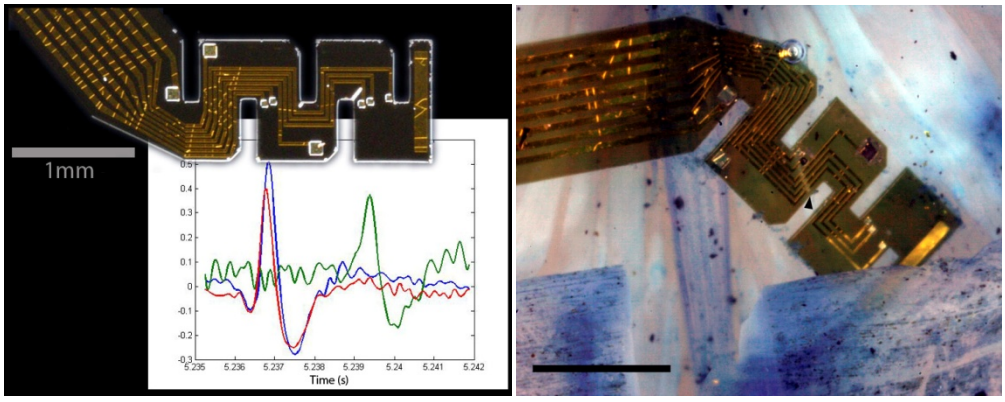

Figure 1.2 – One action potential is illustrated propagating through the polyimide cuff electrode array and is superimposed to illustrate the signal path through the physical device (Top). The array itself is also shown implanted in a live prep stained with methylene blue (bottom) to illustrate relative neural process size. The third root offshoot is indicated by the arrow partially woven through the device.

1.4 Software to Enable the Modern Physiologist

Personal computers have been actively incorporated into science for quite some time now, but the border between personal computers (generally used for offline or theoretical analysis) and data acquisition systems (generally with built-in or highly specialized and elaborate interfaces) has been quite tedious to break down. Over the past decade, the complexity and quality of data acquisition interfaces has increased in something of a parallel evolution with the capacity of the scientific programming package MATLAB (Mathworks Inc.). As the "data acquisition toolbox" in MATLAB has developed to include more interfaces, including the standard sound card port on common PCs and laptops, the processing power of modern PCs has increased to a point where it is possible to create real time analysis programs that allow the user to visualize the behavior of systems with high resolution (sample periods on the orders of tens of microseconds).

I present a system which revolutionizes the modern physiologist's data acquisition and analysis toolset. The program is designed to carry out the following functions:

1. Acquire and visualize multi-channel voltage data in real time from a variety of supported hardware data acquisition interfaces. A Software Oscilloscope.
2. Deliver a pulsed stimulus and trigger waveform that can be utilized in a variety of experiments.
3. Stream the data to disk for offline analysis.
4. In an analysis mode, detect events using parameter thresholds and conduct analysis calculations on the raw event waveforms.
5. Use the detected events to conduct an event triggered with a target simultaneous source to enable cross and auto-correlations.

6. Enable report generation of results with publication quality graphics

This software was inspired by much of the MATLAB work employed for the fly and crayfish projects described above. The project was originally designed for immediate application in the most advanced and well funded neurophysiology lab course in the world (Cornell's BioNB 491). As the program was phased into the lab course, researchers in the Neurobiology and Behavior department began contributing to the program features. The software now represents an opus of MATLAB programming which enables experimentation on systems in all ranges of physiology from molecular and cellular to organismal and behavioral biology.

The software is currently being applied in several research labs at Cornell and is slated for expansion into use at other institutions from the high school to university research levels as freeware.

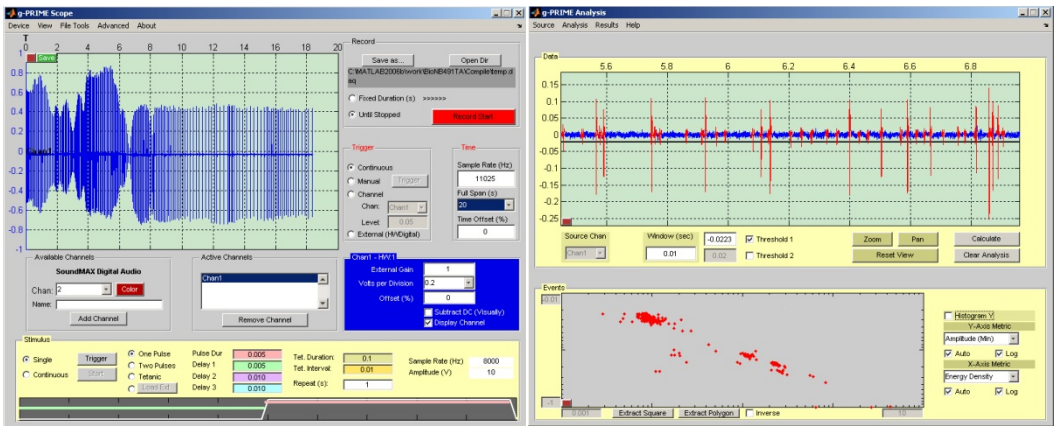

Figure 1.3 – Software designed for physiology recording and stimulation in labs from high schools to university research. The software also implements real time event detection, characterization, and sorting for visualization of physiological responses in real time. This allows the student/researcher to focus less on the interface and more on what is actually happening in the biology.

1.5 Dissertation Objectives

This dissertation discusses the history and applications of small scale peripheral neural interfaces. In addition, I address pertinent information on the methods of neural architecture inference in biological systems and present methods for the construction of two distinct instrumentation systems for neurophysiological applications. I show successful construction of a toolbox of relatively non-invasive devices that infer an effective mapping of underlying architecture of the unique information processing systems that are biological neural networks. This will be contrast to established architecture mapping methods involving highly invasive tools.

The question that I propose to address is not a specific question about a fact of a target system's physiology, but how these specific questions are asked at both the graduate, post-doctoral, and research associate level in modern university research labs. As a result of asking these questions, I hope to present many key solutions to instrumentation and analysis problems in modern neurobiology. These biophysical solutions trivialize many questions posed by physiologists and will enable them to ask new kinds of questions about their systems given access to new resolutions and kinds of data.

The objectives of my work are entirely focused on the design and communication of engineering tools to modern biologists. My goal is to design novel tools for future problem solving in biological systems as well as to develop more efficient means by which the biological scientist may solve their currently stated problems. I will illustrate the means by which several current problem solving tools may be enhanced to offer a new level of resolution in biological systems.

My communication of engineering tools and techniques is extended to individuals from High School to Post Doctoral education backgrounds. My work focuses on the quality, character, and volume of data acquisition for biological system

problem solving. I will conclude with an illustration of modular electronics and programming skills capable of providing a major enhancement of the modern physiologist's data acquisition experience. I will then discuss future goals and implications of these tools.

CHAPTER 2:

AN INEXPENSIVE SUB-MILLISECOND SYSTEM FOR WALKING MEASUREMENTS OF SMALL ANIMALS BASED ON OPTICAL COMPUTER MOUSE TECHNOLOGY

2.1 Target: Ormia Ochracea

Field crickets from Hawaii to Texas to Louisiana are met with a complex set of selection pressures while seeking a mate. The cricket call will invite both mate and parasite to their location. The female *Ormia ochracea* will use her unique directionally sensitive micro-ear to detect hosts and parasitize the target crickets with larvae (Robert 1992, Mason 2001). When presented with a complex field of various targets, the fly will make choices between host targets based on the character and quality of the chirps they emit (Cade 1975, Muller 2001). Ormia will tend to land nearby the sound and walk towards the preferred host.

Figure 2.1 – A female *Ormia* on the Treadmill (Ping Pong Ball with dots)

The underlying neural mechanics of the auditory nervous system are known to a degree (Oshinsky 2002). The complex decision making pathways beyond the afferent nerves represent a complex processing network designed to drive the fly towards a preferred target. Given metrology capable of resolution on time scales

similar to those of individual neuronal units, it is possible to develop a physiologically relevant map of pathway construction in the underlying central processing nervous system.

The afferent nerve paths and immediate transduction of sound to neural signals has been well characterized in *Ormia* (Robert 1998 & 1998, Oshinsky 2002). The structure consists of a mechanically coupled pair of tympana which create an amplified representation of an otherwise minute intra-aural time difference. The tympanic membranes are each connected to sensory nerve bundles by a stiff rod-like cuticular apodeme (Robert 2000). The apodemes couple vibrations into sensory neurons connected to auditory neurons in the two bulba acusticae (BAc). The bulba acusticae is a nerve bundle consisting of approximately 100 neural cells with stretch receptor processes extending to the apodemes, and afferent interneurons extending approximately 50 axons into the thoracic ganglion.

These stretch receptor neurons connect directly to the apodeme structure (Oshinsky 2002). Auditory afferent neurons branch from the bulba acusticae to the thoracic ganglion and innervate other interneurons that branch through the neck and enter into the brain. There are millions of neurons in the fly brain forming networks capable of predicting chirps, selecting an optimal target from a field of stimuli, and determine other components of the nature of the sound source. Sound information could traverse pathways consisting of dozens or hundreds or thousands of pathways before a control signal for muscle activation.

The actual physical wiring of this system involves, as in all other biological neural networks, neurons with chemical or electrical synapses. While action potential propagation along membranes is rapid over such short distances as those found in the fly brain (i.e. several meters per second), chemical synapses will always account for quantized propagation delays on the order of 250μs to several milliseconds. We have

begun our analysis of the fly brain by raising the sensitivity of our metrology to time scales on the order of these events. We can begin to create a systems analysis tool capable of correlating both the response latencies and response characters (including duration, magnitude, and a variety of other metrics) of animal behavior with a prefabricated input space. This system is a first step towards a non-invasive map of a model system neural network.

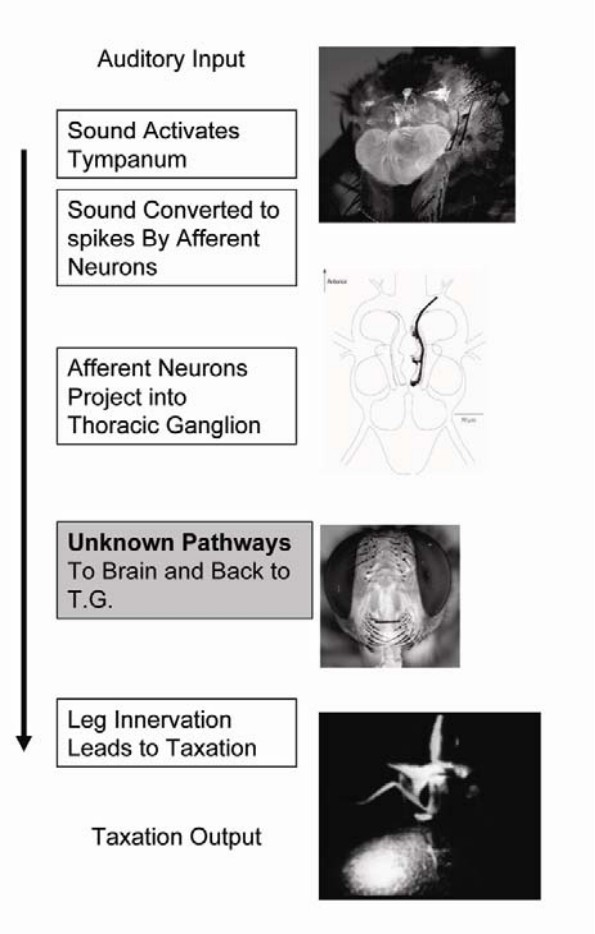

Figure 2.2 – The current understanding of *Ormia*'s neural network construction. Pathways are well explored from afferent nerves to the thoracic ganglion and from the ganglion to leg innervation, but the highly complex brain pathways are unknown. This treadmill system is designed to produce an equivalent circuit model for the information processing structure of the brain of a variety of model systems such as this fly.

The challenge of measuring phonotaxis (movement engendered by an auditory stimulus) has yielded several innovative technological solutions over several decades. These have included closed-loop measurements, where freely-moving animals, either walking or flying, can experience changes in sound intensities, and open-loop measurements, where tethered animals do not experience sound intensity changes as they move in relation to the sound source. Closed-loop measurements have included free walking or flying in open arenas, e.g. (Murphey and Zaretsky 1972, Pollack and Hoy 1979, Ramsauer and Robert 2000), while open-loop methods have involved tethered walking on a rotating y-maze globe (Hoy and Paul 1973), free-walking on a spherical locomotion compensator, the Kramer treadmill, e.g. (Kramer 1976, Webber 1981, Pollack 1984, Doherty 1985), or tethered walking on a modified computer mouse or trackball (Doherty and Pires 1987, Pires and Hoy 1992a,b, Mason 2001, Hedwig and Poulet 2004). The modified optical mouse solution has proven especially useful, as its lightweight nature lets a tethered insect move the sphere, eliminating the need for expensive servo-mechanical hardware.

The modified mouse solution was originally developed by Hoy and colleagues (Doherty and Pires 1987) using a sphere (ping-pong ball) that was small and lightweight enough to be moved by a field cricket's walking motion. An updated version was later developed in the same laboratory to measure the walking motion of a much smaller and lighter animal, the parasitoid fly *Ormia ochracea* (Mason 2001). This version was excellent for resolving directional movements, but had a sample rate of only 40 Hz, thus precluding online high-speed measurements. My upgraded metrology system offers high temporal resolution (up to 2160 samples/sec) measurements of two-dimensional movements.

With a central motion data acquisition device capable of resolving events on the order of the discrete processes in the fly brain, the total system analysis package

was expanded to include calibrated audio hardware and custom software interfaces. I have written several script based programs (Matlab) designed to effectively generate and present a subset of the auditory input space for the fly. The software allows a user to conduct high volume data acquisition in a virtually hands-free environment. A range of stimuli can be queued for presentation, random inter-stimulus intervals are generated to prevent predictive taxation (learning) on the part of the fly, and stimulus response data is automatically stored to disk.

Given the relative simplicity of the fly's response, this model system offers an optimal situation for analysis of a complex biological neural network in a living brain. Over periods of hours, flies will present similar responses to the same stimulus. A single animal from a lab reared population may be presented with hundreds or even thousands of stimuli for response correlation. Simple methods of categorical perception and choice may be mapped as if one where mapping the spectral response of any kind of classical engineering system. My system offers a technological leap in automation and demonstrates an experiment paradigm indicative of the future of multi-disciplinary life science engineering problem solving.

2.2 Hardware & Firmware Design

I acquired an inexpensive optical mouse (MICRO Innovations PD430P) containing an Agilent ADNS-2610 camera chip (the Agilent trademark symbol is externally visible on the center of the optical camera lens). The 2610 camera chip was extracted from the internal circuit board of the mouse and replaced with a pin-for-pin compatible camera chip, the Agilent ADNS-2620, which has a faster frame rate (up to 2300 Hz). This allowed preservation of the optics and the geometry of the housing along with the peripheral circuit components required for operation of the camera chip (i.e. LED driver circuitry, 24 MHz crystal oscillator circuit, and power connections).

The data and control traces from the internal microcontroller to the ANDS chip were cut with a razor blade (pins 3 and 4 on the 2620). The associated data pins of the standard PS/2 connector inside the mouse were cut and soldered directly to the data port pins on the camera chip (orange wire to SCK pin 4 and white wire to SDIO pin 3 on the ADNS chip). The power connection wires on the PS/2 Connector were left as they were connected to the internal circuit board.

For the final stages of preparation, the camera chip lens was removed and three holes were drilled in the plastic surrounding the lens to allow for air flow to float our treadmill ping-pong ball. The mouse scroll wheel was removed. The circuit board, lens, and outer mouse housing were replaced and the seam between the top and bottom segments was covered with electrical tape to create an air-tight seal. A length of silicone tubing was placed in the open mouse wheel slot and sealed into place with a two-component epoxy. At the base of the mouse, above the lens, a plaster of paris mold fit to a ping-pong ball was attached, with a hole drilled in the bottom to allow for air flow and imaging. Finally, the modified mouse was glued upside-down onto a magnet to enable secure placement on metallic surfaces.

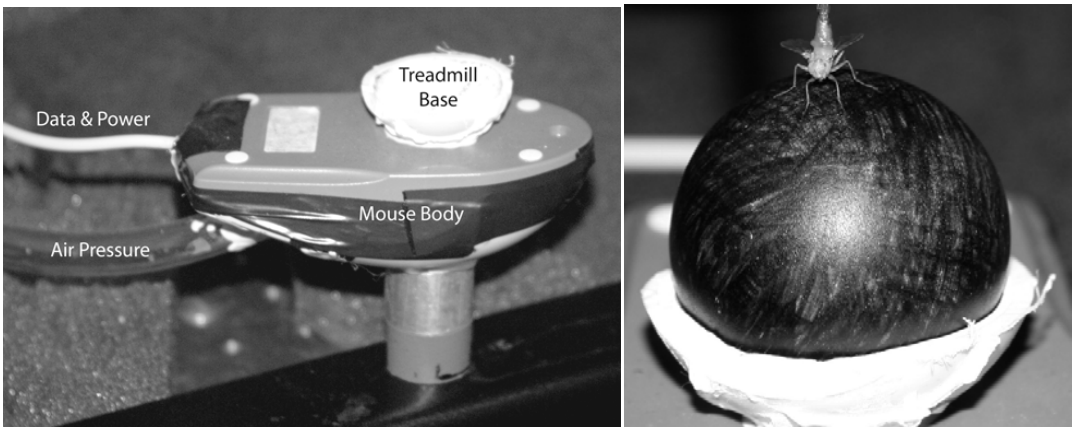

Figure 2.3 – The physical base of the treadmill. The inverted optical computer mouse is modified such that the internal microcontroller is disconnected from the optical camera chip installed in the factory. My own microcontroller is spliced into place to increase the motion sample rate of the system from 40Hz to 2160Hz.

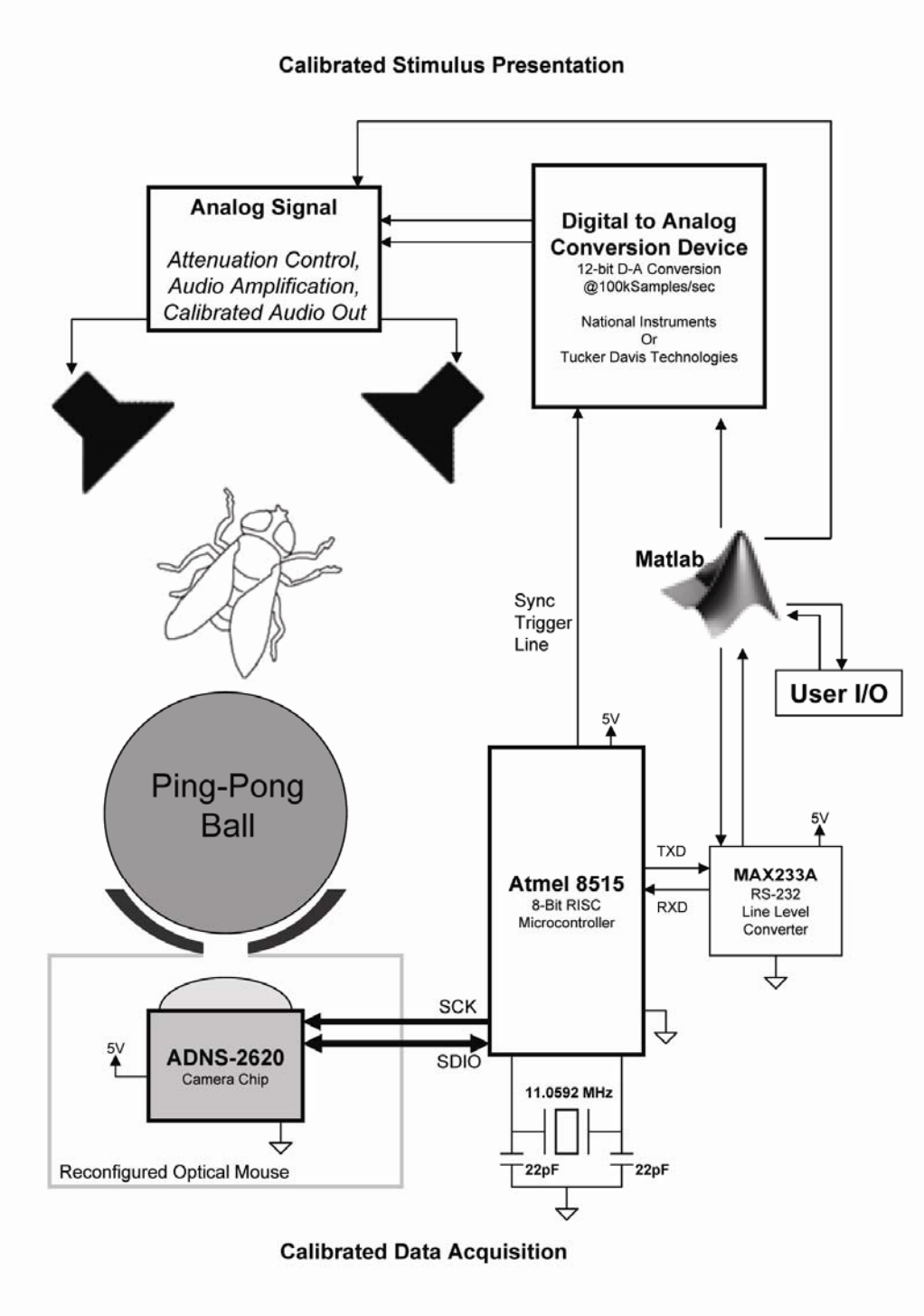

Figure 2.4 – An illustration of the entire experimental system. The user interacts with the analysis system through a MATLAB interface. Each element of the system is calibrated and controlled for accurate resolution at sub-millisecond data sample periods.

RISC Processor

The core timing unit of the data acquisition system contains an Atmel 8515 8-bit RISC microcontroller running at 11.0592 MHz. The protocol for communication with the Agilent chip is described in the chip datasheet and is implemented by toggling states of two general purpose digital I/O pins on the Atmel processor (described later). Communication with a computer was accomplished by utilizing the UART data bus built-in to the Atmel chip and run at 115200 baud through an RS-232 line level converter (Dallas Semiconductor MAX233A). The microcontroller core is built into a printed circuit board containing all the I/O connectors and signal conditioning hardware required to run the system (acquired from www.expresspcb.com).

System firmware was written entirely in Atmel assembly code using the Atmel AVR studio development environment. The firmware allows for a full protocol bridge between the user on the computer and the ADNS camera chip. Any internal register may be read or written to and there are many wrapped functions including a pixel dump. In addition to this functionality is the core data acquisition code. The sample rate may be set by the user and the transfer of data can be started and stopped by byte command sent from the PC. This allows for easy incorporation of the system into total data acquisition and instrument control programs written in programming environments such as LabView (National Instruments, Austin, TX) or MATLAB (MathWorks, Inc., Natick, MA). The sample rate of the system is controlled by an interrupt from a cycle accurate internal timer running asynchronous from the main program loop. This gives sample rate stability equal to the stability of the 11.0592 MHz crystal (50ppm error).

Firmware Outline

The process of bridging the optical mouse sensor interface and the computer user, while allowing for time-critical data acquisition and device calibration, is completely controlled by embedded code written in Atmel AVR assembly.

The Atmel 8515 8-bit RISC microcontroller offers us adequate input/output options to accomplish our task. The chip contains 32 bidirectional digital I/O pins with several alternate functionalities. Two of these pins were dedicated to the two wire serial data connection to the Agilent sensor chip. One pin was used both as a voltage source for output and a high impedance input while the other pin was used as a dedicated clock line output to drive the serial interface. Timing and communication protocols are detailed in the datasheet for the Agilent chip. One digital output pin drove a trigger line capable of synchronizing motion acquisition with stimulus presentation. Two I/O pin alternate functions created a Universal Asynchronous Receive and Transfer protocol signal which was converted (MAX233A Dallas Semiconductor) to an RS-232 standard signal capable of communicating with a user on a computer attached to the system.

Our system is able to acquire data with sub-millisecond accurate sample periods. This is accomplished by an interrupt-driven routine written entirely in assembly and activated by a 16-bit hardware timer. Organization of each command in the routine was important for timing of signal generation and communication with the Agilent camera chip. The main timer interrupt function consists of several calls and must be completed before the timer fires another interrupt request (approximately 463 μsec). The main limitations on sample rate (max achieved: 2160 Hz) are the delay required for the camera chip to prepare the movement data for both the X and Y channels (100 μsec) and the delay required to transmit those data bytes and data headers over the 115.2 kBaud serial data line to the user (approx 165 μsec). Our system thus has a

temporal resolution of 2160 Hz. The camera and lens system in the ADNS-2620 allows for a spatial resolution of 63.5 μm (400 cpi). The assembly code written to handle all functionality of our is described in appendix 1.

In the main timer interrupt function, we have formed the sequence of communications and acquisitions to minimize clock cycles and fit into our target sample rate that can generate 1 kHz bandwidth for motion detection. Both UART data transmission and preparation of the movement data on the Agilent chip run as parallel processes to the execution of the interrupt subroutine. In order to meet the sample rate requirements, the firmware initiates by requesting an X motion value from the camera chip and immediately sending the Y value from the previous sample over the serial port (initially this value is 0x00 which is an invalid motion value). As the system waits for the header and data byte to be sent to the user, the camera chip is processing a data point. When the X data point is ready and the UART transmit buffer is clear, the X value is read and the current Y motion value is requested from the camera chip. While the camera chip is processing the Y motion value, the Atmel controller is sending the X value and header that it just acquired. Byte commands for system operation, data formatting, and general instructions on system usage may be found in the supplementary web material.

The data is formatted as unsigned 8-bit integers with zero centered around 128 (0x80), and 0 and 1 are not possible values as they are used as a header value for Y and X respectively.

A Data Acquisition Period

In order to achieve cycle accurate (90.4ns clock period with 50 ppm error) control of the data sampling period, all timing of the system is controlled by an Atmel 8515 RISC processor running custom written Atmel assembly code. The system's target motion data sample period is 500μs and, in order to properly communicate with

the Agilent ADNS series camera chip, 250ns bit widths are required. All other components of the metrology system, including stimulus presentation hardware, are queued prior to initialization of an acquisition session and a hardware trigger signal is generated by the 8515 to sync all activities with crystal accuracy.

The main data acquisition period is controlled by an interrupt which is triggered by an asynchronous 16-bit timer, internal to the microcontroller, driven by the 11.0592 MHz crystal oscillator. A target sample period is indicated by the user and data is acquired with sample frequencies up to 2160 Hz (463 μs). The interrupt subroutine associated with the timer target match must execute all activities associated with a single x/y sample. It must complete:

1. Request dX and dY values separately from the ADNS camera chip
2. Read dX and dY values from the ADNS chip
3. Convert dX and dY into a readable format by the user
4. Transmit data and headers (4 bytes total) over RS-232

The main time bottleneck involves both the RS-232 transmit period (approximately 90□s per 8-bit symbol at 115.2kBaud) and the time for the ADNS to process a sample request and prepare the data for transmit (100μs). Given a 463μs period, serial transmit of four 90μs symbols and two 100μs data requests from the camera chip would take too long (560μs). Fortunately, the 8515 contains an internal Universal Asynchronous Receiver/Transmitter (UART) which runs in parallel with the main program code. The main data acquisition cycle contains an interleaving of motion data requests from the Agilent camera chip and parallel header/value transmission to the user via UART (line levels converted to match RS-232 by an external chip).

Communication with the Agilent ADNS series Cameras

The specifications for the two-wire serial port on all ADNS model camera chips are identical and described in the component datasheet. The serial communications protocol allows for read/write access to an arbitrary internal configuration, status, or data register aboard the Agilent digital signal processing (DSP) chip.

At the hardware level, the interface consists of two general purpose digital I/O lines connecting the Atmel 8515 RISC processor to the Agilent ADNS camera chip and operating at TTL levels. One digital line acts as a clock signal to latch data bits into or out of the chip. This clock line is maintained as a voltage source (output) at the controller such that the camera chip itself always acts as a slave device. The second digital line is a bi-directional data bus. The controller's general purpose I/O port is converted from a high impedance input to a voltage source output depending on the desired direction of information flow. Data bits are clocked into our out of the device on the rising edge of a digital pulse with a 500ns period. In order to interact with the chip, the user sends an 7-bit register address with a read/write flag attached as an extra eighth bit. The state of the controller port is set to either input or output state and then the data/configuration is clocked in/out as an 8 bit number.

The firmware I have written includes subroutines that control this communication protocol with clock cycle accuracy and are called with interrupts disabled to insure appropriate timing.

```
  sbi PORTB,1     ;Trigger high during acquisition

  ;*************************************
  ;REQUEST DX
  ser temp
  out PORTD,temp          ;Set Port pin state for writing
  out DDRD,temp           ;Set SDIO (PORTD4)to Write to the ADNS chip
  mov adns,deltaX         ;Load in the deltaX register address
  andi adns,127           ;set first bit to zero indicating we want to read
  rcall WriteADNS         ;Send request to chip for deltaX
  clr temp
  out PORTD,temp          ;Set SDIO to high z state
  out DDRD,temp           ;Set SDIO (PORTD4)to read from the ADNS chip
  ;*************************************
  ;SEND DY OVER UART
  ldi serial,0
  rcall UARTOut           ;Precede a dy byte with 0x00 header
  mov serial,dy
  rcall UARTOut           ;if sgo is high, send dy byte on serial port
  ;*************************************
  ;READ DX
  rcall ReadADNS          ;Subroutine for reading a byte from the Agilent chip
  mov dx,adns
  ser temp
  out DDRD,temp           ;Set SDIO (PORTD4)to Write to the ADNS chip
  ldi temp, 0x80
  add dx,temp             ;Convert new dx from 2s compliment to unsigned binary
  ;Make sure DX is not 0 or 1 (these are headers for data points)
  cpi dx,3
  brsh pBx
  ldi dx,2
pBx:
  ;*************************************
  ;Request dY
  mov adns,deltaY
  andi adns,0b01111111 ;set first bit to zero indicating we want to read
  rcall WriteADNS
  clr temp
  out PORTD,temp          ;Set SDIO to high z state
  out DDRD,temp           ;Set SDIO (PORTD4)to read from the ADNS chip
  ;*************************************
  ;Send DX over UART
  ldi serial,1
  rcall UARTOut           ;Precede a dx byte with 0x01 header
  mov serial,dx
  rcall UARTOut           ;send dx byte on serial port
  ;*************************************
  ;Read dY
  rcall ReadADNS
  mov dy,adns
  ldi temp, 0x80
  add dy,temp             ;Convert new dy from 2s compliment to unsigned binary
  ;Make sure DY is not 0 or 1
  cpi dy,3
  brsh pBy
  ldi dy,2
pBy:
  ser temp
  out PORTD,temp
  out DDRD,temp   ;Set SDIO (PORTD4)to Write to the ADNS chip
  ;*************************************
  ;Check for a serial byte in during cli period
  ;Check bit 7 of USR.  If set, we missed a byte during getdelta
  in temp,USR
  andi temp,128
  cpi temp,128
  breq ByteIn

  reti
```

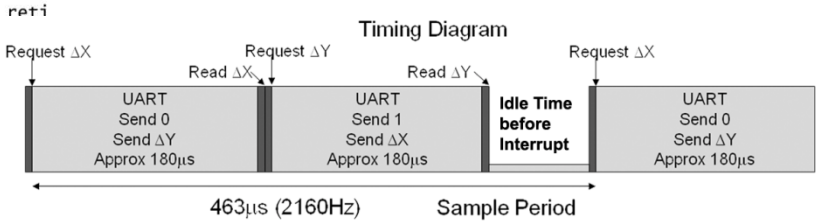

Figure 2.5 – The assembly code for the treadmill data acquisition loop (top) and an illustration (bottom) of the 2160Hz sample period. The data acquisition session is ordered such that a single sample is acquired and transmitted faster than a single sample period.

High Level Protocol Bridge

All communications with the motion tracking chip utilize the hardware level, port toggling, two-wire interface. The firmware contains many features in addition to the raw data acquisition mode. I have constructed a protocol bridge between the RS-232 input and the ADNS two-wire interfaces such that the user may access (read or write) an arbitrary internal register in the Agilent camera chip. This allows the user to access many of the built in calibration and configuration options of the camera device. The user may acquire data such as the surface quality (a measure of the number of features visible by the sensor) or the pixel contrast. The user may also dump an entire 18x18 frame of pixel data from the chip for analysis of focus and other surface information. All of these processes are automated by the Atmel processor and driven by a single byte operation code issued by the user over the RS-232 line.

Hardware Design and Construction

I have designed and fabricated a custom printed circuit board which houses all microelectronic components required for the protocol bridge between a PC based user (connected via RS-232) and a modified optical mouse or other system housing an Agilent ADNS series camera chip. This hardware system mediates all steps of the stimulus presentation and data acquisition process. The printed circuit board contains several key interface elements:

1. 4-pin mini-DIN connector interfaces and powers the ADNS chip
2. BNC connector for a hardware trigger line
3. DB-9 Connector for RS-232 Protocol
4. Power Connection (7-25V DC)
5. On/Off Switch for entire system

In addition to the interface components the board houses a 5V power regulator and line cleaning capacitors (0.1μF), a TTL/RS-232 line level converter (MAX233A, Dallas Semiconductor), the Atmel AT90S8515 RISC controller, and an external oscillator circuit (11.0592 MHz Quartz Crystal Oscillator, and 2x 22pF capacitors). The printed circuit boards were fabricated by www.ExpressPCB.com.

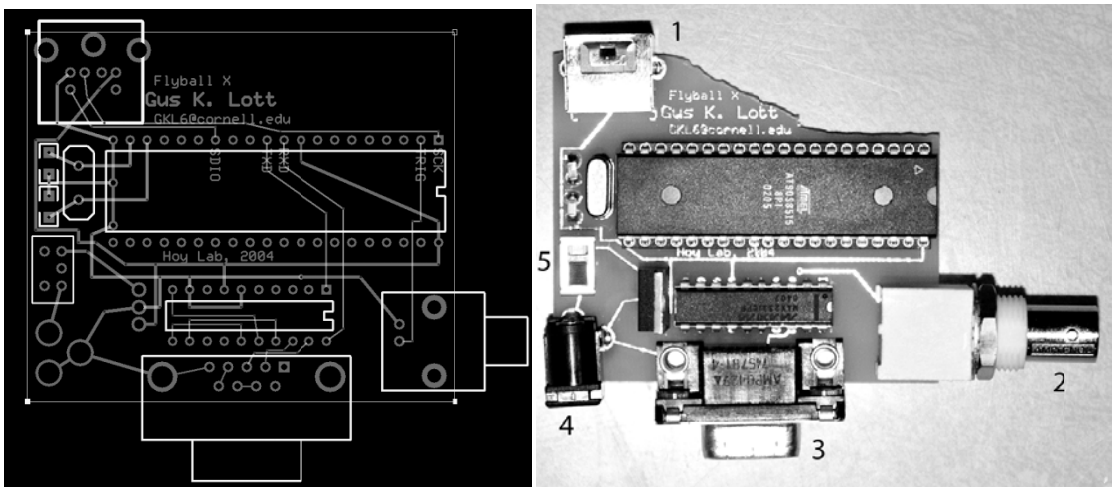

Figure 2.6 – Printed circuit board core of the data acquisition system. Interface components consist of (1) the mini-din 6-pin connector, (2) the data acquisition trigger sync line BNC connector, (3) the DB-9 RS-232 connector for serial data transfer, (4) the power interface for DC 9-25V input, and (5) an on/off switch.

2.3 Preparation of the Fly

Parasitoid flies of the species *Ormia ochracea* (Diptera: Tachinidae, Ormiini) were lab-reared from a natural population collected in Gainesville, FL using sound traps (Walker and Wineriter 1990). Flies were reared on a reversed 12/12 light/dark cycle, and were tested at dusk, as they are crepuscular. Female gravid (larvae-bearing) flies were cooled on ice and fixed to a stiff wire on their dorsal surface with low-melting-point wax. The wire was attached to a micromanipulator, and flies were lowered onto the treadmill in a normal walking position. High-speed videos at 1000

and 2000 fps (Red Lake Camera Systems) were taken to correlate leg movement with trackball movement.

2.4 Signal Structure & System Calibration

Stimuli are designed to mimic the calling song of the field cricket *Gryllus rubens*, the chosen host for this population of *Ormia*. The example simulated calling song consists of trains of 10 sinusoidal pulses at 45 pulses/sec, each trapezoidal-shaped pulse 10 ms in duration with a 1 ms rise-fall ramp. These are generated either at the natural calling song frequency of 5kHz or at an ultrasonic frequency of 24kHz, and are presented at 5dB SPL above threshold. Speakers are positioned at +45° and -45° relative to the fly.

The stimulus field emitted from speakers is calibrated via measurements in the free field using a Bruel & Kjaer (Norcross, GA) type 4138 microphone placed at the position of the fly above the trackball, and connected to a Bruel & Kjaer model 2608 sound level meter. Stimuli are generated and played using custom-written software in Matlab (G. Lott and M.J. Rosen) interfacing with a TDT (Gainesville, FL) System 3 RP2.1 digital to analog converter, fed through a TDT System 3 PA5 programmable attenuator into a stereo amplifier (Harmon-Kardon model HK6100) and out to high-performance tweeters (ESS Systems).

2.5 Software Design & Experiment Control

Coupled to the custom designed complete hardware instrumentation system, I have written, in conjunction with Dr. Merri Rosen, a suite of software tools in the scientific computer environment Matlab. This set of software based interfaces allows me to control every aspect of the data acquisition project and facilitates hours of automated data presentation and acquisition to a tethered fly with little to no human

interaction. These software tools allow for input space generation, stimulus and acquisition sync, and offline data analysis.

Stimulus Generation

I begin interacting with the experiment control tools by parametrically defining a sub-space of a possible input field and a desired parameter resolution. Stimulus files are designed with variable parameters such as:

- Stimulus Duration
- Chirp Center Frequency
- Pulse Rate
- Pulse Duty Cycle

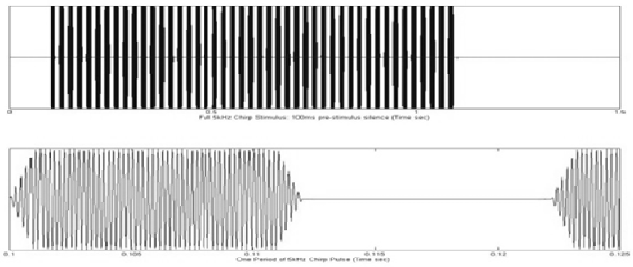

Figure 2.7 – The construction of a single cricket chirp. This pulsed sine wave contains a variety of modified parameters in our experiments. The center frequency, duty cycle, amplitude, and duration of the signal are modified and presented as individual sources or as pairs in choice experiments.

Stimulus Presentation

Once I have a spectrum of input files, they may be loaded for presentation into the main experiment control program in Matlab. Hundreds of experiments may be queued with anything from single stimuli on one of the two speakers to choice experiments with two different signal types from each of the two speakers in the experiment setup. Delays may be set for the initiation of one signal relative to the other and amplitude (dB SPL) may be set for each signal independently. Every

experiment may be entered into an ASCII text based script file and then loaded as a batch experiment into the Matlab program.

Acquisition

Given a series of parametrically defined single stimuli or choice experiments, the software will randomize the stimulus pairs and present them to the fly with random interstimulus intervals. This process prevents learning in the fly and may be deactivated if, in fact, learning and prediction is a parameter I am interested in analyzing. Before presentation of each stimulus, the system will automatically query the treadmill system to see if the fly is moving. The software will continue to poll the treadmill until no motion is detected for a given period of time and will then initiate the stimulus presentation and subsequent data acquisition. This gives me total experiment design control and automates the entire process with little to know bad data sets.

2.6 Initial Data Acquisition & Verification

Walking phonotaxis was measured in female gravid flies via the motion acquisition system, and simultaneously recorded with high-speed video in order to visually corroborate the fine-scale timing of the trackball movement. Four frames of 1000 fps high-speed video (Figure 2.7) show a fly's response at four time points during phonotaxis: sound onset, leg movement onset, ball movement onset, and sound offset. A flashing LED (visible to the upper right of the fly in frame A) coordinated with the voltage creating the sound, allowing visual latency measurements based on frame time. Leg movement onset was visible at 18 msec latency, where the fly began to raise its legs in preparation for walking. Initiation of trackball movement was visible at 57 msec, correlating accurately (at 1 kHz resolution) with the latency indicated by the treadmill system. A second fly was recorded with higher resolution, 2

kHz video (web supplement). This fly's leg movement was visible at 41 msec, and the onset of ball movement was 74 msec as measured both by video and the treadmill. Onset latencies measured by the 463 μsec treadmill system are therefore accurate to within the 500 μsec resolution of the high-speed video. Thus ball movement onset precisely indicates onset of phonotactic walking motion, which lags behind the first leg movements of the fly by 30-40 msec.

I tested how well the system was able to encode small differences in movement latencies. Two stimuli with the amplitude envelope of the cricket host's calling song were presented to the fly, at either calling song or ultrasound frequency (5 kHz or 24 kHz). Each stimulus was presented at 10 dB SPL above the threshold that reliably elicited a phonotactic response 50% of the time. With my system, I am able to measure onset latencies, velocity over time, direction, and movement duration. For example, for the fly depicted in figure 2.9, there were clear differences in movement latencies to calling song versus ultrasound frequencies. The latencies to move toward a 24 kHz chirp stimulus were longer and more variable than those toward a 5 kHz stimulus (54.7 ± 1.6 ms toward 5 kHz, 85.0 ± 16.0 ms toward 24 kHz; figure 2.9, *left panels*). Similarly, the distance traveled was shorter for 24 kHz than for 5 kHz (0.28 ± 0.03 inches for 5 kHz versus 0.13 ± 0.02 inches for 24 kHz), and this fly's accuracy varied for the two stimuli (19.0 ± 3.8° for 5 kHz versus 29.5 ± 7.4° for 24 kHz; figure 2.9, *right panels*). Similarly, the time course of velocity differed for these stimuli (peak velocity = 1.95 ± 0.17 m/s, occurring at 0.31 ± 0.01 sec for 5 kHz versus 1.11 ± 0.17 m/s, occurring at 0.30 ± 0.01 sec for 24 kHz; figure 2.9, *middle panels*). My system can therefore effectively quantify multiple components of phonotaxis that may be sensitive to various stimulus manipulations.

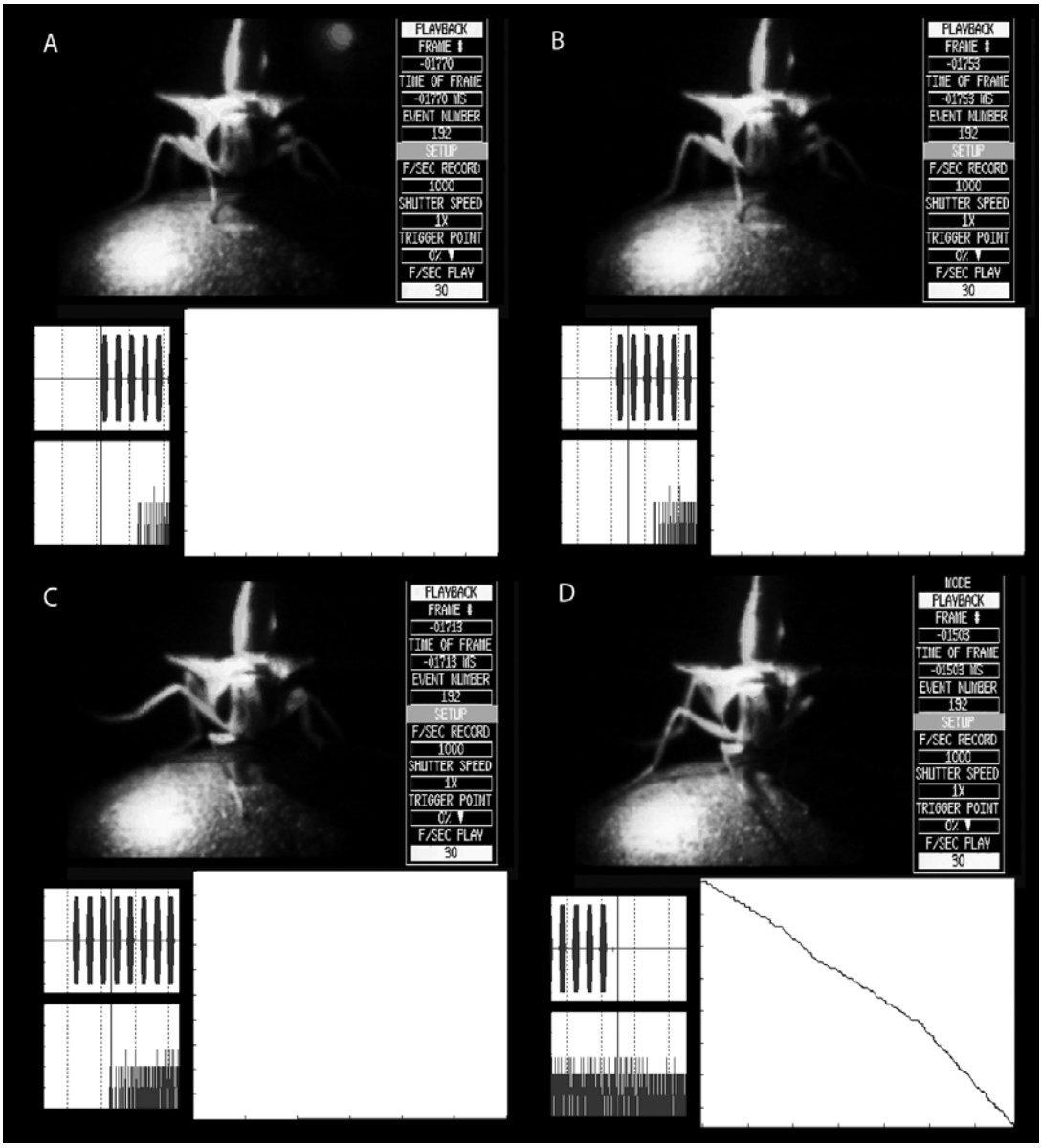

Figure 2.8 – High speed video sequence of a female fly on the treadmill responding to a cricket chirp. Shown in each frame are the actual video (*top*), the synthesized cricket chirp (middle left), the magnitude of the ball velocity (*bottom left*), and the X-Y track of the motion of the fly on the treadmill. The speaker was positioned at 45° to the left of the fly's head. Frames depict movement of the fly at four time points: (A) initiation of chirp (t=0ms), (B) initiation of leg motion (t=17ms), (C) initiation of ball motion (t= 57ms), and (D) end of chirp, including the track of the fly's two-dimensional movement (bottom right) (t=267ms).

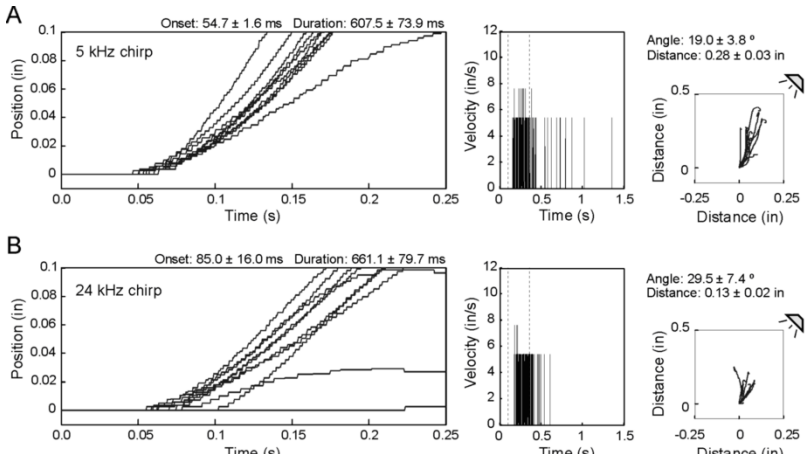

Figure 2.9 – Small temporal differences in various components of phonotaxis are quantifiable by the treadmill system. A and B depict one fly's responses to 5 kHz and 24 kHz cricket chirps, respectively, arriving from a speaker at 45° (see *right panels*). The position and velocity of movement over time are depicted in the *left* and *middle* panels, while the path of the response in x and y coordinates is depicted in the *right* panels. Our manipulation of stimulus frequency affects several parameters of the fly's phonotactic walking response, including onset time (*left panels*), duration, velocity (*middle panels*), distance traveled, and angular accuracy (*right panels*). Stimulus duration is depicted as a *dotted line* in the *middle* panels.

While onsets were extremely accurate, the 2 kHz video revealed an intermittent small jitter in the ongoing movement tracking. This is presumably an artifact of the Agilent ADNS-2610 camera chip. These inexpensive chips will return accurate motion value over time. There just may be some jitter and delay between when motion is detected and actually reported by the chip. I believe that the jitter is due to quantization noise from the relatively high speed data acquisition relative to ball velocity or simply actual jitter in the motion of the ball that we could not otherwise detect. They are, however, extremely accurate devices given their cost and intended application. Another possible source of this jitter could be from small fluctuations in the position of the ball due to the flow of air on which it sits. Small turbulent factors could causes this kind of error to present, but given that the ball tends to stay in place,

this would indicate a zero-mean noise in the data and could contribute to the observed jitter due to the sheer small scale of the pixel resolution that this camera offers. Exact details of this noise source remain unverified.

The main limitations we see in our system involve the fact that the ball only picks up linear motion of the animal while not reporting information about the actual initiation of leg motion. So as the subject moves a leg initially to begin motion, the ball is not rolling to transfer motion to the camera chip. The result is a 30-50ms delay in detection of motion yet it does give us an accurate measure of when actual phonotaxis occurs. We have analyzed the high speed video files and determined that inertia of the ping-pong ball is insignificant in our recordings. The low friction air flotation system incorporated in the device allows the ball to move freely. Motion detection by the camera chip corresponded with the initial movement stroke of the subject's legs. While it may have to exert extra force to get the ball rolling, we can detect that initiation of motion within 1ms (resolution of the high speed video).

2.7 Ongoing Experimentation, Categorical Perception, Sensitivity Mapping

This treadmill system has been successfully applied to the study of *Ormia*'s taxation behavior by a variety of people including undergraduate research assistants and post-doctoral researchers here at Cornell. Input space mapping has repeated much of the initial mapping work illustrated by Oshinky in his cell based work in the fly. Amplitude was modified over hundreds of stimuli. The tuning of the system was verified to have maximum sensitivity in the chirp carrier frequency range from 5 to 10 kHz. This highly sensitive range traverses the same band of carrier frequencies produced by the field cricket.

In addition to the threshold sensitivity mapping, choice experiments were designed to map a variety of parameters that the fly uses to distinguish optimal parasite target amongst a field of candidate crickets.

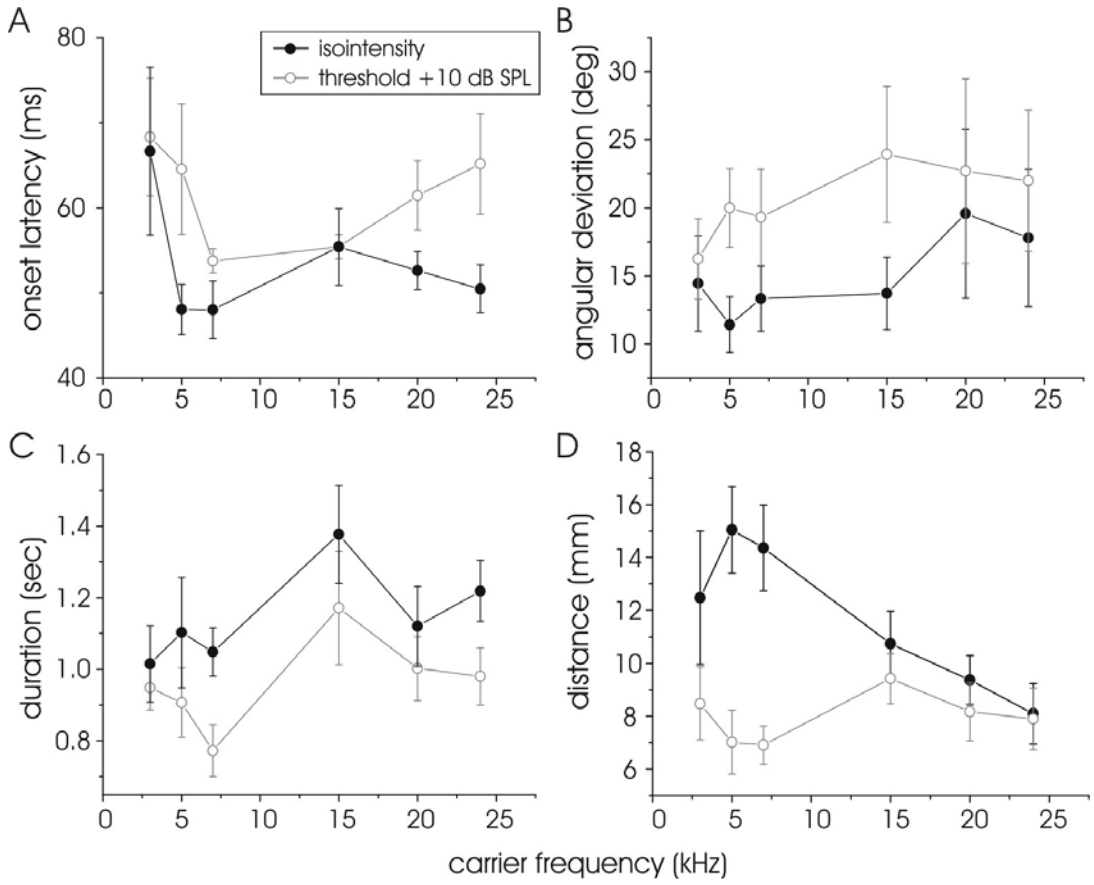

Figure 2.10 – An example of a series of trials of chirps at two distinct amplitude levels and at six distinct carrier frequencies between 3 and 25 Hz. Onset latencies, for example, illustrated rapid response in a frequency range around the carrier for the calling song of the target cricket (5-8 kHz).

2.8 Applications to Other Systems

Finally, work has led into a study of categorical perception between attractive signals such as those presented by the host candidates and repulsive signals generated at ultrasonic frequencies by bats (predators). Categorical perception work has been

carried out in conjunction with free flight recording devices involving a tethered fly and high speed video data.

One of the largest successes of this work is the communication of data acquisition, calibration, and analysis techniques to several gradate students and post doctoral research associates who exemplify the modern biologist up to this point in the advancement of technology. These biologists possess a detailed intuition for biological problem solving and problem expression yet lack the fine tuned experience with instrumentation hardware and software design concepts that have, until recently, been mainly the domain of engineering and physical sciences.

The physiologists involved in this research have begun to incorporate calibration tools into their instrumentation and have developed an enhanced appreciation for the spectrum of tools available to them to increase their data resolution and accuracy. The hardware trigger line built into the treadmill system is an example of one such accuracy facilitating tool. The calibrated and high resolution accuracy of this system has inspired many research institutions to incorporate this device and related instrument control and data acquisition techniques into their physiology labs.

Notably, the treadmill system has been adapted to studies of taxation in *Drosophila* in the Axel lab at Columbia University in New York, and in the cricket Gryllus Rubens (amongst other organisms) in the Mason lab at the University of Toronto.

CHAPTER 3:

POLYIMIDE MICRO-CUFF ELECTRODE ARRAYS FOR RECORDINGS FROM SUB-MILLIMETER NERVE PROCESSES IN SMALL ANIMALS

3.1 Introduction

In the previous chapter, I described high level behavioral analysis of a model neural system. I will now continue with a description of a set of instrumentation tools and analysis methods designed toward monitoring the underlying discrete elements of the nervous system at critical junction points in the organism. In this section, I illustrate the conception, design, construction, and application of a polymer cuff-style microelectrode array. I then describe the application of this device to peripheral nerve bundles in a living system.

This electrode concept is formulated around analysis of the underlying electrochemical activity in the same paradigm as the stimulus-to-behavior correlation described in the last chapter. My electrode arrays are designed to monitor action potentials propagating at multiple points along a nerve. Propagation may be analyzed for insight into direction, character and quality of individual units in the target tissues and for inference of activity both proximal and distal to the implantation site.

3.2 Extracellular Recordings in Multi-Neuron Environments

The nature of neural systems is in no way fundamentally different than any information system analyzed in engineering fields today. The primary subunit of these pinnacles of biological adaptive feedback control systems is the neuron. This single cell in all animal nervous systems has a variety of functions that involve the collection, analysis, and storing of information as well as actuation of motion and other regulatory functions. The main components of the neuron are the dendritic input

fields, the cell body processing center, and the output conduit called the axon. Connections between multiple neurons or between neurons and other tissues may change over time through a variety of methods of synaptic plasticity or during growth and development. These complex networks of neurons make up the fundamental information processing construct within all living animals.

One of the main processes that allows information to propagate amongst neural cells in these networks is a threshold activated cascade of electrolytic material across the cell membrane. As ions rush into or out of the cell to equilibrate the actively generated membrane potential, the resulting change in the nearby solution's electrodynamic equilibrium is referred to as an action potential. It should be noted that not all neurons use these quantized cascade events to carry information, but the vast majority of the signal sources in animal neural systems do. Some cells utilize a graded membrane potential to encode continuous information. An example of this process can be seen in the photosensitive cells in the human eye (photoreceptors).

The field generated by neurons may be picked up by placing a wire nearby the signal source of interest. The motion of the electrolytes into the cell creates a potential gradient throughout the solution and charged particles will flow to equilibrate this potential. By sealing a conductor near the tissue in the vicinity of the potential, and by creating a lower impedance path for equilibration to the conductor than to the nearby solution (referred to as a high sealing impedance), one may force action potential field equilibration to include electrons on the surface of the recording electrode. This capacitive coupling between the conductor and the nearby solution can be monitored through connecting this otherwise insulated electrode wire to an instrumentation system. These small electric currents initiated at the tip of the electrode cause nearly effortless changes in potentials relative to a solution reference point.

Other Neural Signal Recording Methods

Classically, there are two distinct electronic based recording paradigms for extracting neural signals. In addition to the above described placement of a conductor in the nearby solution, a recording element may be inserted directly into the cell of interest. This intracellular approach, where a sharp needle is inserted into the cell itself and signals are monitored from the inside out, offers high resolution for the behavior of a single cell. This technique allows for a display of the full membrane potential yet requires an immobilized preparation and a highly constrained electrode. As one can imagine, when penetrating a cell (diameters of 2-20 microns), there is very little room for actual natural physical behavior of the entire system.

By "electronics based recording," I refer to a capacitive coupling of the electrochemical motion of various ionic species across the cell membrane to electrons in a low impedance wire. It should be noted that there are a variety of methods of extracting membrane polarization information using voltage sensitive photo-active species of molecular dyes. These techniques, however, depend on the opacity of tissue to a variety of wavelengths of light and are limited by the inherent toxicity and persistence of the currently developed fluorescence sources. While not addressed here in detail, it should be noted that this approach to the problem of network behavior analysis is a promising field and may, in the future, offer a wide variety of highly sensitive tools for neural activity monitoring. This may be particularly true when combined with capacitive measurement based tools.

This Extracellular Recording Project

Described herein is an application of the extracellular recording concept to a peripheral nerve bundle in the crayfish, *Procambarus clarkii*. I have selected a bundle of tissue which contains six axons with tonic firing patterns (crowded signal

environment) for demonstration of a novel approach to polymer (in this specific case, polyimide) substrate micro-electrode technology. My electrode array acts as a sub-millimeter scale cuff device designed to constrain tissue in a three-dimensional organism's body cavity. The tissue is constrained such that high sealing impedances may be obtained between the recording sites and the neural tissue relative to the external solution.

This electrode scheme offers a self contained tool for micro-nerve recording. The polyimide array contains five recording sites of 40□m diameter along the length of this constrained nerve process, a reference electrode designed to contact the nearby free solution, and an isolated pair of large diameter electrodes for direct capacitive modification of the nearby solution's electrolytic balance (nerve stimulation). Application of this array to peripheral crayfish motor neurons generates propagation data from several sites along the tonic process. The recordings present remarkable signal to noise ratio for relatively low sealing effort.

The device obtains signal peak to average noise ceiling ratios of 10-18dB depending on the unit of interest in the tissue and maintains this signal quality for several hours across recording sites. These signal-to-noise ratios are achieved with a bare nerve stretched across the array in open saline with no other containment steps. My arrays detect local propagation information such as relative action potential conduction velocities and specific sub-cellular events including sporadic propagation failure in single axons in an intentionally traumatized preparation.

The instrumentation system described in this chapter, like the treadmill system previously described, involves a ground up construction of every component of the experiment apparatus:

- A surgical implantation technique has been designed to allow for implantation of a custom fabricated electrode along a target neural process.

- The electrode's unique geometry is particularly designed for this application to the tonic third root ganglionic offshoot in the crayfish tail.
- The electrode array connects to signal conditioning instrumentation through a custom designed multi-channel interconnect system.
- The multi-channel interconnect interfaces with a custom designed signal conditioning pre-amplifier.
- Amplified signals are digitized using a calibrated multi-channel data acquisition system with control software written in MATLAB.
- All components of the instrumentation system are calibrated using a NIST traceable calibration standard to ensure accuracy.

3.3 Action Potentials: Characteristics, Classification, and Sorting

The elementary component of these analog recordings is the action potential. This bipolar pulse shaped signal represents the transit of a current source (the depolarizing cell membrane) across the face of a recording element (the micro-electrodes). This signal has a characteristic shape that is associated with a variety of parameters relative to both the recording apparatus and the neuron signal source. The two main characteristics of this pulse are the pulse width in time and it's amplitude in volts.

Characteristic Signal Shape

Pulse width corresponds directly to the conduction velocity of the action potential along the neuron and the span of the seal of the neural tissue to the physical recording element. The signal begins increasing as the crest of the depolarization approaches the location of the recording element. The movement of ions creates local potential gradients, and the action potential signal correlates directly to a decrease in

the impedance path of the field equilibration between the axon and the electrode relative to the path between the electrode and the bath solution. As the action potential cascade passes the physical location of highest sealing impedance, the signal reaches a peak. As it moves beyond this position, the current path inverts as the signal begins to move away from (instead of towards) the recording element. The associated voltage peak is then inverted and fades back to equilibrium as the action potential propagates away from the recording element.

Amplitude of the pulse then correlates directly to the maximum ratio of current path impedance between the electrode and the cell membrane and the impedance path through tissue to bath solution. In addition to the sheer path resistance, amplitude of action potentials is often (though not always) correlated to axon diameter. A larger axon would contain more membrane pores through which ions would travel and thus generate a larger ionic current.

The elements of the signals, therefore, have components both based on the underlying physical nature of the neurons in the target tissue and of the relative nature of the current path between that physical neuron and the recording element relative to the external solution. Knowing the general shape of the action potential pulses, I have devised an optimal linear filter to enhance signal to noise ratio. This is effectively an auto-correlation algorithm with a series of archetypal pulse shapes of varying parameters of amplitude and width. By scoring each detected action potential relative to a basis set of relevant basis elements, a distribution of specific classes may be formed.

Design of Signal Detection and Sorting Methods

I have designed such a basis using autocorrelation of pulses found within this specific nerve in the crayfish. Two spikes with similar amplitudes and different pulse

widths are extracted from a data set and used as a two dimensional basis. Each data set is correlated with the given basis vector and the resulting peak score of each detected spike is described in a two dimensional field relative to the scale of it's correlation to each of the two basis spikes. A high score with the short pulse basis vector relative to the longer pulse basis vector would create a dispersion of data related to pulse width. Scoring high or low on both normalized basis elements would indicate a dispersion of amplitude amongst the pulses.

By casting spikes in a basis set spanning both amplitude and pulse width we can develop an optimal means by which to extract individual units from a multi-unit neural recording. In many other methods of basis set definition (such as stochastic proximity embedding or principle components analysis, see appendix 11) the user develops a basis set that contains information about the variable components of the signals relative to one another and incorporates noise elements into the data as well. By taking an initial with a noise free filter with characteristics spanning the known variables of the spike, I have developed an optimal linear filter for both signal to noise ratio enhancement and for separation based on pulse width and amplitude. Such filter applications are common in modern day communications engineering when characteristics of the transmitter are well known. These methods translate seamlessly into the biological arena.

Complex analysis in higher dimensions is required for successfully resolving overlapping spikes in a single channel recording, but once basis vectors have been established for individual spikes in the target tissues of interest, one may express overlaps as superpositions of underlying elements in this multidimensional space. This complex multi-dimensional analysis is beyond the scope of this work.

Given this comprehension of the characteristic signals of interest, their sources, and the factors that increase signal to noise ratio, I begin to demonstrate the

construction of a micro-fabricated tool set and instrumentation arsenal capable of addressing this problem in a multi-unit nerve bundle on a sub-millimeter scale.

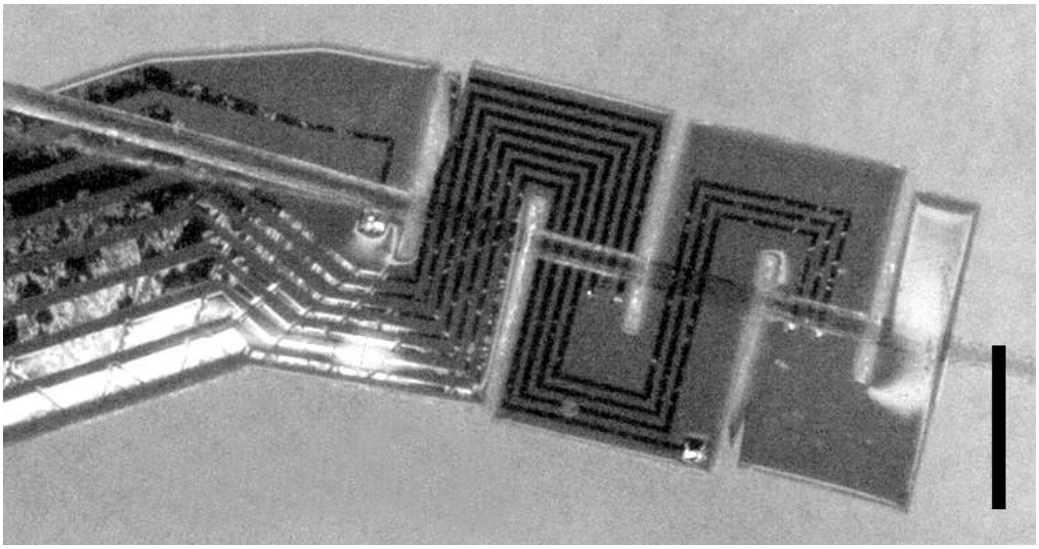

Figure 3.1 – The conceptual micro-cuff implantation design woven around a human hair. Scale bar is 500μm. Structural vias constrain the process to the recording site areas.

3.4 Designing the Polyimide Micro-Electrode Array

Target: Sub-Millimeter Diameter Multi-Unit Nerve Bundles

Multi-unit nerve fibers exist in all animal organisms. It is the obvious solution in nature for information distribution. Accessing these conduits of neural information may yield data about activity both upstream and downstream from a recording site. Placing multiple recording sites along a single process will allow for separation of multiple signals based on signal character and conduction velocity between elements.

For decades, cuff electrodes have been employed to study information in these high value targets (Veraart 1998, Naples 1988, Popovic 2000, Rodriguez 2000, Tarler 2004, Yoo 2005). This geometry allows for multiple electrodes to be placed in contact with tissue, and the structure of the cuff electrode increases the sealing impedance and

decreases the motion of the array relative to the tissue in a behaving organism (thus increasing signal quality). In many recordings from nerve conduits with large number of units (hundreds or thousands), general activity may be detected as combined field potentials.

I have developed a method for the construction of a novel device capable of small footprint implantation upon nerve processes containing few neurons (less than 20). This device is designed for extraction of sensory or muscle control information with the ultimate resolution necessary for the most sensitive feedback neural prosthetic devices (i.e. resolution of single neuron activity).

Array Conceptual Structure and Substrate Material

Cuff electrodes offer ideal geometries for peripheral applications in fields from basic model system electrophysiology to neural prosthetic devices. Nerve processes in a variety of organisms contain dozens, hundreds, or even thousands of neurons. Nerve cords carry information from the periphery sensory neurons to the central processing networks or back from the central nervous system for peripheral control. Tapping into these conduits allows for inference of underlying neural network structure and information transfer. Classically, cuff electrodes span scales of millimeters around large neural processes and gather bulk field potential activity of neurons acting in concert.

We present a novel micro-cuff electrode array capable of connecting to fine neural processes with diameters ranging from 20-200□m. The electrode design utilizes out-of-plane bending and the structural strength of a common biocompatible polymer, polyimide (Haggerty 1989, Richardson 1993, Seo 2004), to constrain target neural tissue for *in vivo* implantation. The electrode arrays are fabricated using established photolithographic micro-fabrication methods (Rousche 2001). The

precision of sub-millimeter fabrication tools allows for feature placement on size scales of individual elements in a neural process. Flexibility of polyimide allows for relative ease of implantation along free tissue processes while the rigidity of the material on small scales allows for process constraint to recording sites.

Fabrication Methods

The micro-machined polyimide process consisted of a photolithographic three exposure procedure. The back plane of the device was patterned in photosensitive polyimide, a lift-off patterning method established a chrome adhesion and gold conduction layer, and an insulating polyimide layer was patterned identically to the back plane allowing for interconnects and electrode localization and exposure.

The electrodes were designed using the computer aided design package L-edit and were transferred to the GCA/MANN 3600F Optical Pattern Generator (PG) using software provided by the Cornell NanoScale Science and Technology Center (CNF). Phtotoresist coated chrome masks were exposed and patterned on the PG and the patterns were exposed in a chrome etchant.

A standard four-inch silicon wafer of arbitrary lattice structure was prepped by Plasma Enhanced Chemical Vapor Deposition (PECVD) of a five micron oxide layer. This sacrificial layer was later dissolved (HF or Buffered Oxide Etch) to release the arrays from the substrate.

Once this sacrificial layer was in place, a thin layer (~10 microns) of photosensitive polyimide (Dow Corning, Photoneece, PWDC-1000) was spun onto the oxide layer using the recommended spin profile (700rpm for 10 sec. followed by 2100 rpm for 30 seconds) with a pre-exposure bake at 115 C for 3 minutes. A back plane polyimide layer was patterned using a 30 second exposure with 365nm light from the EV 620 contact aligner. Exposed polyimide was developed in an alkaline solution (2-

3% TMAH) until the underlying oxide layer was completely exposed. The back layer was baked using the provided eight hour bake profile in a dedicated polyimide bake oven (YES 450).

The conductor layer of the device was patterned using a lift-off process. An initial layer of Shipley 1813 photoresist spun onto the wafer at 4000 rpm for 30 seconds. This layer was exposed using the contact aligner (EV 620, four second exposure) and the resulting image was reversed in an ammonia oven (YES-58SM Image Reversal Oven). Acidic (exposed) sections of the photoresist were rendered inert by the ammonia vapor and the resulting substrate was flood exposed (EV 620) and developed in an alkaline solution (2-3% TMAH). Diffraction at the pattern edges causes the (now inverted) substrate to contain exposed polyimide regions with undercut photoresist sidewalls. This would facilitate the lift-off conductor patterning process.

The photoresist patterned substrates were mounted in the evaporation chamber of the CVC SC4500 E-gun evaporation system. The initial 25nm chrome adhesion layer was evaporated onto the substrate followed immediately by a 125nm gold conductor layer by e-beam evaporation of the individual materials without breaking vacuum. The resulting substrate was sonicated in acetone for approximately 5 minutes. The conductor material deposited on photoresist was released and the cured polyimide base remained with electrode wires patterned.

Following the conductor patterning, a final insulating layer was aligned and patterned using an identical method as described above for the back plane layer. After the final polyimide layer was patterned, developed, verified, and cured, the substrates were submerged in a buffered oxide etch (40% HF) for approximately 30 minutes. The sacrificial oxide layer was dissolved and the polyimide and gold layers were not

attacked. The devices floated freely off of the substrate, were rinsed and transported back to the lab for preparation, characterization, and implantation.

This process would tend to yield 40% arrays with the majority of connections intact. Small particles in the polyimide would cause bubbles in the curing process rendering the arrays unusable in about half of the cases and periodically, the HF etch would attack the chrome adhesion layer and cause conductor failure for one or more electrodes in the array.

Design Evolution

The first electrode configuration (Figure 3.2.1) was designed with a series of trapezoidal flaps. The array was designed for flexing during implantation to allow for the flaps to pin and compress the nerve along the center of the device where the recording sites were aligned. Flexing the device turned out to not be a feasible approach for implantation. With polyimide structures on sub-millimeter size scales (50:1 or smaller aspect ratio) there appears to be considerable resistance to bending. A new conceptual design yielded a device where, instead of bending, individual sections of the array are lifted or twisted. The zigzag design (Figure 3.1 & 3.2.2 & 3.2.3) allows for the nerve to weave within the structure of the array and at the same time require only large scale manipulation of the entire device (i.e. lifting or twisting of segments. This second generation array still presented some difficulties during implantation. Nerve processes would tend to catch on sharp corners when weaving through the device. A successful implantation in this generation of the device did, however, yield action potential recordings in a crayfish tail preparation. A final device design (Figure 3.2.3) solves the majority of the structural problems in the array implantation by increasing via size and rounding sharp corners.

The final electrode geometries took several successive stages of conceptual design, actual prototype fabrication, and test implantations in living systems. The inspiration for the project came from a combination of electrode array projects from both the Craighead and Hoy Research groups. Conrad James' 124 Electrode planar micro-array (James 2004), (My work on this project illustrated in appendix 3) and Andrew Spence's 16 electrode linear array (Spence 2003 & 2007) were applied to two vastly different systems (cultured rat hippocampal neurons and explanted cricket ventral nerve chord). Both of these recording systems were designed for highly specialized recordings. The developed polyimide system has applications in almost all organisms and new electrode geometries can be prototyped within a week to apply the arrays to an arbitrary target either in vivo or in vitro.

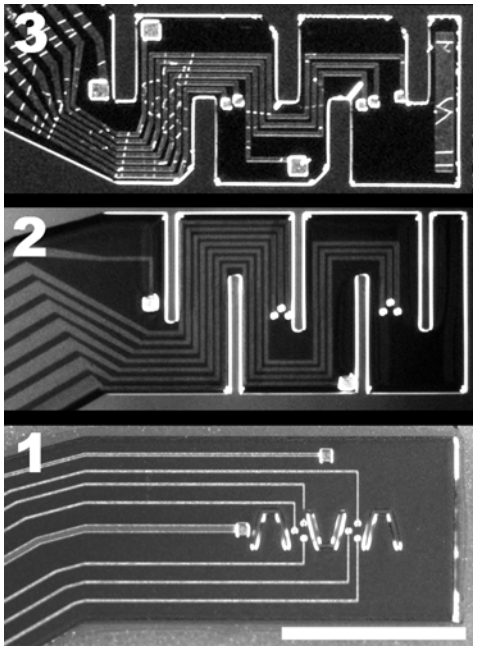

Figure 3.2 – The evolution of the polyimide micro-cuff array. Initial design (1): trapezoidal "flap" features hold tissue in device. Second generation device (2): allows for tissue process interleaving. The current device (3): refined to utilize the weaving structure but forced process implantation into non-overlapping weaving vias with angled corners which would not catch on the tissue. Scale bar is 1mm.

Bridging electrode material from solid silicon based substrates (glass or silicon crystal) to thin and flexible polymers was the first design decision. Polyimide was chosen due to its history and development as both an insulating material in the silicon semiconductor industry and a commonly used material in medical devices such as shunts and other neural prosthetic devices (including the cochlear implant). The final polyimide process used for these electrodes seems ideally suited for the fast prototyping nature of this system. A photosensitive polyimide (Photoneece, Dow Corning PWDC-1000) can be rapidly patterned using a UV exposure and alkaline developer in minutes with features on the scales approaching the individual cells in the systems of interest (5-50 micron).

3.5 Array Characterization and Calibration: Instrumentation and Software

The arrays are visually sorted to a final yield of approximately 40% reliably conducting devices. Failed devices result from bubbling of the polyimide during the curing process or in electrical failure of the conductor layer due to peeling. The relatively thin base of the interconnect portion of the array is fortified by a thin section of printer transparency film (3M Inc) cut to size (Figure 3.4). The device is affixed to the film section with a cyanoacrylate adhesive and silver paint is applied to the interconnect leads for structural reinforcement. Without reinforcement, gold traces will sheer off over repeated manipulations.

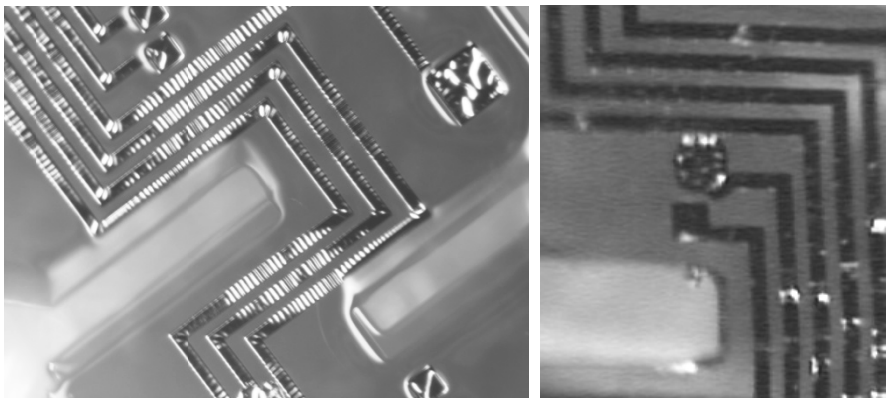

Figure 3.3 – An Illustration of the three layer process of the array fabrication (left frame) where the layers where offset in this failed device on the periphery of the silicon wafer substrate. The platinization process reduces the interfacial impedance of the electrode to the solution from approximately 100kOhms (right frame, bottom electrode) to approximately 5kOhms (right frame, top electrode)

Interconnects & Instrumentation

An 8-pin flat flex cable and board edge connector connects the array to a custom pre-amplifier. Eight, 3-stage, 66 dB gain, amplifiers built around AD623 (Analog Devices) are implemented with a pass band between 1-8 kHz (Figure 3.5). The amplifiers are driven by a DC battery source isolating electrode signals from conventional power supply line noise. The electrode interconnects, crayfish tail prep, and preamplifier circuit are all housed in a faraday cage to further reduce noise. The amplified signals are directed into individual, sequentially sampled (within a single sample period), analog inputs on a multifunction 12-bit, 1.25 MSample/s, data acquisition device (NIDAQ-6070E National Instruments) through a custom connector. This data acquisition board offers a variable gain beyond the preamplifier stage of 0-54 dB. Recordings were carried out using custom written MATLAB (Mathworks Inc.) code.

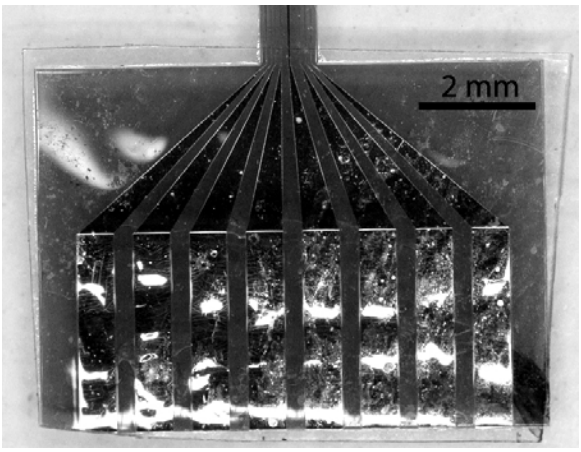

Figure 3.4 – The 1mm pitch electrical interconnect at the opposite end of the device from the electrode tips. A 2cm shaft connects the array process to this interconnect area. Structural support was gained by attaching the relatively flexible polyimide to a semi-rigid plastic backing for insertion into a board edge connector.

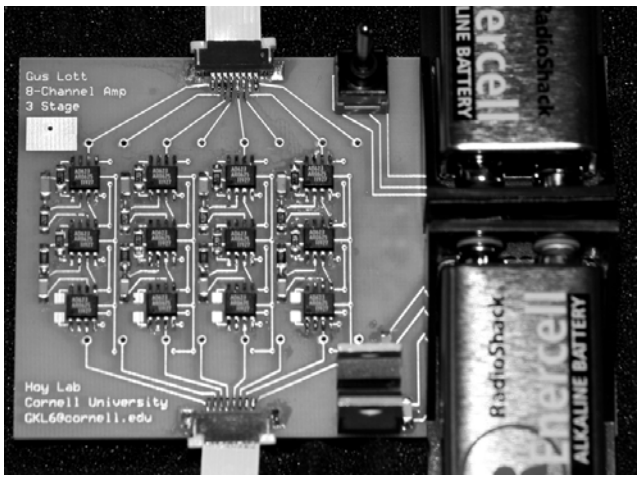

Figure 3.5 – An 8-channel, 3 stage, band pass preamplifier. Shown here are 4 channels (4 identical channels on the back of the printed circuit board) consisting of a unity gain, high impedance input stage followed by an RC high pass filter and amplification stage. The final stage amplifier is an RC anti-aliasing low pass filter and amplification. The total amplifier offers approximately 66dB gain and a pass band between 1 and 8 kHz. The device is driven by a regulated and isolated DC power source to reduce line noise.

3.6 Application: Red Swamp Crayfish (Procambarus Clarkii)

Dissection & Implantation

An adult red swamp crayfish (*Procambarus Clarkii*, approximately 10-14cm in length) is chilled in a freezer for approximately 8 minutes until movement is sluggish or non-existent. The tail is severed from the base of the cephalothorax and the remaining portion of the animal is returned to the freezer. The tail is pinned out in a dissecting dish and submersed in crayfish saline (5.4mM KCl, 205mM NaCl, 13.5mM $CaCl_2 \bullet 2H_2O$, 2.6mM $MgCl_2 \bullet 6H_2O$, 2.3mM $NaHCO_3$, and 2mM dextrose) (van Harreveld 1936). The swimmerets are removed and a T-shaped incision is made in the cuticle of the second or third tail segment. One shallow incision is made immediately caudal and parallel to the sternite and another directly along the midline. Carefully peeling back the cuticle, the superficial flexor muscle innervating, tonic nerve is exposed. The nerve is a thin process extending from the ventral nerve chord ganglion at approximately a 45 degree angle.

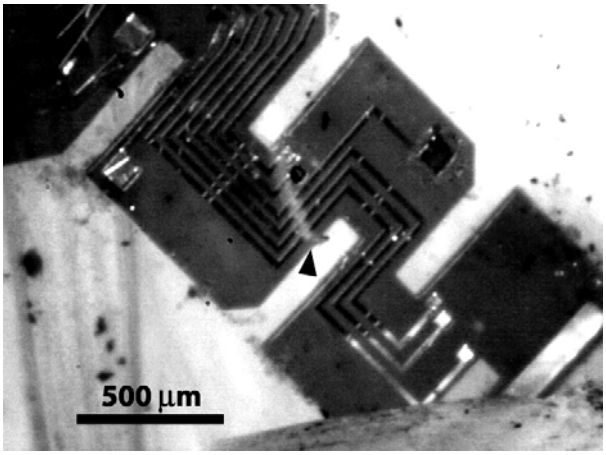

Figure 3.6 – An implanted array in an arbitrary tail segment of the crayfish. The neural system has been stained with methylene blue and the ventral nerve chord traverses the left side of the frame. The thin tonic nerve process (arrow) is shown woven in the device and has been contrast enhanced for identification.

The polyimide electrode array is maneuvered parallel to the nerve. Our installation technique utilizes a pair of hand held probe hooks to push and tug the nerve above or below the device thus weaving it in and out of the structural vias (Figure 3.6). The relative size of the array and the stiffness of the polyimide hold the nerve tightly to the recording sights with little room for lateral or longitudinal movement.

Our initial application of this novel electrode substrate geometry is in the red swamp crayfish, *Procambarus Clarkii*. The crayfish offers several easily accessible peripheral processes which express tonic activity and contain local feedback loops to modify neural activity. The thin, tonic, third root offshoot of an arbitrary tail segment ventral ganglion contains five excitatory neurons and one inhibitory neuron. These neurons activate a series of postural control muscles through a dissinhibitory innervation scheme (Wine 1974). The nerve is readily accessible as a free process directly under the ventral cuticle over a distance of several millimeters and has a diameter of approximately 50-75 m. This thin nerve's spectrum of neuron sizes and functions along with its signal rich firing scheme and proximity to the ventral surface make it an ideal test bed for new micro-electrophysiology devices.

Observed Waveforms

Signals acquired in this crawdad tail preparation are discrete action potentials originating from any of the 6 neurons in the tonic offshoot of the third ganglionic nerve root. Waveform period varies as a function of both the recording site width and action potential conduction velocity. Spikes were observed with biphasic shape and characteristic pulse widths between 2 and 4ms and the largest and fastest spike peak amplitude is approximately 8-10 times that of the smallest and slowest spike class in the recordings.

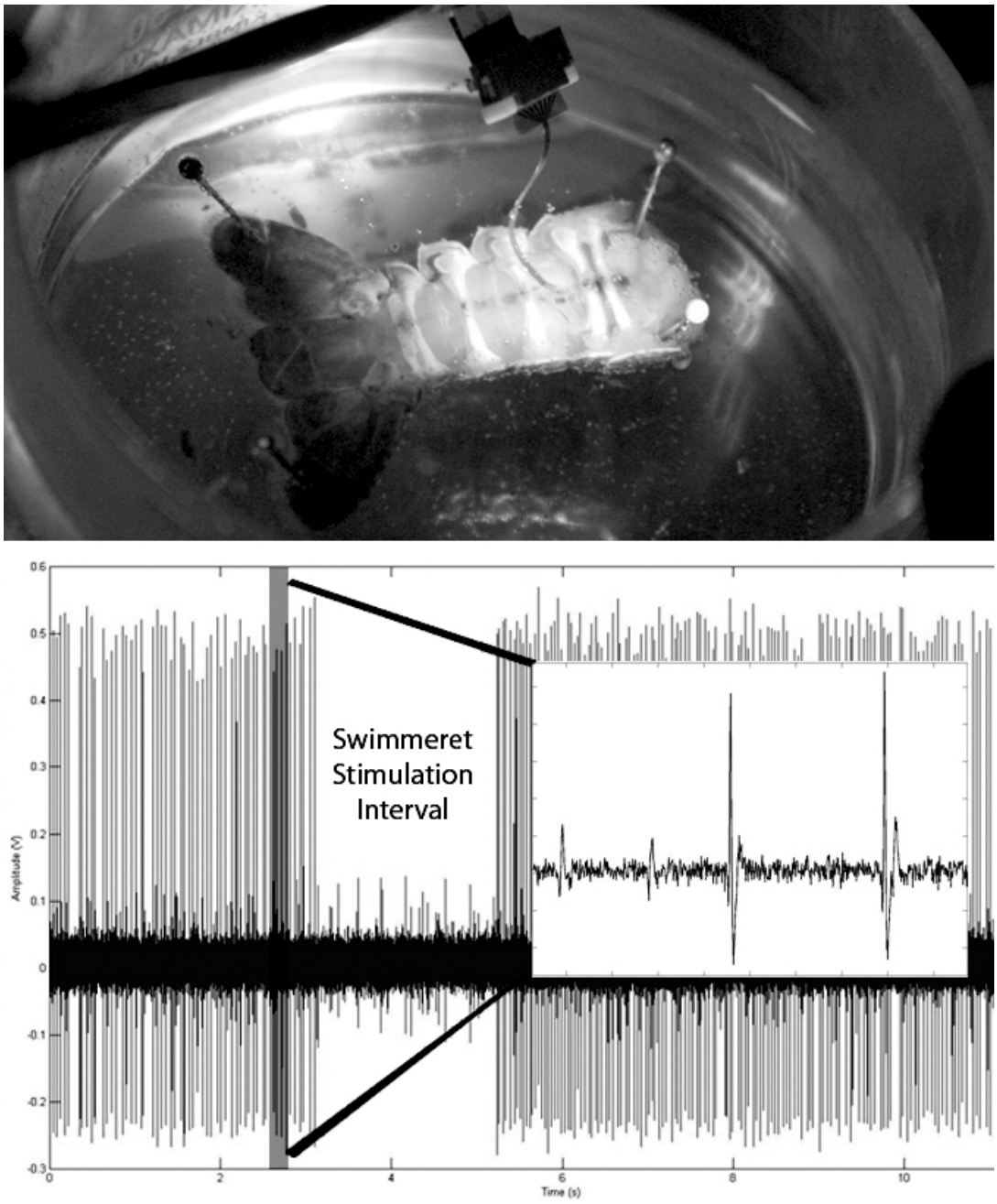

Figure 3.7 – Successful implantation in a severed crayfish tail and corresponding recordings from a single channel demonstrating the quality and character of recordings from this system with relatively little sealing effort. The suppression period of the large action potential corresponds to mechanical stimulation of the swimmeret stump. Horizontal ticks represent 2 second intervals.

Multi-Channel Recordings of Action Potentials in a Tonic Nerve

Existing cuff electrode arrays are manually curled into cuff tubes, tightly wrapping target tissues, with inner diameters on the order of millimeters in order to isolate recording sites from external signals. Our cuff design harnesses the rigidity of polyimide on sub-millimeter scales to constrain a neural process tightly against recording sites for optimal signal acquisition. Planar fabrication methods allow for precise placement of recording sites and tissue constraint features.

A single process is woven within the structure of the device and reliable recordings are acquired over a period of several hours in a severed tail preparation (Figure 3.7). Spike propagation is observed along the length of the nerve (Figure 3.8). Calculated velocities for action potentials in the tonic nerve are on the order of 1-2m/s with error derived from an inability to resolve the actual electrical center of the recording sites (site size and separation distance are of similar scales). Multiple action potential classes are recorded from distinct neurons (Figure 3.7 Large and Small Amplitude Spikes). Mechanical stimulation of a swimmeret stub suppresses spiking activity in the large inhibitory neuron in order to verify that the signals we monitor are originating from the tissue of interest and are not artifacts from the instrumentation.

A nerve was pinched at a location between recording groups on the electrode array in order to induce some form of trauma. We have observed intermittent propagation failure (Figure 3.8) of action potentials between recording groups. Action potential propagation failure and success was observed on time scales on the order of 50ms for a unique neural unit in a relatively motionless preparation leading to the conclusion that this was a neural phenomena and not an artifact of the recording apparatus. It is known that the nerve tissue contains only supporting cells and axons. These axons have proximal cell bodies located in the ventral nerve chord ganglion and distal neuromuscular synapses to superficial flexor muscles (Wine 1974). Thus we

infer that this periodic propagation failure is due to mechanics within the sub-cellular structure of the axon itself relating to the induced trauma.

The main time limitation on device fabrication resides in the geometry conception, computer aided design, and photolithographic mask fabrication. Once these initial steps for a given geometry are completed, the main time constraint is the polyimide cure process.

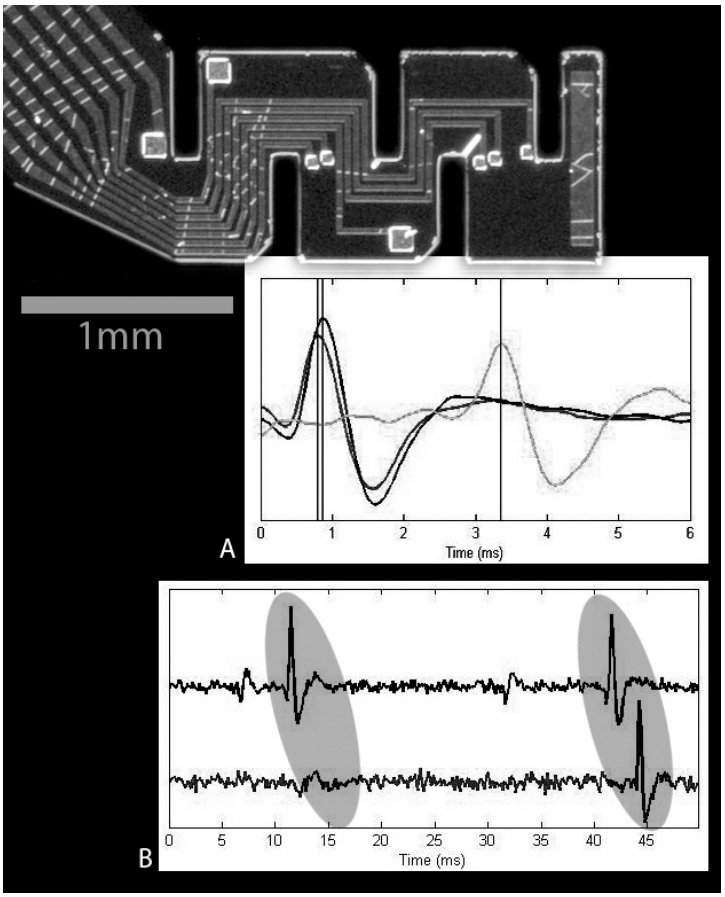

Figure 3.8 – Array recordings from multiple channels along a single process. Propagation of a single action potential is shown (A) with AP peaks indicated by vertical lines. The initial channels represent the relatively short distance between upstream recording sites and the third spike indicates the longer distance between electrode groups. In an intentionally traumatized preparation, we observed periodic AP propagation failure between proximal and distal electrode groups (B).

CHAPTER 4:
gPRIME – DATA ACQUISITION, STIMULUS & ANALYSIS SOFTWARE FOR NEUROPHYSIOLOGISTS IN RESEARCH AND EDUCATION

4.1 Introduction

In this chapter, I describe the construction and application of a suite of software based tools for physiological data acquisition and analysis. This will demonstrate the integrative power of my entire project by illustrating how the micro-electronics and micro-device structures are connected to the human being thus completing the construction of total engineering analysis toolset. This toolset connects the physiology of a target system through a calibrated metrology system to a human being who is asking a question.

Applications are illustrated including analysis of recordings from the crayfish peripheral motor neurons, an analysis of mating behaviors in the fruit fly, behaviors of mildly electric fish, and in a teaching environment connecting undergraduate students to physiology in a manner which frees them from understanding the interface in order to focus on the physiology of interest. The software system, designed in MATLAB, is expanded to generic applications in neurophysiology and medicine and is launched to other institutions for educational purposes.

This software system is a stand alone program generated in Matlab (Mathworks Inc.). The program is designed to interface the researcher to their installed data acquisition hardware whether that's the generic windows sound card input, or a multi-channel calibrated National Instruments data acquisition system. The software acts as a virtual oscilloscope program offering the researcher full control over the behavior of the data acquisition hardware. The software also allows for

rudimentary stimulus generation in addition to the input features. Data may be streamed to disk in a format native to their hardware interface while storing every detail of the internal computer setup during acquisition.

In addition to the data visualization, stimulation, and storage and retrieval functionality, the software boasts event detection in real time with several display methods. Amplitude threshold crosses may be detected and a windowed period of time may be analyzed based on maximum amplitude, minimum amplitude, and main frequency component of the signal. This delivers a variety of information that can characterize spike style signals common to neural systems (such as action potentials or electric organ discharges from fish).

Figure 4.1 – Cornell undergraduate students actively stimulating a muscle and monitoring synaptic plasticity using my program in Cornell's Neurophysiology Laboratory Course, BioNB491.

4.2 Previous System in BioNB491, StimScope & FreqHisto

Since 2001, students in the neurophysiology lab course here at Cornell (BioNB 491) have used a set of software tools in MATLAB that allowed the students to acquire data and carry out limited threshold sorting and histogram generation. While

this software enabled students to carry out all of the assignments, it had much room for improvement.

When I took the course in 2003, I was introduced to the capacity of the program and the amazing nature of the course. I saw a prime example of a large amount of energy spent to support neurobiology education wherein, a small modification to some components of the instrumentation could release the students to fully explore the preparations. I wrote a version of the software that I would have used in the course during the recording phase of my polyimide microelectrode array project. I used this software to record from the systems I worked on and kept the interface as generalized as possible. This program provided the basis for the software I would later write and implement in the course.

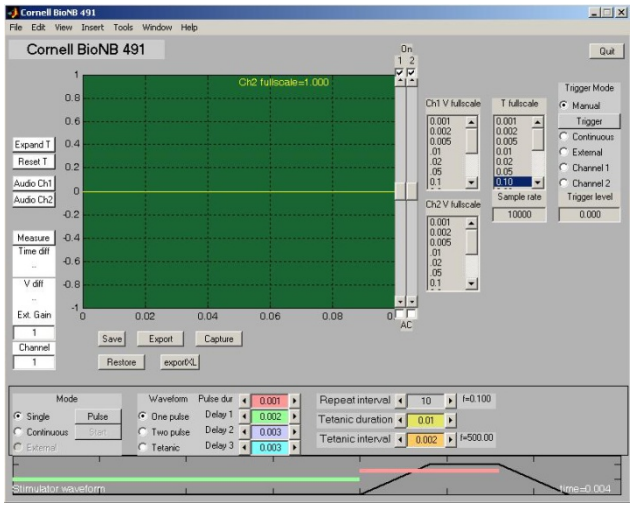

Figure 4.2 – StimScope by Dr. Bruce Land: the gPRIME predecessor.

4.3 Project Goals

In the spring of 2007, I was given the opportunity to assist in the teaching of the course. I began my time as teaching assistant by rewriting the software

oscilloscope functions into a separate program that could take full advantage of the latest features of the MATLAB data acquisition toolbox. In particular, the ability to visualize data as it was being acquired (previously unavailable, data would be displayed after acquisition was finished) and the ability to stream data to disk for storage.

After this interface was designed, several professors heard about the re-write and began to take an interest in the project. After consulting and presenting the space of possibilities of the MATLAB toolboxes and the data acquisition interfaces available, I designed the following goals for a total data acquisition and analysis opus.

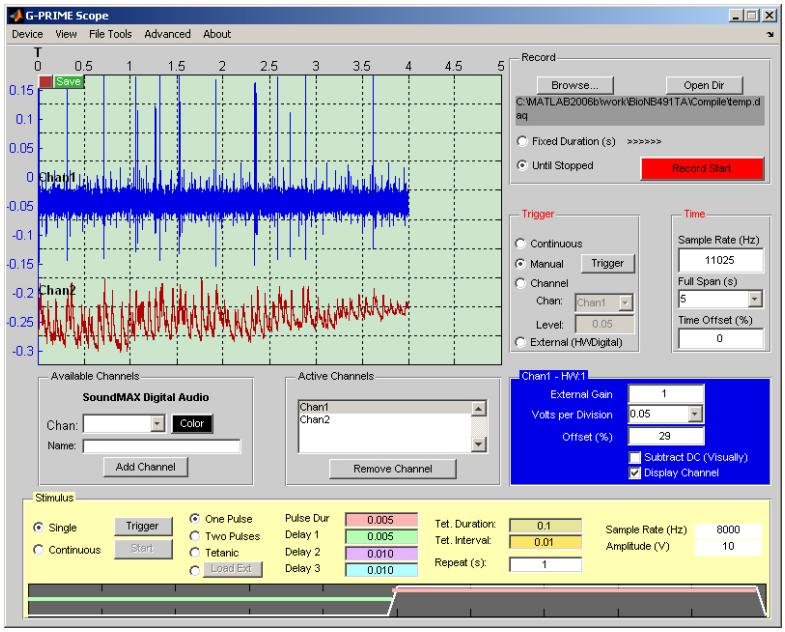

Figure 4.3 – The main software oscilloscope interface. The bottom panel controls stimulus delivery. Two channels of data from electrically active cells are shown.

Data Visualization, Acquisition, and Stimulation

1. Real-time multi-channel data visualization updating several times per second
2. Stream data to disk for offline analysis

3. Real-time spectral analysis for visual signal analysis
4. Delivered controlled stimulus pulses of a given voltage and pulse duration
5. Compiled as a stand-alone program that runs independent of MATLAB

Analysis

1. Both real-time and offline data analysis
2. Signal conditioning with band pass filters and rectifiers
3. Double threshold event peak detection and rejection
4. Multiple analysis metrics for each event visualized in a 2D space
5. Event triggered correlation with a target
6. Report generation capable of creating publication quality graphics

I have achieved these goals and implemented them in the classroom to students of the neurophysiology lab course, biomedical and mechanical engineering students from other courses, and as part of the freshman explorations experience at Cornell. In addition to these educational applications, the software has been applied to research projects in a variety of biological systems in the Neurobiology and Behavior department including drosophila mating behavior (C. Dustin Rubinstein of the Hoy Group) and electric fish spike activity analysis (Carl D. Hopkins). These researchers are acting as beta testers for the software which is now a permanent fixture in the BioNB 491 lab course. The software is slated for expansion into a variety of laboratory environments at Cornell and at institutions across the country. The program is also aimed at high school and community college applications bringing physiology into the grasp of students at all levels.

The software is tentatively named gPRIME (G') which stands for "Physiology Recordings & Identification of Multiple Events." It is written entirely in the

MATLAB scientific programming language. The critical elements of the code itself are illustrated in appendix 2.

4.4 Data Visualization, Stimulation, Storage, & Retrieval

The data acquisition portion of the program is designed to perform all of the functions of a stand-alone oscilloscope while providing storage functionality otherwise not available. The interface contains a variety of sweep trigger modes including continuous, manual, and channel voltage level. Interface sample rate may be set and gain ranges for each channel are controlled when available. The user may smoothly drag the vertical position of traces around the scope window and control voltage scales for individual channels in the scope window. The user may visually subtract the DC component of a signal for display in order to allow for easy visualization of events happening in situations where a membrane potential is involved.

4.5 Event Capture & Analysis

As an advanced option, the user may enter into a data analysis for real time or offline manipulation of data sets. The interface supports a variety of file types for offline analysis including the Axon Instruments "Axon Binary File (ABF)" file type which is a popular program for intracellular physiologists. The signal may be conditioned by a pair of third order Butterworth FIR filters implemented at a variety of selectable frequencies in a logarithmic span of frequencies into ultrasonic ranges. Once the signals are conditioned, the user sets a threshold level (either visually or numerically) along with a window width. When threshold crossing events are detected, an event is extracted based on the window width and a selectable "center on threshold" parameter.

Once extracted, the system calculates 7 parameters associated with each event. This basis transform compresses the hundreds or thousands of data points in an event into a relevant analysis basis for common event classes found in a variety of lab preparations.

1. **Event Time (sec)**
    - Relative to start of the last input reset or beginning of the file
2. **Rate Between Events (Hz)**
    - Inverse of difference between event times
3. **Interval Between Events (sec)**
    - Difference between event times
4. **Minimum Amplitude in Window (V)**
    - This value is used to reject events outside of the two-threshold range if thresholds are set below the mean value of the signal
5. **Maximum Amplitude in Window (V)**
6. **Peak Frequency Component (Hz)**
    - The mean value of the window is subtracted from the signal and the Fast Fourier Transform (FFT) is calculated. The peak value of the FFT is extracted. This value is proportional to the derivative of the signal or to the pulse width of a bipolar signal.
7. **Energy Density**
    - Energy density is the sum of the FFT divided by the number of points in the FFT. It is a fairly clean relative measure of signal energy as noise sources will tend to have constant energy density. The energy density of signals relative to one another then depends only on the contribution from the signal source. This is the best "amplitude" measurement in most cases.

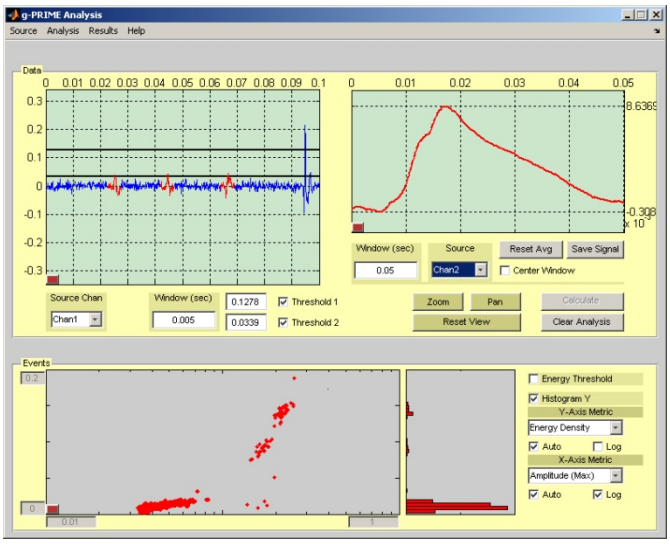

Figure 4.4 – The analysis mode fully engaged. Events are detected with peaks in an amplitude range (top left) and are cross correlated with a second channel of data (top right). The events are cast into a two dimensional space in real time for clustering (bottom left) and a histogram of the vertical axis data may be displayed (bottom right).

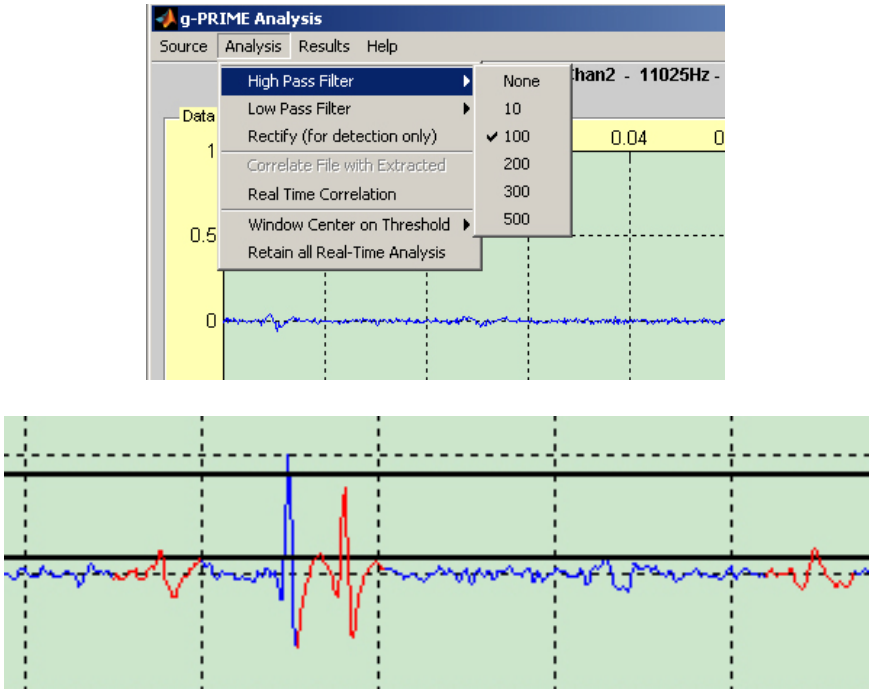

Figure 4.5 – Double thresholds may be used to acquire events that peak within a range of amplitudes (bottom). A variety of analysis options are available to the user for real time and offline data (top).

4.6 Event Correlation and Report Generation

Once analysis metrics are calculated, the user may select out a subgroup of the signals for real time or offline correlation with another signal source. This tool allows the student or researcher to orient their experiment to a site of interest with active activity. Signal correlation between channels of acquired data is a powerful tool for the physiologist.

Once a subset of events has been selected from a given data set, the user may groom the data set to extract outliers from clusters and otherwise poorly grouped signals. These analysis result subsets and raw traces may be saved for further analysis outside of this program. The values may also be loaded into the program for correlation with an associated data set.

At all times, the user may extract generated graphics or generate histograms of analysis parameters into publication quality figures.

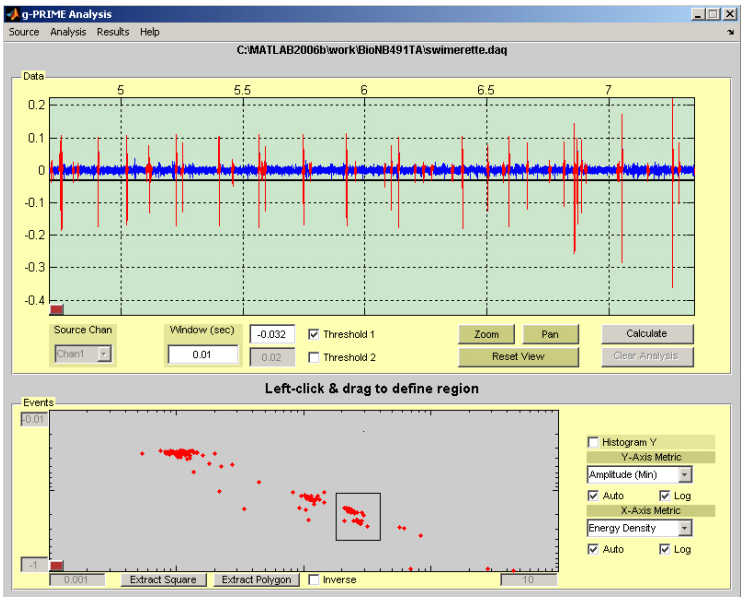

Figure 4.6 – Clusters of individual spikes may be extracted from offline data for report generation or cross correlation.

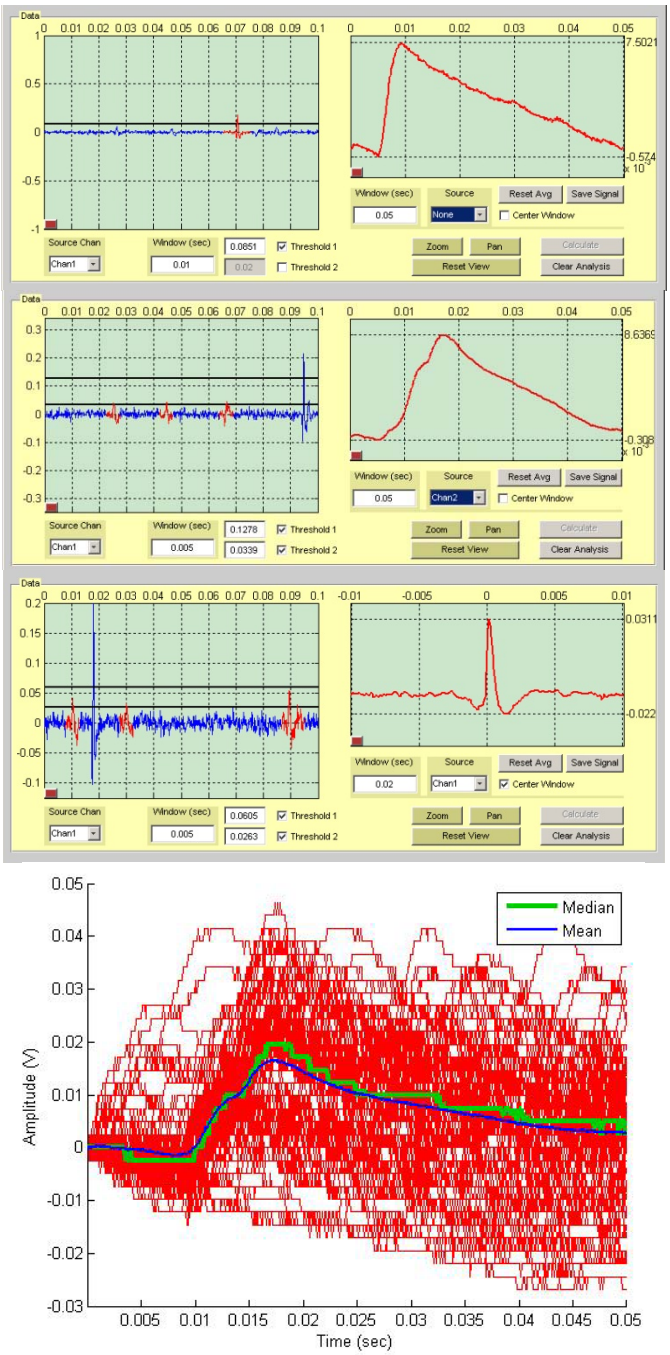

Figure 4.7 – Two distinct post synaptic potentials (PSPs) in a muscle correlated with two action potential classes in an extracellular recording from a multi-unit nerve bundle innervating the muscle (top 2 frames). An autocorrelation of a low signal/noise signal to generate a matched filter (third from top). Offline correlation allows the user to detect the reliability of their correlation results.

4.7 Applications

The program is currently disseminating into the general use of neurophysiologists at Cornell. This program fills a vital hole in the toolset of the modern physiologist. It will enable many studies of cellular and behavioral physiology. The following images represent example applications of the software to a study of systems including, the Crayfish (*Procambarus Clarkii*), the pond snail (*Lymnaea*), and *Drosophila*.

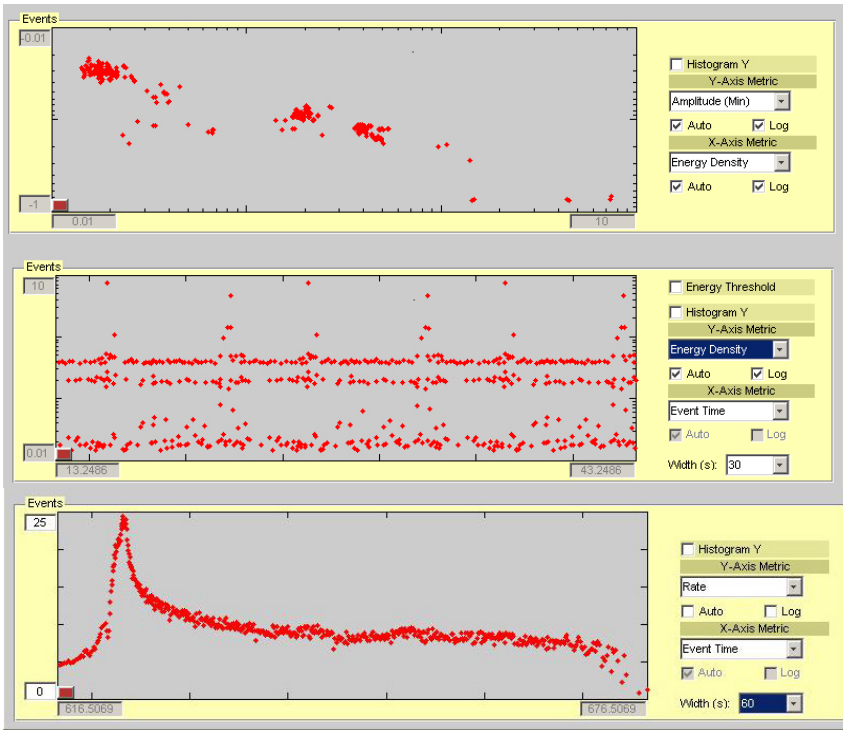

Figure 4.8 – Examples of data acquisition sessions. Parametric display in a two dimensional space of analysis parameters reveals clusters of certain types of action potentials (top). The same data is displayed as it is acquired in real time illustrating the distinct peak amplitude level differences between the events (middle). A rate display also demonstrates a sensory adaptation to a stimulus (bottom).

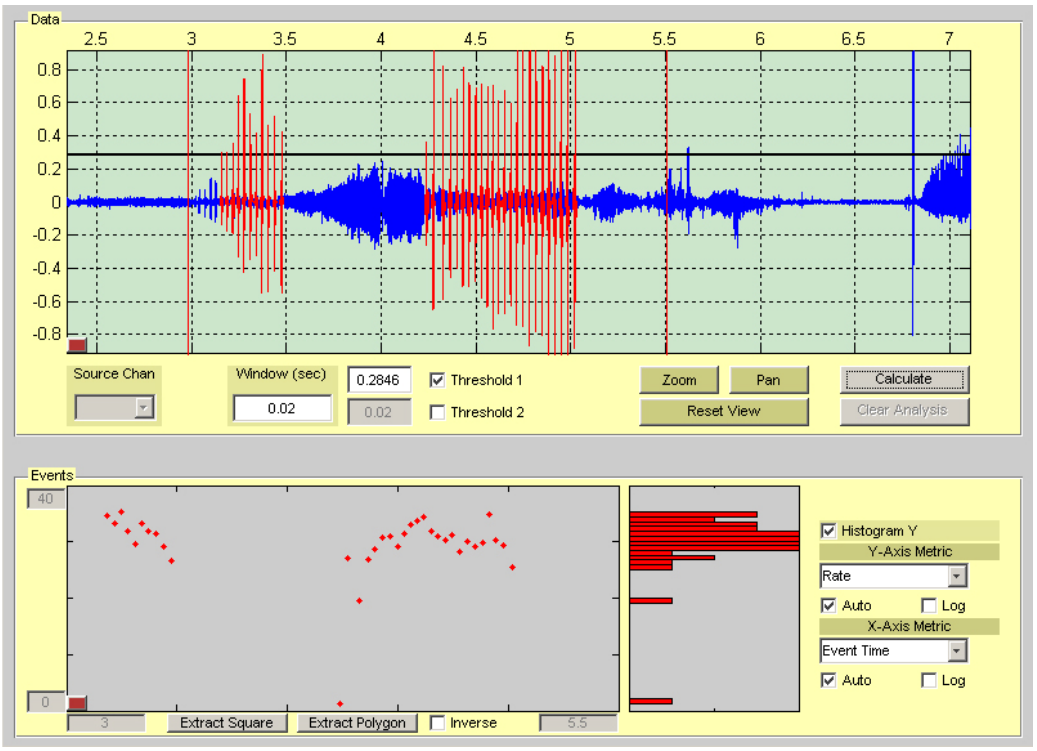

Figure 4.9 – Applications of this software to recordings of the mating behavior of certain *Drosophila*. The detected rasp frequency may be modified by genetic factors.

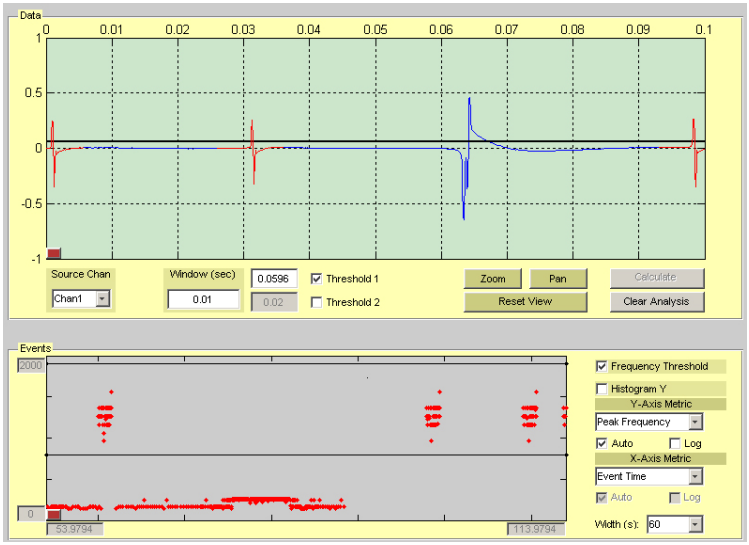

Figure 4.10 – Frequency thresholds applied to electric fish recordings. The low frequency male signals may be rejected in real time even though they cross the amplitude threshold for detection. High frequency female signals are detected.

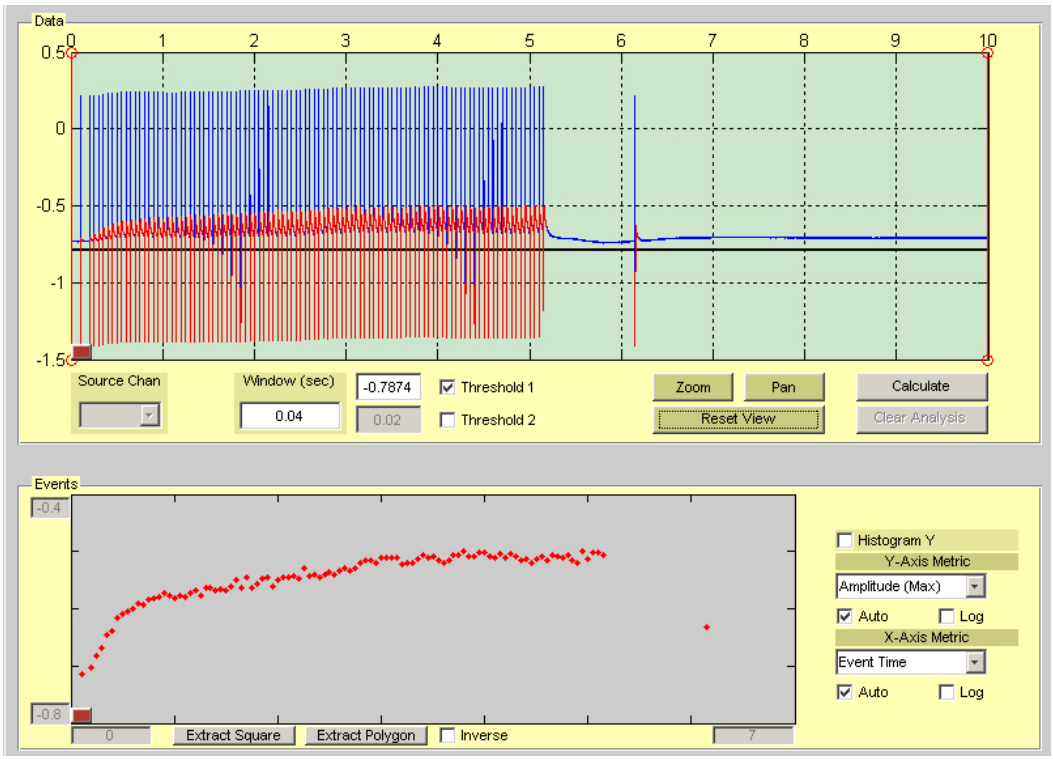

Figure 4.11 – Synaptic plasticity is actively detected as the system reports the max amplitude of post synaptic potentials elicited by a tetanic stimulus.

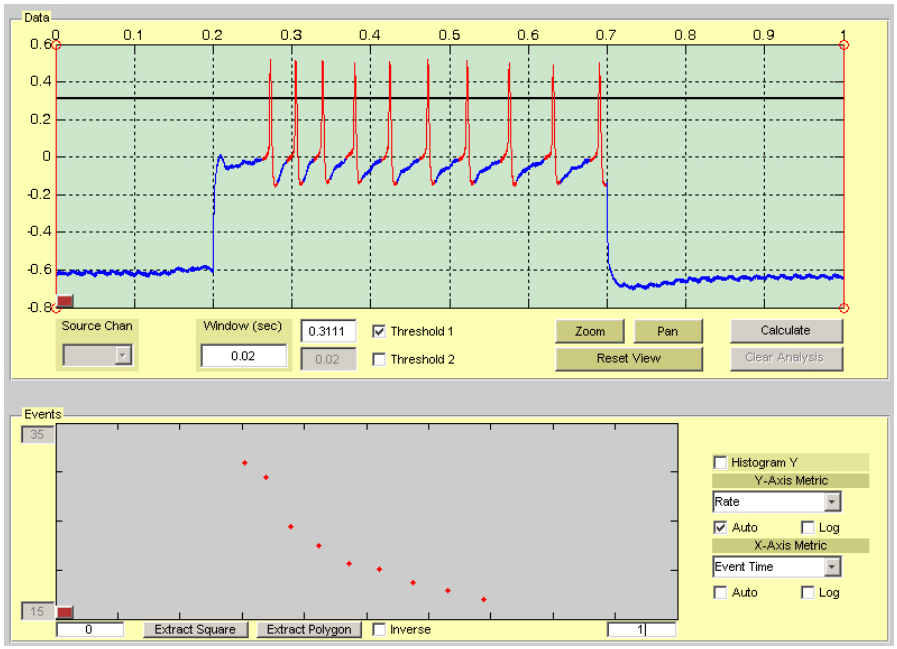

Figure 4.12 – Analyzing the rate behavior of action potential bursts in the snail brain (in this case, Post-Inhibitory Rebound).

4.8 Discussion & Future Direction

This project has already taken a life of its own and is expanding into applications in a variety of research labs. I hope to see this program released into the public domain for education and university research applications. In the spirit of my dissertation, this project expresses the hybridization of physiologists with modern engineering tools. It is the most actively incorporated product of my work at Cornell immediately into the neurobiology community.

This software system enables the user to fully utilize their data acquisition system and allow them to visualize their results in many highly relevant ways. It frees the biologist's intuition to focus on the systems on interest while not being limited by their toolset. At the same time, the software doesn't act as a black box whose inner workings are a mystery to the user. It offers powerful tools that the user must understand in order to implement and presents them in a way that the physiologist can intuitively learn and incorporate these expressions of engineering and biophysics into their analysis of their own project's data.

CHAPTER 5:

APPLICATION OF G-PRIME TO AN ANALYSIS OF THE STIMULUS RESPONSE OF 6 CRAYFISH MOTORNEURONS AND 2 CRAYFISH SENSORY NEURONS

5.1. Introduction

The Red Swamp Crayfish (*Procambarus Clarkii*) is an ideal teaching environment for neurophysiology. The peripheral neural system is well studied (Kennedy & Takeda 1965, Wine et al 1974, Velez & Wyman 1978, Pearce & Govind 2002, 2003) and offers many systems analogous to those found in mammalian nervous systems. Of particular interest is a small tonic nerve containing six neurons that innervate a thin, ventral sheath of muscle tissue tasked with active postural control. This nerve offers an easily accessible nerve bundle with a variety of sizes, shapes, and functions of neurons. The small number of neurons allows for student discrimination of individual units from extracellular recordings.

The nerve innervates an easily accessible muscle group with little isolation from the nearby solution's electrolytic contents. This nerve/muscle system allows for the illustration of a wide variety of properties common to all neural systems. The students may study neuromuscular innervation, short term synaptic plasticity, and the ionic basis of the resting potential in the superficial muscle tissue cells. Of particular interest is the multi-terminal innervation of the muscles by multiple neurons. In mammalian systems, muscle tissue is generally innervated by a single motorneuron. This multi-terminal innervation scheme offers a system analogous to that found in mammalian brains where higher order processing cells receive inputs from a variety of cells with dynamic synaptic strengths.

On the extreme dorsal side of the animal in the tail is the superficial extensor muscle. This muscle is responsible for extending the tail to a straight position after the deep flexor has flexed the tail as part of the animal's escape response. There are two specific neurons on each side of each segment of the tail tasked with encoding the degree and rate of stretch of the tail. These muscle receptor organs (MROs) have mechanically gated channels and synapse onto the superficial extensor muscle such that membrane flexing of the muscle correlates to channel opening in the synapsed receptor membrane (McCarthy & McMillan 1995). The two sensory neurons respond to different rates of stretch and have distinctly different adaption rates to stimulus. In a teaching environment this illustrates a somatic sensory system (proprioception) and the concepts of stimulus adaption and multi-neuronal encoding for increased sensory dynamic range. Adaptation in these neurons is analogous to adaptation in sensory systems found in virtually all mammalian systems. Multi-neuronal encoding of a broad dynamic range of signals (through multiple MROs) is analogous to systems such as hearing (multiple neurons for multiple sound frequencies) and temperature sensation (Capsaicin/Menthol receptor neurons for a higher temperature dynamic range).

In this chapter, I apply g-PRIME to these motor and sensory neuron bundles for a spectrum of neural systems analysis. I compare previously utilized software tools from the Cornell neurophysiology course (BioNB491) to the output of g-PRIME while characterizing the reflex response of the poly-neuronal motor neuron bundle and the sensory adaptation characteristics of the multi-unit proprioception organs. I will illustrate the power and accessibility of the crayfish preparation and how my software may be applied to acquire both real-time and offline details of the specific electrophysiology of these model systems.

5.2. Six Crayfish Motorneurons, Reflex Activity, and Innervation (Neuroanatomy)

The crayfish tail is a relatively simple structure of repeated segments with 3 distinct layers of muscles. From the ventral to dorsal surface, the crayfish has a thin sheet of several superficial flexor muscles responsible for control of posture, a thick "deep flexor" muscle responsible for rapid swimming and escape response behavior, and a dorsal superficial extensor muscle which provides feedback about tail position while activating to extend the tail back to a resting position after an escape response pulse.

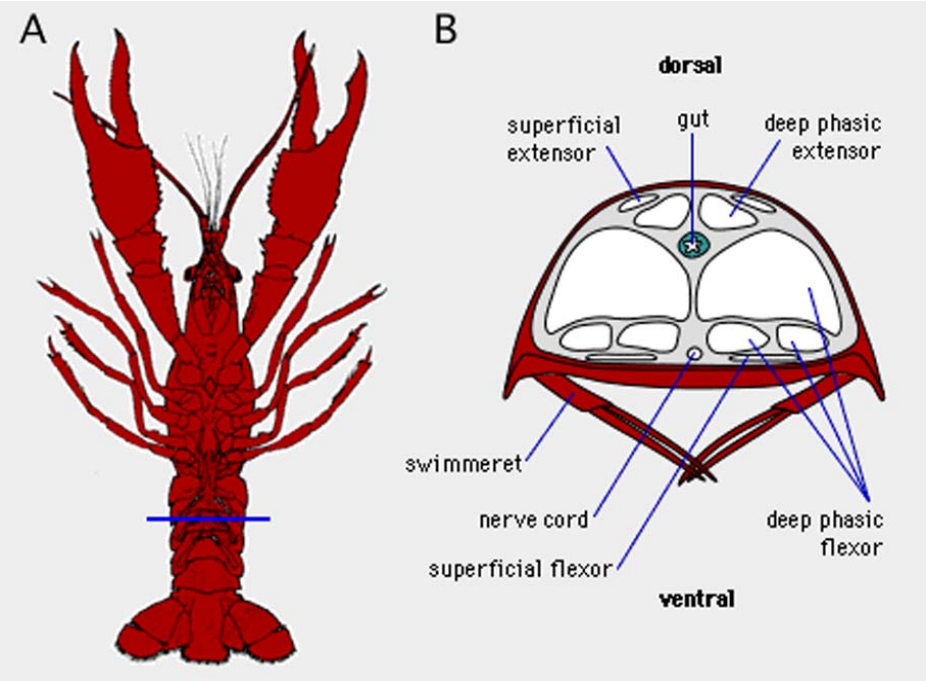

Figure 5.1 – The muscular anatomy of the crayfish tail. Illustrated here are the flexor muscles of interest in this brief study. (Wyttenbach et al 2002)

Of particular interest in this study is the superficial flexor muscle. These muscles are actively innervated by a tonic set of six distinct motorneurons (Kennedy & Takeda 1965). Work in chapter three of this dissertation illustrates and application of my

switchback polyimide array to this system. These six motorneurons are contained in a nerve bundle which appears as the third tissue offshoot of the central ganglion of the ventral nerve cord in each segment of the tail. The nerve bundle has a diameter ranging from 30 to 75□m and the individual neurons in the bundle have diameters ranging from 250nm to 900nm (Pearce 2002). The entire nerve bundle extends from the nerve cord to the muscle with approximately three millimeters of free tissue accessible for recordings directly under the ventral cuticle. This small bundle of neurons contains five excitatory neurons and one inhibitory neuron. This offers a spectrum of neuron sizes and chemical behaviors (inhibitory or excitatory).

As mentioned, this nerve is responsible for postural control in the crayfish. Central pattern generators in the crayfish ventral cord ganglion create a complex "always-on" (tonic) behavior of spiking in this nerve. This kind of behavior is common in many organisms (including humans) and is an important part of rapid reflex response systems in virtually all organisms with complex neural networks and long propagation times from central brains to peripheral components. This tonic activity can be actively modified by signals from the brain or from local feedback loops.

Many mechanical inputs to anatomical elements of the crayfish tail cause complex behavior in this nerve bundle. Mechanical stimulation of the swimmeret appendages causes suppression or down-regulation of several excitatory neurons and up-regulation of the inhibitory neuron. Mechanical stimulation of the hairs on the tail fan causes up-regulation of several excitatory neurons. In both cases, changes are relative to tonic (no stimulus) behavior of this neuron bundle.

This feedback motor network is the first exposure of students in the Cornell BioNB 491 to extracellular recordings in living systems. When combined with intracellular recording from the superficial flexor muscles, the students study a variety

of topics including multi-terminal innervation, short term synaptic plasticity, and the ionic basis of the cell resting potential (Wyttenbach et al 2002).

For the last several years, students have utilized software which discriminates action potentials based on raw signal amplitude to offline data files (Figure 5.2). This measurement is highly noise prone and offers no discrimination for the first and second or third, fourth, and fifth action potential amplitudes in the bundle.

I will illustrate the power of g-PRIME to provide both real-time and offline analysis of recorded action potentials in this nerve bundle. I will readily discriminate signals from each of the six units in the nerve and illustrate the rate behavior of each in response to external stimulus at the tail fan (excitatory) and at the swimmeret (inhibitory). In both cases, rate changes will be described relative to spontaneous activity.

5.3. Analyzing the Reflex Activity of 6 Motorneurons in the Crayfish Periphery

Materials & Methods

Adult Crayfish (approximately 4-6 inches in length) were chilled on ice for approximately 10 minutes. The tail was detached from the body and placed in a physiologic saline solution (5.4mM KCl, 205mM NaCl, 13.5mM $CaCl_2 \bullet 2H_2O$, 2.6mM $MgCl_2 \bullet 6H_2O$, 2.3mM $NaHCO_3$, and 2mM dextrose) (van Harreveld 1936). Swimmerets were removed leaving only short stumps and an incision was made in the ventral cuticle along the midline in an arbitrary tail segment. A second incision was made posterior and parallel to the sternite creating a T-shaped incision. The ventral cuticle was peeled back to reveal ventral nerve cord and the deep flexor muscle. The third nerve is clearly visible under the cuticle as an acute offshoot of the ventral nerve cord approximately one third of the way down the segment.

A glass intracellular electrode of arbitrary tip sharpness was pulled and the tip was broken back with forceps to create an opening on a size scale larger than that of the nerve. The glass electrode was inserted into a suction fitting and saline was pulled into the electrode to create a connection with the recording apparatus. The electrode was mounted in a micromanipulator and maneuvered near the nerve offshoot. Suction was applied, the nerve was pulled into the electrode tip, and the electrode was pressed against the deep flexor muscle to improve the sealing impedance (and thus the signal to noise ratio of the recordings).

Signals were pre-amplified by and AM Systems 1700 series AC-Coupled extracellular amplifier with a band pass filter allowing 300-5,000Hz signals and a gain of 1000 (60dB). Once amplified, signals were acquired and digitized using a National Instruments E-Series DAQ multifunction data acquisition card at 10kSamples/sec (PCI-6024E).

Signals were recorded and visualized using the software illustrated in the previous chapter, g-PRIME. Threshold levels were set based on amplitudes of action potentials relative to the noise level, an event window was selected of approximately 5ms, and the energy density (sum of the Fourier spectral components) was calculated for each detected event. In all cases, detected events were converted into a seven parameter representation as illustrated in the previous chapter. Both energy density and peak voltage amplitude were compared in order to compare previous techniques with those offered by the new software.

Signals were recorded under various stimulus conditions. Initially, spontaneous activity was recorded. A wooden (non-conductive) dowel was used to mechanically stimulate the tail fan hairs and swimmerets. Reflex activity was monitored and analyzed on a per-unit basis.

In all cases, neurons are referred to by number increasing in size from smallest amplitude action potential (neuron 1) to largest amplitude action potential (neuron 6).

Results

The neural system consisting of nerve 3 and its various feedback inputs for reflex activity represent a highly complex system when dealing with individual action potentials and a simple system when looked at on neuron rate behavior levels. I have applied g-PRIME in three distinct stimulus situations: spontaneous (unstimulated) activity, swimmeret stimulation, and tail fan stimulation. In each situation, characteristic behaviors are observed, relative firing rates are calculated and event vs. time traces are displayed. In each case, Specific histograms were created relating the rate behavior of neurons 3, 4, and 5 in the bundle. These neurons seemed to show particularly interesting rate behavior changes in response to the various stimuli. The rate response of all units to these anatomical stimuli is illustrated in figures (Figure 5.2, 5.3, & 5.4).

During spontaneous behavior, all neurons fire at relatively stable rates. The third amplitude neuron is the most active while the maximum amplitude (sixth) neuron has only sporadic activity and may turn off completely (Figure 5.2).

In the case of swimmeret stimulation (Figure 5.3), all neurons (except for neuron 5) seem to have down-regulated activity and the largest nerve is deactivated entirely. The system will adapt to steady pressure on the swimmeret and return to normal behavior within approximately 15sec of stimulus. Neuron amplitude 5 is up-regulated in this stimulus paradigm. It is known that the 5^{th} amplitude neuron in the bundle is the inhibitory neuron (Kennedy & Takeda 1965) so it is expected that neuron 5 would be up-regulated while the other 5 neurons are down-regulated or stopped all together.

In the case of telson (tail fan) stimulation, the system responds with an up-regulation of all firing behavior (except for neuron 5, the inhibitory neuron). Neuron 3 does not seem to display any rate change while neuron 4 increases in rate behavior to match that of neuron 3 (Figure 5.4). The rate behavior of the 6th amplitude neuron seems to increase dramatically and in the displayed data set, begins to exhibit bursting behavior.

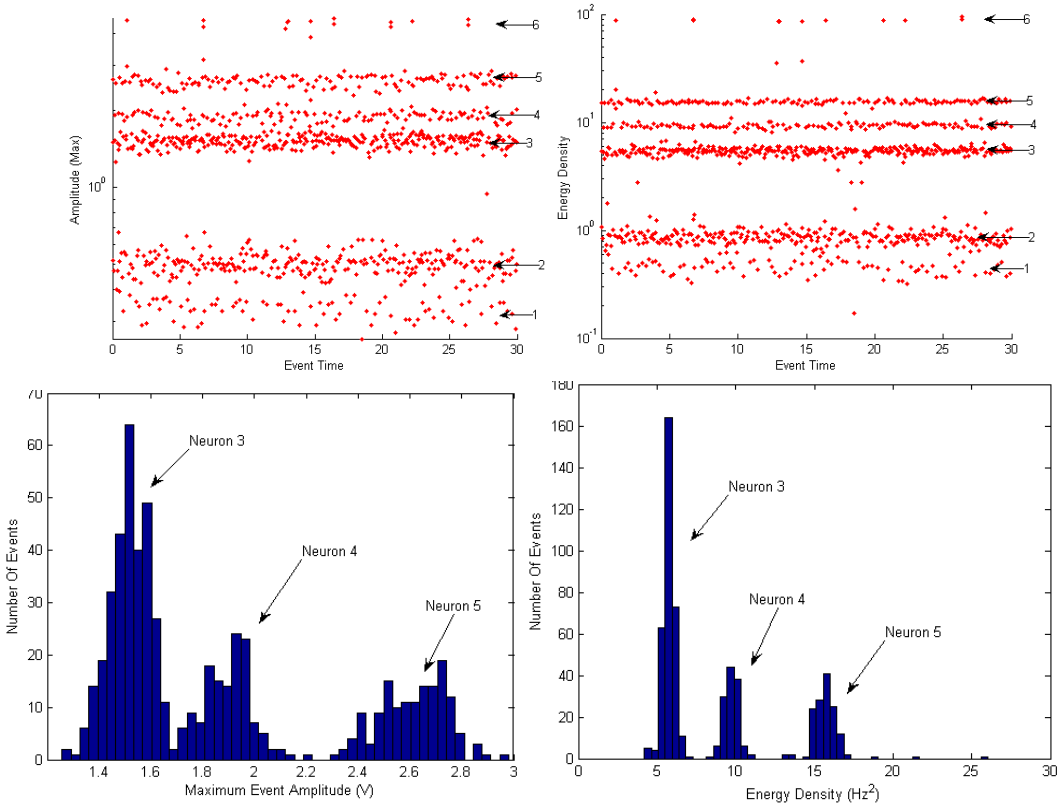

Figure 5.2 – Spontaneous (tonic) Behavior of the crayfish third nerve offshoot of the ventral cord ganglion in an arbitrary tail segment. Relative firing rates all illustrated for all neurons (top) and focus on the rate behavior (over 30 seconds) of the 3rd, 4th, and 5th neurons are displayed (bottom). The difference in result resolution between the old system, freqhisto, by Dr. Bruce Land (bottom left), and the new system, g-PRIME (top traces and bottom right).

In both stimulus cases, the rate behavior of the inhibitory neuron is only slightly changed compared to the dramatic rate changes in the other 5 neurons. There is up-regulation of the inhibitor during swimmeret stimulation. Auto-correlations were carried out with individual spike classes in each stimulus mode and no clear synchronicity was discovered between spikes in any pairs of units within 50ms of an event (Figure 5.5).

Both dual and quad bursting was observed in the largest unit in the nerve. Inter-burst intervals were on the order of several hertz while inter-spike intervals within a burst were distributed with a skewed Gaussian distribution with mean at approximately 150Hz (Figure 5.6).

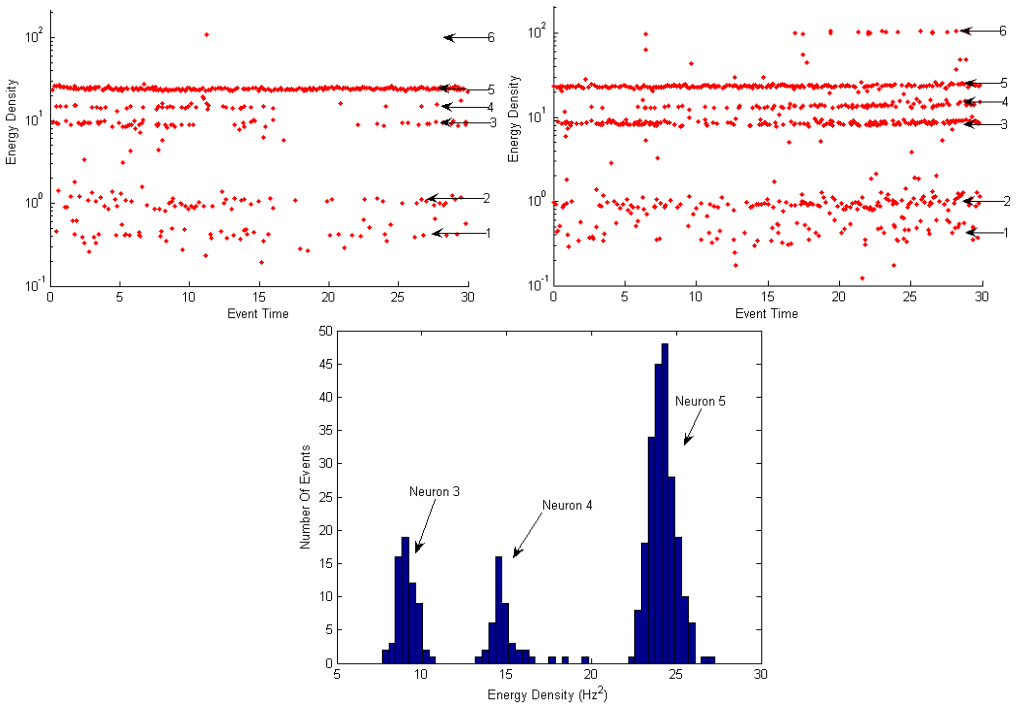

Figure 5.3- Swimmeret Stimulation illustrates a suppression of all activity in neurons except for neuron 5, the inhibitor. Neuron 5 is significantly upregulated compared to the spontaneous firing regime. Recovery from swimerette stimulus is illustrated (top right).

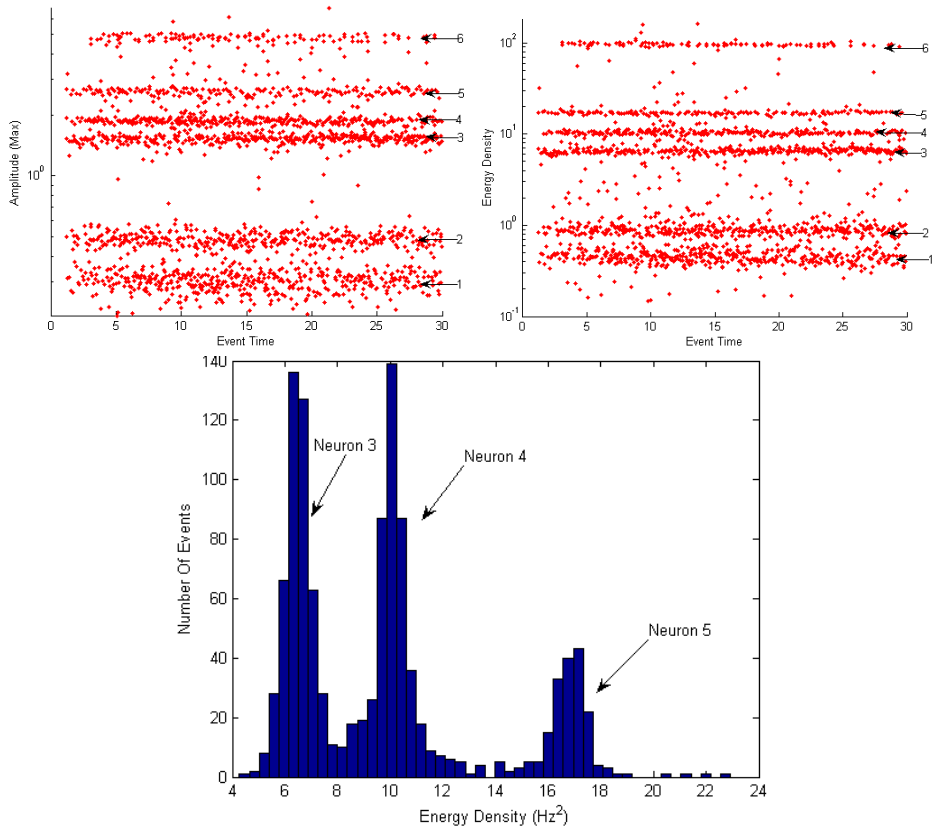

Figure 5.4 – Tail Fan Stimulation produces increased activity in all neurons except for neuron 5 and neuron 3. Neuron 3 and 5 remain unmodified. The peak amplitude of detected events is illustrated (top left) and the energy density of events is illustrated (top right). Events detected between neuron amplitude 2 and 3 (top right) indicate spike overlaps which are not apparent in the peak only plot (top left). This causes false positive event detection in amplitude thresholding when overlaps are involved.

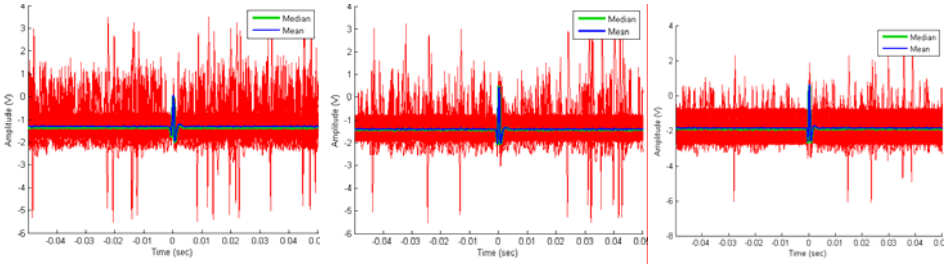

Figure 5.5 – Examples of spike autocorrelation of neurons 3 (top left), 4 (top right), and 5 (bottom) for spontaneous activity. No clear temporal correlations between spikes or within a single spike class are observed within a 100ms window centered on the detected events. This was repeated in each stimulus paradigm yielding the same results.

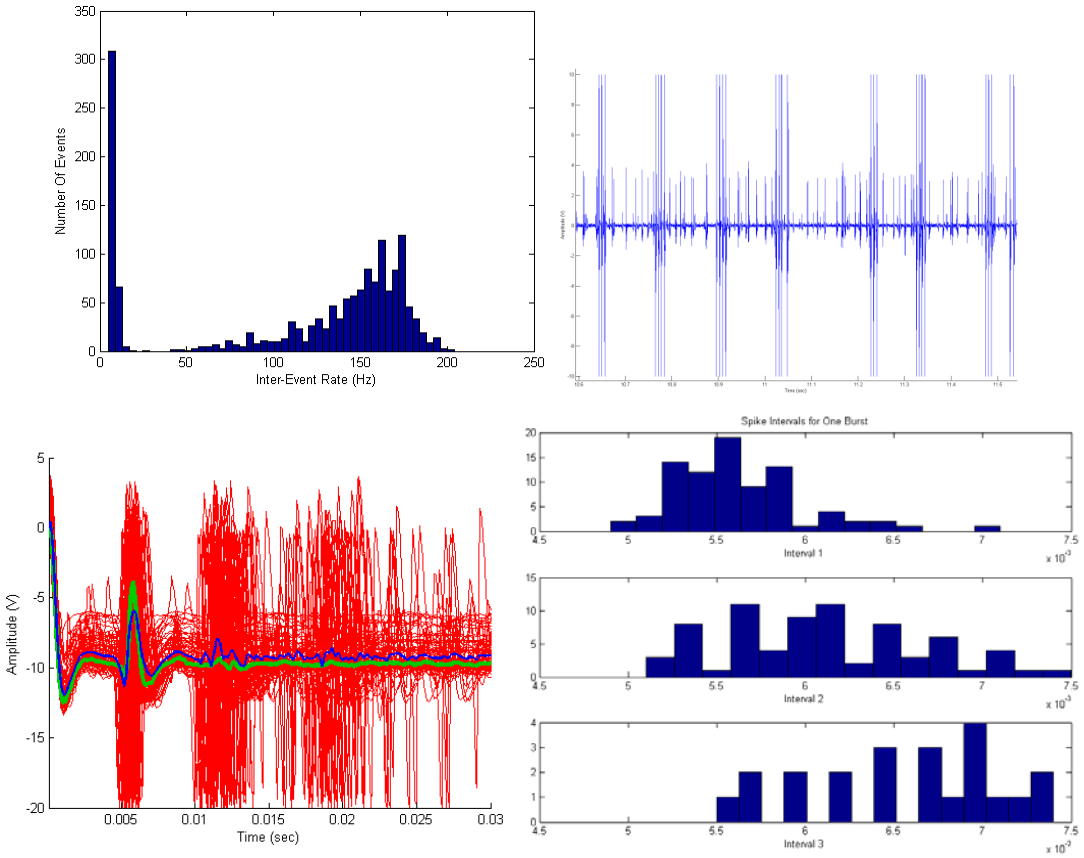

Figure 5.6 - Bursting Behavior of Neuron 6 is illustrated. The general rate distribution for inter-spike intervals in the burst demonstrates the low frequency inter-burst interval and the distribution of high frequency inter-spike intervals (top left). The raw trace is demonstrated (top right). Correlation of the initial spike of a burst with the rest of the burst illustrates a decreasing delay and correlation between spikes in a single burst (bottom left) and the interval between spikes in the burst shows a steady decay as more spikes are produced (bottom right).

5.4. Two Crayfish Proprioceptor Neurons, Sensory Adaption, and Rate Sensitivity

On the dorsal side of the crayfish tail there are two superficial extensor muscles considerably smaller than the deep flexor muscles. These muscles are tasked with stretching the tail during the recovery phase of the escape response tail pulse in the crayfish. In order to determine the location of the tail, the crayfish has two stretch receptor neurons which encode the amount and rate of stretch of the tail.

In each tail segment there are two stretch receptor neurons on each side of the organism and connect into the ventral nerve cord through the second nerve offshoot. These neurons constitute an example of the somatic sense of proprioception (knowing where body components are). The difference between the two neurons is their adaption rates to stretch. Each neuron synapses to the superficial extensor muscles such that stretches in the muscle translate into stretches in the muscle receptor organ (MRO). Mechanically gated ion channels cause membrane depolarization when the cell is stretched and activate action potentials (McCarthy & McMillan 1995).

The differences between the two MROs in a given muscle have to do with adaption rates to stimulus. The smaller neuron (MRO1) produces action potentials in response to a wide range of slow stretches of the tail. The larger neuron (MRO2) is only recruited during rapid stretch behavior (such as the escape response flex). The sensory neurons adapt to stimulus through cellular mechanisms which counter the effect of the mechanically gated ion channels (Swerup 1983). Through this process, the crayfish tail can encode slow tail stretches and fast tail stretches with high dynamic range.

This system offers an ideal teaching environment for and introduction to sensory neuroscience. Adaption occurs in virtually all sensory systems found in nature. In general, sensory neurons detect changes in stimulus versus static stimulus levels (think of how the nose will adapt to a smell over time). The students can see/hear a clear spike rate representation of a stimulus when they physically flex the tail, may record and characterize the behavior, and see adaption over a period of several seconds. The preparation in the crayfish makes the neurons trivially accessible even with the most basic tools. In addition to the accessibility of the neuroanatomy, the action potentials are very large and require little amplification.

The g-PRIME software may acquire signals from an acquisition input as readily available as the sound card input and may discriminate MRO action potentials based on peak amplitude. While my software does not produce curve fits, the rate behavior of the sensory signals may be extracted and loaded into a third party program such as MATLAB or Microsoft Excel in order to calculate peak rates and adaptation parameters.

I will illustrate recordings from the stretch receptor neurons at a variety of stretch amplitudes and illustrate the different adaptation rates between MRO1 and MRO2 to provide a picture of a relatively simple yet high dynamic range sensory system.

5.5. Analyzing Sensitivity and Adaption Rates of Two Muscle Receptor Organs

Materials & Methods

Adult Crayfish were prepared similar to the method described in section 5.3 for the multi-unit nerve recordings. Instead of creating careful incisions to reveal the ventral nerve chord, scissors were utilized to cut along the carapace allowing for the ventral cuticle and sternites to be peeled back and removed. A thumb or finger is inserted from the anterior end of the severed tail and the muscle tissue from the ventral surface to the deep flexor and the gut are extracted leaving only the superficial extensor muscle in the tail. This process should leave severed neural connectives floating free in the saline at the periphery of the tail segment along the sidewall along the convex side of a carapace tail segment.

A suction electrode (as described in 5.3) is used to capture the free nerve process. A small length of thread is attached to the tail fan by puncturing a hole in the center fan segment. This thread is attached to a manipulator such that the tail may be stretched to arbitrary distances during the experiment.

Signals were acquired as described in section 5.3 and pairs of amplitude thresholds were used to discriminate between the two muscle receptor organs. The rate behavior

of the events was extracted using the g-PRIME analysis features and the data was fit to a curve (using the MATLAB Curve Fitting Toolbox).

Results

Examples of extracted adaptation behavior of both MRO1 and MRO2 are illustrated in Figure 5.7. In the case of MRO1, the rate behavior seems to include both a slow and fast adapting component for a highly accurate exponential fit (R-square = 0.99). This indicates a primary adaption component with a time constant of 217ms. The slow adapting component of this system expresses a time constant on the order of 3.2s. The time constants for the adaption of MRO1 are relatively easy to calculate due to the response range of the neuron. The process expresses adaption behavior in a range of stretch rates that can be readily achieved by slow pulling on the tail.

MRO2 expressed an adaption rate approximately 3 times faster than that found in MRO1. The decay was complex and rapid so a fit was problematic, but a time constant of decay seems to be on the order of 70ms. Confidence of this recording is low due to the fast flexing required to generate adapting behavior.

The illustration of these distinct firing rates represents the cause for all of the behavior of these two proprioceptor neurons. The fast adaption rate (a cellular/molecular mechanism) in MRO2 causes the cell to respond only to flex rates much larger than that of MRO1 (i.e. faster stretch is required to overcome the fast adaptation rate of the cell). This allows for a higher dynamic range of rate encoding in the crayfish than would be available with one cell.

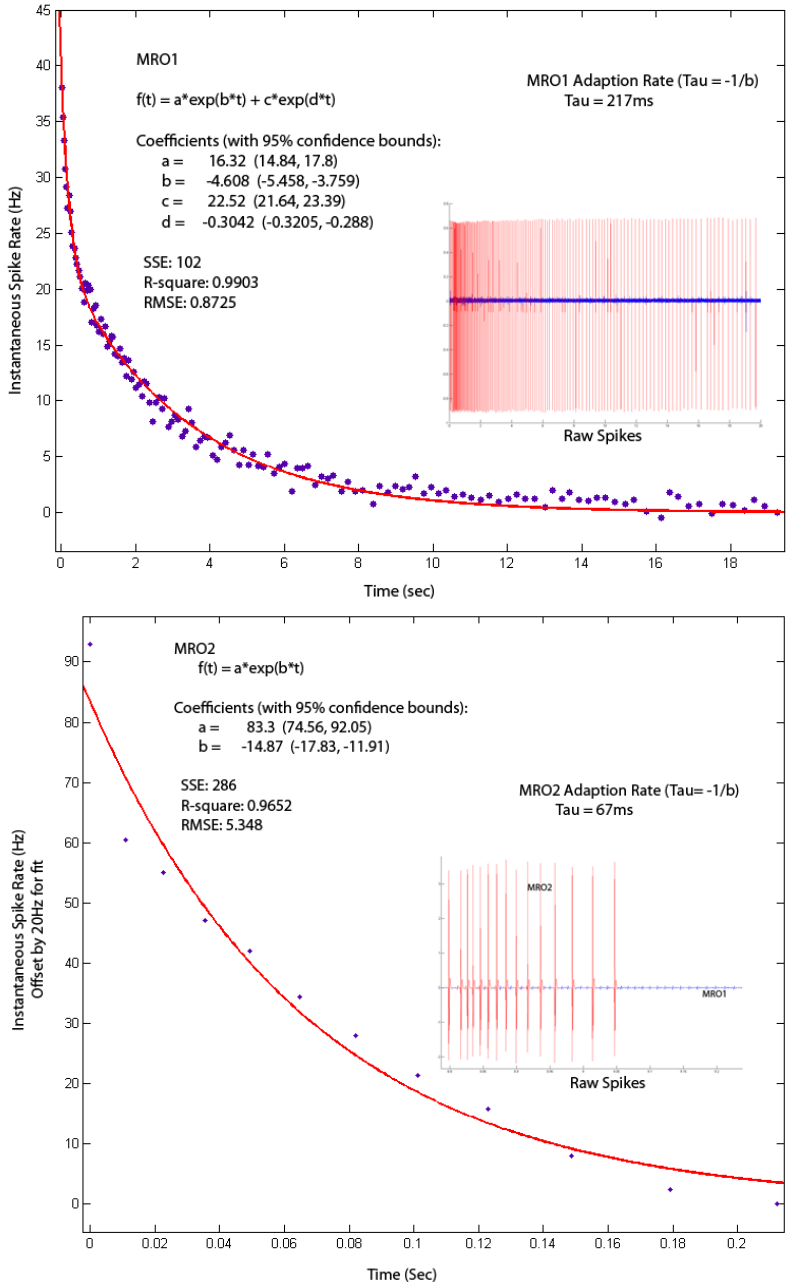

Figure 5.7 – Adaption rates of the two MRO Neurons in the crayfish superficial extensor muscle system. The best fit to the MRO1 data (top) seemed to require a pair of exponentially decaying components indicating the possibility of a long and short temporal component to the MRO1 adaption process. It was hard to find an example of MRO2 with more than 2 or 3 spikes and this example does not fit very closely to an exponential so the confidence bounds on the MRO2 decay rate are large (bottom).

5.6. Software Application Discussion

Here I have expressed the capacity of the g-PRIME software to produce insights into a multi-unit motor-neuron system with complex behavior and into the rate behavior of a relatively simple two-element proprioceptor sensory system. These applications are extracellular recordings from crayfish neurons, but the software is not limited to this type of recording or analysis. As was illustrated in chapter 4, g-PRIME contains powerful event analysis tools that may detect any type of discrete event and provide several useful analysis parameters to differentiate signal sources in a 9 dimensional space (see section 4.5).

All of the data illustrated in this chapter may be visualized in real time by students in a laboratory environment or replayed into a sound card (or other data acquisition device) for analysis in an environment where amplifiers are not available. The clarity of the adaption rate plots and the distinct energy density levels in the above explorations translates the student through the instrumentation and connects them directly to the behavior of the physiology they are studying.

As one can imagine, this software is not merely limited to educational environments. These powerful tools are in application across a broad range of cellular, molecular and organismal applications in both teaching and research environments. A few example applications of this powerful tool are:

1. *Application in Mildly Electric Fish*

   The laboratory of Dr. Carl Hopkins at Cornell University is applying the g-PRIME software suite to a study of mating behavior in a variety of mildly electric fish. Electric organ discharges (EODs) have characteristic frequencies between species and individual animals may be readily discriminated using the real time or offline “peak frequency” detection component of the software.

2. *Application in Fruit Fly Genetics and Mating Behavior*

Dustin Rubenstein of the laboratory of Dr. Ron Hoy at Cornell is applying the software to a study of the courtship behavior of the fruit fly (*Drosophila Melanogaster*). He uses the event threshold tools to discriminate peaks of tapping at rates around 30Hz. This rate may vary based on several genetic mechanisms. g-PRIME is allowing him to rapidly record and access rate behavior of the animals during their courtship behavior.

3. *Application in a Study of the Dynamics of Neurotransmitter Release (Crayfish)*

Students in the Spring 2007 Neurophysiology Lab course (BioNB491) are applying the g-PRIME software to their final projects for the class. They are studying the initial slope of elicited post synaptic potentials (PSP) in the crayfish neuromuscular junction (as illustrated in figure 4.7). The initial slope of the PSP is directly proportional to the amount of transmitter released into the synapse for muscle innervation. The students are monitoring variation of this initial slope as a function of various synaptic enhancement and depression pharmacological agents.

CHAPTER 6: APPENDICES

APPENDIX 1

TREADMILL ASSEMBLY CODE

Introduction

This appendix deconstructs my custom written Atmel Assembly code associated with the treadmill time critical data acquisition system described in chapter two. This assembly code is written for operation in an Atmel AT90S8515 8-bit RISC microcontroller driven by an external 11.0592MHz Oscillator.

The structure of this code takes advantage of the internal asynchronous 16-bit timer/counters and the Universal Asynchronous Receive and Transmit Protocol (UART). The main program initializes, defines a variety of setup parameters, and then is completely interrupt driven while the processor sits in an infinitely looping main program sequence. Therefore, the structure of the code can be expresses as follows:

- Initial Component Configuration and Interrupt Activation
- Interrupt Subroutines
    - UART Receive Complete Interrupt (Data from the User)
    - Timer/Counter1 Compare Interrupt (Initiate a Sample Period)

Assembly Commands Used

This program was written in Atmel AVR-Studio as a raw assembly file. The commands used are described in the AVR 8-bit RISC processor instruction set summary available from Atmel Inc (www.atmel.com). The following list of commands describes the basic operations utilized in this program.

Table A1.1 – Utilized Components of the Atmel Instruction Set

rjmp	*Relative Jump* Move to a certain location in the program	**add**	*Add Registers* Add two registers, results loaded into first register
reti	*Return from Interrupt* Return to jump point in program after interrupt processing. Enable interrupts	**andi**	*And Immediate* Logical AND operation between a register and a constant
clr	*Clear Register* Set register contents to 0x00	**cpi**	*Compare Immediate* Compare register to constant
ser	*Set Register* Set register contents to 0xff	**breq**	*Branch if Equal* Jump to location if compare result is zero
ldi	*Load Immediate* Load a constant into a register	**sbrc**	*Skip if Bit is Clear* Skip next command if register bit is clear
out	*Push an I/O register State* Place a value into a configure, status, or port register for I/O	**sbrs**	*Skip if Bit is Set* Skip next command if register bit is set
in	*Read an I/O register State* Load an I/O register value into target register	**brsh**	*Branch if Same or Higher* Branch to a target if compare result is positive
rcall	*Call a Subroutine* Jump to a target location in memory (farther than **rjmp**)	**sbr**	*Set Bit in Register* Does not apply to I/O registers
sei	*Set Interrupts* Activate Global Interrupt Flag	**cbr**	*Clear Bit in Register*
cli	*Clear Interrupts* Deactivate Global Interrupt Flag	**ori**	*Or Immediate* Logical OR between register and constant
sbi	*Set Bit in I/O Register*	**mov**	*Mov Register* Copy register contents
cbi	*Clear Bit in I/O Register*	**nop**	*No Operation* 1 Cycle, Change Nothing

Reset and Initial System Configuration

At system startup, the controller executes the command at address 0x0000 in program memory. This is a subroutine vector pointing to the reset interrupt subroutine and the main program will continue to loop indefinitely while responding to other interrupts. This configuration software carries out the following tasks:

- Define Pointer Names for Used Registers (Assembler Directives)
- Initialize the Interrupt Subroutine Vector Table
    - RESET, Timer/Counter1 Compare, UART Read Complete Interrupts
- Initialize Data Registers, I/O Ports, and Memory Pointers
- Configure I/O Port Direction, Configure UART, Configure Timer
- Wait for the ADNS Camera Chip to Initialize (100ms)
- Connect to the ADNS Chip, Input Basic Configuration and Reset Camera
- Determine ADNS Chip Model (In this application, 2620)

The assembly code to carry out these tasks is contained in the next two pages in two-column format. Actual assembly code or assembler directives crucial to the code are listed in bold while comments describing the various specific actions are in italics and preceded in all cases by a semicolon (;). This will be the standard for code illustration in this appendix. Raw assembly code files may be found in the supplementary web material.

```
;**********Flyball X**********
; This code will interface with the
; agilent ADNS series micro-camera
; via a clocked serial interface and transmit
; delta x and delta y
; values over an RS232 interface to a computer.
;
; The user may control sample rate and
; acquisition start by the same software interface
;
; Written by Gus K. Lott III,
; (c)November 0x7D4
; Member of the Hoy Research Group
; GKL6@cornell.edu
;
; Serial interface baud rate of 115200
; Clock frequency 11.059 MHz (90.4ns cycles)

.include "C:\AVRTOOLS\8515def.inc"
;**** Define Registers
.def chipID         =r16
.def deltaX         =r17
;ADNS register containing Delta_X info

.def deltaY         =r18
;ADNS register containing Delta_Y info

.def config         =r19
;ADNS config register

.def pixeldump      =r20
;ADNS register for pixeldump request

.def pixeldata      =r21
;ADNS register to read the pixeldata from

.def serial         =r22
;Byte to be sent out or read in from the UART

.def mask           =r23
.def temp           =r24

.def timer          =r25
;8 bit timer target

.def sgo                =r26
; Trigger Line & Data Acquisition
; State 0x0 or 0xff

.def adns           =r27
.def dx             =r28
.def dy             =r29


; This is the initial area of program memory
; containing interrupt subroutine pointers

.cseg
.org $0000
rjmp RESET  ;Initialization routine
reti        ;Unused interrupts
reti
reti
rjmp GetDelta   ;Timer/Counter1 Compare
reti
reti
reti
reti
rjmp ByteIn ;Serial read complete interrupt
reti
reti
reti
```

```
;Reset Subroutine
RESET:

;Init Registers
clr dx
clr dy
clr sgo
ser chipID

;ldi temp, 0xff
;out DDRB, temp
;ldi temp, 0xaa
;out PORTB, temp

;ADNS-2620 registers
ldi deltaX,0x43
ldi deltaY,0x42
ldi config,0x40
ldi pixeldump,0x48
ldi pixeldata,0x48

;Setup Stack Pointer
ldi temp, LOW(RAMEND)
out SPL, temp
ldi temp, HIGH(RAMEND)
out SPH, temp

;Setup PORTs
ldi temp,0xff
out DDRB,temp       ;OUT
out DDRD,temp       ;OUT
ldi temp,0x01
out PORTB,temp
;Raise SCK for ADNS, Trigger low

;Setup UART
ldi temp,5
out UBRR,temp
;Set Baudrate at 115.2k for
11.0592MHz Crystal
ldi temp,0x98
out UCR,temp
;Enable UART I/O and RX
Complete interrupts
ldi temp,115  ;output an ASCII s on
startup
out UDR,temp

;Setup 16-Bit Timer
ldi temp,0
out TCCR1A,temp
ldi temp,8
out TCCR1B,temp
;Enable compareA interrupt and turn
counter off
ldi temp,0
out TCNT1H,temp   ;Clear Timer
out TCNT1L,temp   ;Clear Timer
ldi temp,20
out OCR1AH,temp   ;2160Hz
ldi temp,0
out OCR1AL,temp
ldi temp,64
out TIMSK,temp
;activate compareA interrupt

;**********************************
***
;SET UP ADNS CHIP

;pause >50ms
ldi temp,5
;Set clock prescaling to 1024 cycles per
tick
out TCCR0,temp
ldi timer,216  ;216*1024/CK=20ms
rcall Pause          ;20ms
rcall Pause          ;20ms
rcall Pause          ;20ms
rcall Pause          ;20ms

;Write a configuration bit
mov adns,config
;Config register Location
ori adns,128
;Bit 7 set = next will write
rcall WriteADNS
;Pause 100us
```

```
ldi temp,2
;Set clock prescaling to 8 cycles
per tick
out TCCR0,temp
ldi timer,145
rcall Pause
;Write configuration settings
ldi adns,1
;Normal Operation and LED
always on
rcall WriteADNS

;pause >50ms
ldi temp,5
;Set clock prescaling to 1024 cycles
per tick
out TCCR0,temp
ldi timer,216  ;216*1024/CK=20ms
rcall Pause    ;20ms
rcall Pause    ;20ms
rcall Pause    ;20ms
rcall Pause    ;20ms


;Determine Part ID and
;command register locations
rcall IDCheck

;Activate Interrupts
sei

ProgLoop:
rjmp ProgLoop      ;Loop Forever
```

Data Acquisition Sample Period

When the internal 16-bit Timer/Counter1 reaches its target counting value, a data acquisition period is initiated. In the *Ormia* application, our sample period was approximately 463□s (2160Hz). A full sample period must be completed within this window with remaining time to check for and process possible user control input.

The main time limitations in the data acquisition process exist in two separate processes. First, the Agilent camera chip requires a 100□s delay to prepare the data line for output once a read is requested. The second time bottleneck is the time it takes to broadcast the data byte and header byte over the UART at 115.2kBaud. Fortunately, both of these processes may be decoupled from the main program since they take place in separate architectures.

The final structure of the main program loop at 2160Hz is illustrated in Figure A1.1. Data is transmitted over the UART line concurrent to the ADNS microprocessors preparation of the motion data for acquisition.

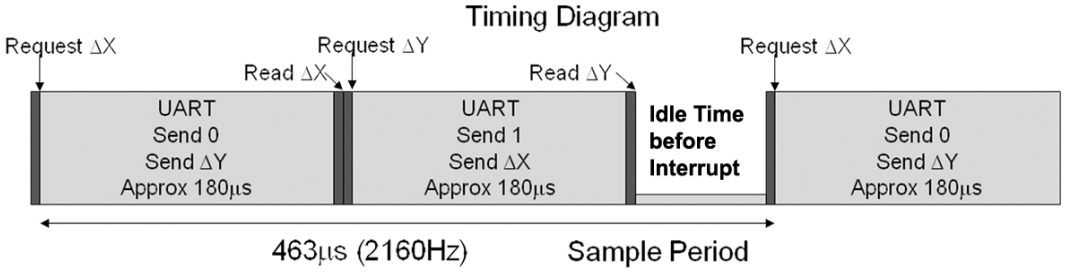

Figure A1.1 Timing diagram for a single sample period of motion data. Major delay and process overlapping are illustrated in order to demonstrate that the timer interrupt period is not overrun.

This timer interrupt subroutine accomplishes the following tasks:

- Request a Delta-X sample from the ADNS chip
- Broadcast a Delta-Y sample from a previous sample period
- Request a Delta-Y sample from the ADNS chip
- Broadcast a Delta-X sample from this sample period
- Read and store a Delta-Y sample for the next sample period

In addition to the raw data acquisition and transfer to the PC based user, this subroutine checks the UART status register for a "byte received flag" which would indicate that a command from the user had arrived during the sampling period. During a data acquisition period, other interrupts are deactivated in order to ensure accurate execution times.

There are specific references to the subroutines named "WriteADNS" and "ReadADNS" which handle writing to and reading from the camera chip respectively. The details of these subroutines will be illustrated in the next section of this appendix. The assembly code for the data acquisition period follows.

```
;#############################
####
;Handle the data acquisition and
transmission
GetDelta:

sbi PORTB,1
;Trigger high during acquisition

;**************************
*********
;REQUEST DX
ser temp
out PORTD,temp
out DDRD,temp
;Set SDIO (PORTD4)to Write to
the ADNS chip

mov adns,deltaX
;Load in the deltaX register
address
;to the ADNS data register for
transmission
andi adns,127
;set first bit to zero indicating we
want to read
rcall WriteADNS
;Send request to chip for deltaX

clr temp
out PORTD,temp
;Set SDIO to high z state
out DDRD,temp
;Set SDIO to read from the ADNS
chip

;**************************
********
;SEND DY OVER UART
ldi serial,0
rcall UARTOut       ;Preceed a
dy byte with 0x00

mov serial,dy
rcall UARTOut
;if sgo is high, send dy byte on serial port

;**********************************
***
;READ DX
rcall ReadADNS
;Read a byte from the Agilent chip
mov dx,adns

ser temp
out DDRD,temp
;Set SDIO (PORTD4)to Write to the ADNS
chip

ldi temp, 0x80
add dx,temp
;Convert new dx from 2s compliment
;to unsigned binary

;Make sure DX is not 0 or 1
; (these are headers for data points)
cpi dx,3
brsh pBx
ldi dx,2
pBx:

;**********************************
***
;Request dY
mov adns,deltaY
andi adns,0b01111111
;set first bit to zero indicating we want to
read
rcall WriteADNS

clr temp
out PORTD,temp
;Set SDIO to high z state
out DDRD,temp
;Set SDIO to read from the ADNS chip

;**********************************
***
;Send DY over UART
ldi serial,1
```

```
rcall UARTOut       ;Preceed a dx byte with 0x01

mov serial,dx
rcall UARTOut       ;send dx byte on serial port

;*****************************************
;Read dY
rcall ReadADNS
mov dy,adns

ldi temp, 0x80
add dy,temp
;Make sure DY is not 0 or 1
cpi dy,3
brsh pBy
ldi dy,2

pBy:

ser temp
out PORTD,temp
out DDRD,temp
;Set SDIO to Write to the ADNS chip

;*****************************************
;Check for a byte in during cli period
;Check bit 7 of USR.
;If set, we missed a byte during getdelta
in temp,USR
andi temp,128
cpi temp,128
breq ByteIn

reti
```

Interface to the ADNS Camera Chip

The Agilent ADNS series camera chips utilize a non-standard 2-wire interface for which there is no built-in protocol onboard the architecture of the 8515. The protocol consists of simple addressed register read/write operation. The user initially sends the register address of the target internal data or configuration register and indicates (bit 1) whether a read (0) or write (1) is desired. If the user wishes to write to the register, he may immediately write the contents of the register over the data line. If the user wishes to read from the register, he must convert the 8515's I/O port to high impedance input mode and wait 100□s for the ADNS chip to convert its I/O port state and prepare the data.

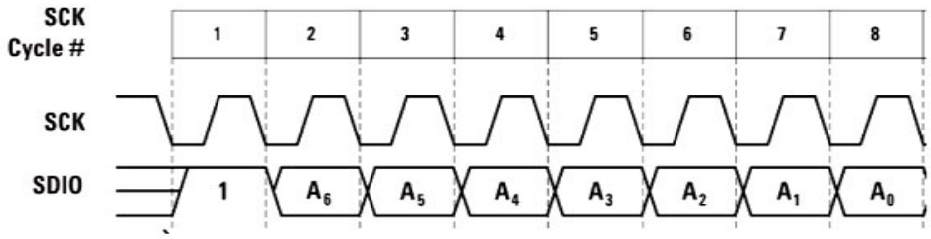

Figure A1.2 A single byte being clocked into the ADNS chip utilizing a non-standard Agilent 2-wire interface. An address input byte (bit 1 = 1) demonstrates a desire to write to the addressed register.

Beyond the details of the actual contents of the communication stream, the hardware level process involves simply clocking data bits into a serial format using a 500ns period logical pulse illustrated in Figure A1.2 (from the ADNS-2620 data sheet). The figure illustrates exactly the process executed for a write operation in the following code. The main distinction between the "*ReadADNS*" and "*WriteADNS*" subroutines is that the write function empties the contents of a register onto the data bus while the read function acquires the data bits into a register from the data bus. In both cases, this register is referred to as "adns" in the assembly code. I/O port read/write state is controlled in the routine that calls these functions. Port state must be high impedance input for reading and current source for writing.

Note that the basic clock period of the system is 90.4ns (11.0592 MHz) and "nop" (no operation) represents a single cycle and is used to pad bit widths at the I/O port.

```
;##############################
;Subroutine to Write a Byte to the
 ; ADNS Chip
; based on the interface protocol in
; the ADNS datasheet.
;Byte to send is stored in adns
WriteADNS:
        nop
        nop
        nop
        nop
cbi PORTB,0 ;Drop SCK
        sbrc adns,7
        ser temp
        sbrs adns,7
        clr temp
        out PORTD,temp  ;SDIO
Value
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        nop
cbi PORTB,0 ;Drop SCK
        sbrc adns,6
        ser temp
        sbrs adns,6
        clr temp
        out PORTD,temp
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        nop
sbi PORTB,0 ;Drop SCK
        sbrc adns,5
        ser temp
        sbrs adns,5
        clr temp
        out PORTD,temp
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        nop
cbi PORTB,0 ;Drop SCK
        sbrc adns,4
        ser temp
        sbrs adns,4
        clr temp
        out PORTD,temp
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        nop
cbi PORTB,0 ;Drop SCK
        sbrc adns,3
        ser temp
        sbrs adns,3
        clr temp
        out PORTD,temp
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        nop
cbi PORTB,0 ;Drop SCK
        sbrc adns,2
        ser temp
        sbrs adns,2
        clr temp
        out PORTD,temp
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        nop
cbi PORTB,0 ;Drop SCK
        sbrc adns,1
        ser temp
        sbrs adns,1
        clr temp
        out PORTD,temp
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        nop
cbi PORTB,0 ;Drop SCK
```

```
        sbrc adns,0                     ret     ;Return from subroutine
        ser temp
        sbrs adns,0
        clr temp
        out PORTD,temp
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        nop
```

```
;#############################
;Subroutine to Read a Byte from
the ADNS Chip
;Byte is stored in adns
ReadADNS:
        clr adns  ;Initialize read
register
        nop
        nop
        nop
cbi PORTB,0 ;Drop SCK
        nop
        nop
        nop
        nop
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        in temp, PIND
cbi PORTB,0 ;Drop SCK
        nop
        sbrc temp,4
        ;Bit 4 (PIND4) is SDIO
        sbr adns,128
        ;if SDIO is high, set bit in
register
        nop
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        in temp, PIND
cbi PORTB,0 ;Drop SCK
        nop
        sbrc temp,4
        sbr adns,64
        nop
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        in temp, PIND
cbi PORTB,0 ;Drop SCK
        nop
        sbrc temp,4
        sbr adns,32
        nop
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        in temp, PIND
cbi PORTB,0 ;Drop SCK
        nop
        sbrc temp,4
        sbr adns,16
        nop
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        in temp, PIND
cbi PORTB,0 ;Drop SCK
        nop
        sbrc temp,4
        sbr adns,8
        nop
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        in temp, PIND
cbi PORTB,0 ;Drop SCK
        nop
        sbrc temp,4
        sbr adns,4
        nop
sbi PORTB,0 ;Raise SCK
        nop
        nop
        nop
        in temp, PIND
cbi PORTB,0 ;Drop SCK
        nop
        sbrc temp,4
        sbr adns,2
        nop
sbi PORTB,0 ;Raise SCK
        nop
```

```
nop                                 sbr adns,1
nop                                 nop
in temp, PIND
sbrc temp,4                         ret     ;return from subroutine
```

Initializing Data Acquisition & Cleaning Up at End of Session

Two single byte commands may be issued by the user to start or stop the data acquisition stream (0xff and 0xfe). If the user wishes to start the data acquisition stream, the program will connect to the ADNS chip and clear the motion data accumulation registers to ensure that motion captured corresponds to the initiation of the sampling period only. Once the internal motion accumulator registers have been cleared, the system activates the internal timer to define the sample period.

Given the structure of the data acquisition sample period, the Y value for a given time period is not transmitted to the user until the beginning of the next sample period. If the user wishes to stop sampling, the system must finish transmitting this final data point to the user (since there will be no next sampling period).

These two functions are accomplished by the "PreClean" and "SendLast" subroutines.

```
;*******************
;Clean out the DX and DY registers in ADNS
; before acquisition starts
PreClean:

;READ VALUE OF DX
ldi temp,0x10
out DDRD,temp
;Set SDIO (PORTD4)to Write to the ADNS chip
mov adns,deltaX
andi adns,0b01111111 ;read bit mask
rcall WriteADNS
;Send request to chip for deltaX
ldi temp,0x00
out DDRD,temp
;Set SDIO (PORTD4)to read from ADNS chip
cbi PORTD,4
;Set SDIO to 0 (tristate Z in read mode)

ldi temp,2
out TCCR0,temp
ldi timer,139  ;100us
rcall Pause

rcall ReadADNS
ldi temp,0x10
out DDRD,temp
;Set SDIO (PORTD4)to Write to the ADNS chip

nop
nop
nop

mov adns,deltaY
andi adns,0b01111111        ;read bit mask

rcall WriteADNS ;request DY
ldi temp,0x00
out DDRD,temp
;Set SDIO (PORTD4)to read from ADNS chip

rcall Pause

rcall ReadADNS
ldi temp,0x10
out DDRD,temp
;Set SDIO (PORTD4)to Write to the ADNS chip

ldi dy,0
ldi dx,0

ldi temp,9
out TCCR1B,temp
;turn on timer/counter1
rjmp ByteInDrop

StopSend:
mov temp,sgo
ldi sgo,0x00
cbi PORTB,1 ;Trigger low on stop

;if temp is now ff,
;       the stop byte transitioned sgo to low
;if temp is now 00,
;       the stop was sent when already stopped

cpi temp,0xff
breq SendLast
;Data Recording ends and
;send final DY value to cleanup.
rjmp ByteInDrop
```

```
                                    ;turn off timer/counter1
;############################
;If sgo is dropping to zero,        ldi serial,0
;we want to send the last DY before rcall UARTOut
quitting.                          mov serial,dy
SendLast:                          rcall UARTOut

ldi temp,8                         rjmp ByteInDrop
out TCCR1B,temp
```

Handling User Input

A PC based user may send single byte commands across the RS-232 serial line. When a byte arrives at the controller's UART data register, the assembly instructs the system to analyze the command code and branch to a certain subroutine if permissible. 8-bit values range between 0 and 255. The command codes and associated functions are listed in table A1.2. This subroutine is a series of compare and branching statements. If sampling is active, the system will not all for any function activation other than a sample stop request. Conditional branch statements in AVR Assembly have a limited reach in the program space, so they branch to a table that utilizes the farther reaching "rjmp" function in order to reach the actual command subroutine.

Table A1.2 – Byte Codes and Associated Functions for the Treadmill System

Byte Value	**Action**
255	Initiate data transmission and set trigger line high
254	End data acquisition and set trigger line low (default state at startup)
253-20	Sets frequency of motion acquisition to 11.0592MHz/(n*256)
5-19	*(does nothing) Reserved*
4	Return ADNS chip ID (ASCII 4 character String)
3*	Change RS232 Baud Rate
2	Dump a single image (see video dump section)
1*	Read a byte from the ADNS camera chip
0**	Write a byte to the ADNS camera chip
*Must be followed by a second byte: the address to the register to read **Must be followed by 2 bytes: the address and contents of the resister to write	

```
;############################
;Switchyard to handle input bytes
ByteIn:
;*************************
;Check to see if there was an error in the
;recieved data
in serial,UDR          ;read in byte
cpi serial,255
breq ByteIn255
;Does the user wish to START sending?

cpi serial,254
breq ByteIn254
;Does the user wish to STOP sending data?
cpi sgo,255
breq ByteInDrop
;User can not interrupt a data stream
;to preserve samplerate

cpi serial,0
;Does the user wish to send an arbitrary
;byte to the camera chip?
breq ByteIn0

cpi serial,1
breq ByteIn1
;Does the user wish to read a byte
;from the camera chip?

cpi serial,2
breq ByteIn2
;Is it a request for a pixel dump?

cpi serial,3
breq ByteIn3
;Does the user wish to change
;the UART baudrate?

cpi serial,4
breq ByteIn4  ;ChipID spit over UART

cpi serial,20
brlo ByteInDrop
;Do not set samplerate above
;20*256*90.4ns = 2160

out OCR1AH,serial
;Set the sample period to a user defined value
;between 4*256 and 254*256 cycles

ByteInDrop:
;Check to see a new byte arrived during
; operations and rjmp to ByteIn and repeat
in temp,USR
andi temp,128
ldi mask,0
cpse temp,mask
rjmp ByteIn

reti
;For branching, branch commands only have
;a range of 64 instructions
;rjmp has a range of 2k in either direction
ByteIn0:
        rjmp SendArbitrary
ByteIn1:
        rcall IDCheck
        rjmp ReadArbitrary
ByteIn2:
        rcall IDCheck
        rjmp Video
ByteIn3:
        rjmp BaudRate
ByteIn4:
        rcall IDCheck
        rjmp ChipIDf
ByteIn255:
        rcall IDCheck
        rjmp StartSend
ByteIn254:
        rjmp StopSend
```

Pixel Dump Wrapper

The Agilent ADNS series camera chip allows for dump of a series of pixels across the serial two-wire interface. The user may access this information in order to determine focus, surface quality, and contrast. The process of accessing pixel data in the chip requires sequential write/read (100□s delays) operations. Periodically, during a single frame read, bad pixel data may be acquired and a "valid data" bit is cleared in the resulting pixel byte. The user must discard this pixel and request again. Once the system is done, it loops and another frame may be acquired (a start of frame bit indicates this transition).

In order to simplify and expedite this process, I have written a subroutine that automates the acquisition of a single frame of pixel data for the user. This is an important component of system calibration. The subroutine completes the following tasks:

- Request pixels and transmits them over the UART serial data line
- Discard bad pixel data and re-request
- Stop when new Start-of-Frame bit is detected

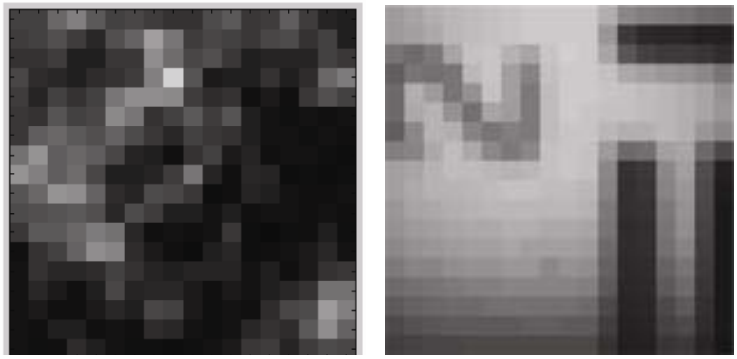

Figure A1.3 Single 18x18 64-bit grayscale images acquired using the pixel dump functionality. At left, the surface of a black painted ping-pong ball. At right, datasheet image of USAF test standard.

```
;*************************
;Pixel Dump to user
Video:
;Pause 100us
ldi temp,2
;Set clock prescaling to 8 cycles
per tick
out TCCR0,temp
ldi timer,145
rcall Pause

mov adns,config
ori adns,0b10000000
;make sure write/read bit is set

rcall WriteADNS     ;Send byte to
chip

;Pause 100us
ldi temp,2
out TCCR0,temp
ldi timer,145
rcall Pause

ldi adns,0x01       ;Always
Awake

rcall WriteADNS
;Send second byte to chip

;100us pause is 139*8 clock cycles
ldi temp,2
out TCCR0,temp
ldi timer,139

;Request pixel dump
mov adns,pixeldump
ori adns,128        ;set write bit
ser temp
out DDRD,temp
rcall WriteADNS

nop
nop
ldi adns,0
rcall WriteADNS     ;write anything

;Pause 100us
ldi temp,2
out TCCR0,temp
ldi timer,145
rcall Pause

ldi dy,128
Pixel:

mov adns,pixeldata
andi adns,127       ;clear read bit
rcall WriteADNS     ;request a pixel
clr temp
out PORTD,temp
out DDRD,temp

;Pause 100us
ldi temp,2
out TCCR0,temp
ldi timer,139
rcall Pause

rcall ReadADNS      ;Read in a
pixel value
ser temp
out DDRD,temp

mov dx,adns
andi dx,0b00111111
;isolate the pixel data and stick it in dx

mov temp,adns
andi temp,128       ;isolate start of
frame bit
eor dy, temp
;Before first loop, dy is set to 0x80
;First pixel of frame will be
0x80(eor)0x80=0x00
;and clear dy.. Subsequent pixels of frame
wil be
;0x00(eor)0x00=0x00 and keep dy clear.
First
```

```
;pixel of next frame will be 0x00(eor)0x80=0x80
;and load 0x80 into dy

ldi temp,128
cp dy,temp
breq pixDrop
;if next frame starts, end capture

mov temp,adns
andi temp,64          ;isolate data valid bit
cpi temp, 0
breq Pixel
;if data is bad, re-request pixel

;Loop until UDR is empty
pixTXWait:
in temp,USR           ;UART Status Bits
ldi mask,0b00100000
and temp,mask         ;Isolate UDR Empty status bit
cpse temp,mask        ;See if UDR is Empty
rjmp pixTXWait

out UDR,dx            ;Send Pixel Value

rjmp Pixel

pixDrop:
sbrc adns,7
rjmp ByteInDrop
rjmp Video
```

Various Subroutines for Protocol Bridging

The remaining subroutines in the code handle the various user functions. Their functionality is straight-forward and they utilize the previously described communication protocol subroutines. These functions handle the following activities:

- Read an arbitrary ADNS chip register to the user
- Write to an arbitrary ADNS register from the user
- Set the Agilent chip UART baud rate (see table A1.3)
- Handle starting and stopping of the data sampling
- Arbitrary delay (utilizing timer/counter0)
- UART Write function which **blocks** until transmit is complete

Table A1.3 – Byte Commands and Associated UART Baud Rates

Baud Rate	Byte	Time to Broadcast
4800	143	417.5 □s
9600	71	209.1 □s
14400	47	139.7 □s
19200	35	105.0 □s
28800	23	70.2 □s
38400	17	52.9 □s
57600	11	35.5 □s
76800	8	26.8 □s
115200	5	18.2 □s

```
;*****************************
*
;Read from the ADNS camera.
Send byte to user
ReadArbitrary:
ser temp
out DDRD,temp
;Wait for second byte to arrive
from the user
in temp, USR
andi temp,128
;Recieve complete bit is bit 7 in the
USR
ldi mask,128
cpse temp,mask
rjmp ReadArbitrary

in serial,UDR
andi serial,0b01111111      ;clear
write/read bit
mov adns,serial
rcall WriteADNS  ;Target Write
Register Byte

clr temp
out PORTD,temp
out DDRD,temp

;Pause 100us
ldi temp,2
out TCCR0,temp
ldi timer,139
rcall Pause

rcall ReadADNS  ;read byte from
the chip
ser temp
out DDRD,temp

;Send the byte to the user
mov serial,adns
rcall UARTOut
rjmp ByteInDrop
```

```
;*********************************
*
;Change the UART Baud Rate
BaudRate:
;Wait for second byte to arrive from the
user
in temp, USR
andi temp,128
;Recieve complete bit is bit 7 in the USR
ldi mask,128
cpse temp,mask
rjmp BaudRate

in serial, UDR
out UBRR,serial
rjmp ByteInDrop

;*****************************
;Read in 2 serial bytes and send them
;to the ADNS camera chip
SendArbitrary:

ser temp
out DDRD,temp
;Wait for second byte to arrive from the
user
in temp, USR
andi temp,128
ldi mask,128
cpse temp,mask
rjmp SendArbitrary

in serial, UDR
ori serial,0b10000000
;make sure write/read bit is set
mov dx,serial
;ADDRESS Byte is stored in dx

Send2Wait:
;Wait for third byte to arrive from user
in temp, USR
andi temp,128
ldi mask,128
cpse temp,mask
rjmp Send2Wait
```

```
in serial,UDR
mov dy, serial  ;WRITE VALUE is stored in dy

rcall IDCheck

mov adns,dx
rcall WriteADNS      ;Send byte to chip

;Pause 100us
ldi temp,2
out TCCR0,temp
ldi timer,145
rcall Pause

mov adns,dy

rcall WriteADNS   ;Send second byte to chip
rjmp ByteInDrop
;*****************************
;Toggle Send/Silent state of flyball
StartSend:
mov temp,sgo
ldi sgo,0xff
eor temp,sgo
;if temp is 0xff, initiate acquisition

cpse temp,sgo
rjmp ByteInDrop
;############################
;Subroutine to Pause for an arbitrary period
;of time using t/c0
;timer target is stored in timer
Pause:
ldi temp,0
out TCNT0,temp      ;Clear Clock

TimeLoop:
in temp,TCNT0       ;get current clock value
cp temp,timer
brsh EjectTimer
;if the timer value > the target, drop out
rjmp TimeLoop   ;otherwise, continue looping

EjectTimer:
        ret
;################################
;Subroutine to write a byte out the UART
;Byte to write is stored in serial
;serial is set to 0x40 when complete
UARTOut:

ldi temp,0b01000000
out USR, temp
;Transmit Complete Status Bit Reset to zero

;Make sure the UART is ready for another byte
preTXWait:
in temp,USR                 ;UART Status Bits
ldi mask,0b00100000
and temp,mask       ;Isolate UDR Empty status bit
cpse temp,mask      ;See if UDR is Empty
rjmp preTXWait

out UDR,serial
;Load up the byte for transmission
ldi mask,0b01000000

TXWait:
in temp,USR                 ;UART Status Bits
andi temp,0b01000000
;Isolate TX Complete status bit
cpse temp,mask      ;See if TX is complete
rjmp TXWait

ret
```

APPENDIX 2

gPRIME CODE HIGHLIGHTS

Introduction

This appendix contains specific segments of the PRIME MATLAB code. Several segments were chosen in order to demonstrate how the interface and data acquisition actions are carried out. Most of the code is designed using similar techniques as those illustrated here.

Included Code Segments

1. gprime
    - Main function which handles all internal object calls by utilizing a "switch yard" technique.
2. MakeGUI
    - Construct the initial graphical user interface for the software oscilloscope
3. ChannelAdd
    - Handle addition of a channel to the scope interface
4. MoveThresh
    - My solution for dragging line objects around an axis in Matlab
5. ChangeTrig
    - Restart the analog input system after resetting the trigger state
6. GoStartStop
    - Handle initiating and ending data logging
7. StimParam
    - Check for parameter errors and construct the output stimulus waveform

8. SAF
    - The “Samples Acquired Function” interrupt that is called to update a sweep width of data
9. SAF Visualization
    - Handle the creation of the single sweep FFT or the Spectrogram
10. ConvertTXT
    - One of many data conversion tools, this one converts a native file to *.txt
11. aConnect
    - Link the analysis interface to the active scope or to a file
12. aFiltCalc
    - Construct a 3rd order Butterworth filter
13. aChangeAnalysisDisplay
    - Update the analysis window with new parameters corresponding to detected events. Also, update the Y-Axis histogram
14. aRTCalc
    - All analysis metrics are calculated in real time while data streams in from the scope SAF function. Also handles real time correlation if selected.
15. aSetPolyRegion
    - Drawing a polygonal region in an analysis space
16. aDrawSubEvents
    - Extract the enclosed samples from the polygon and display the corresponding events in a separate window
17. aSubSingleShow
    - Visually grooming a subset of events
18. LoadSubEvents
    - Offline correlation of extracted points with a target

The following text contains commented code from the g-PRIME MATLAB function file, a separate summary description, and images designed to illustrate each function's behavior in the program.

1. gprime

This function contains the opening "switch yard" that handles branching within the MATLAB function file. Internal subfunctions can not be called by name from interface elements that the user interacts with. Subfunctions may be accessed through the use of "function handles" or in the manner illustrated here.

If the function is called with no arguments (nargin=0), the program initializes and constructs the graphical user interface. If the program contains one or more arguments, the first argument is a string that is the name of a subfunction. If there are two input arguments, the second argument is treated as an input parameter to the subfunction.

A "To-Do" list of desired future functionality is also contained here and is where I keep a list of known bugs. As of this incarnation of the program, this list contains only desired functionality. These may be included in future versions of the program.

```
function gprime(varargin)
% gPRIME (G')
% Physiology Recordings & Identification of Multiple Events
%
% Written in Spring 2007 by
% Gus Lott (GKL6@cornell.edu)
%
% Copywrite Gus Lott 2007, free distribution allowed for education and
% university research purposes as long as original authorship is referenced when cited.
% Industrial use prohibited.
```

```
%Entry Switchyard
switch nargin
   case 0
      daqreset
      delete(findobj('tag','gSS07'))
      delete(findobj('tag','gSS07splash'))
      delete(findobj('tag','gSS07anal'));
      makegui
   case 1
      feval(varargin{1})
   case 2
      feval(varargin{1},varargin{2})
end
```

2. makegui

This is the main initialization function which handles the construction of the main software oscilloscope figure. The figure consists of several "uicontrol" elements arranged as children of 6 "uipanel" objects.

The program initiates with a splash screen which is displayed while the program registers interface drivers and scans the computer for installed/supported hardware. Once available interfaces have been detected, the main figure window launches and the scope is constructed with all UI elements in "off" enable mode (greyed out, **fig. A2.1**).

Error handling in user input edit boxes is carried out by storing the numeric value in the element's "userdata" property. When the value in a text box is changed, the value is converted from text into a number. If the result is not a number (i.e. bad text value) or outside of a specified range of allowed values, the "userdata" is extracted and inserted into the text object. If the value is a valid number in the allowed range, the userdata is updated with the new value. An example callback string that would carry out this functionality is as follows.

```
['if isnan(str2double(get(gcbo,"string")));',...
'set(gcbo,"string",num2str(get(gcbo,"userdata"))); else;',...
'set(gcbo,"userdata",str2double(get(gcbo,"string"))); end;']
```

Exclusivity of radio buttons in groups is also a common task that needs to be carried out in the interface. In order to accomplish this, all of the handles pointing to the radio button group members are stored in each of the members' "userdata" properties. When a radio button is pressed, the userdata is extracted and all of the radio buttons are set to off and then the one that was pressed is set to on. An example of a callback string that accomplishes this task follows.

```
['set(get(gcbo,"userdata"),"value",0);',...
'set(gcbo,"value",1); gprime("ChangeTrig");]
```

All pointers to objects (handles) in the user interface are stored in a data struct that is placed in the main figure's "userdata" property for access later in subfunctions (the pointers in this function are local variables and not readily available to functions called at a later time). The main figure is given a unique "tag" property so that it can be quickly referenced in subfunctions. This struct acts as the anchor that allows for easy referencing of all interface elements and many state variables. The following code is used, frequently, to extract the pointers in subfunctions.

```
gui=get(findobj('tag','gSS07'),'userdata');
```

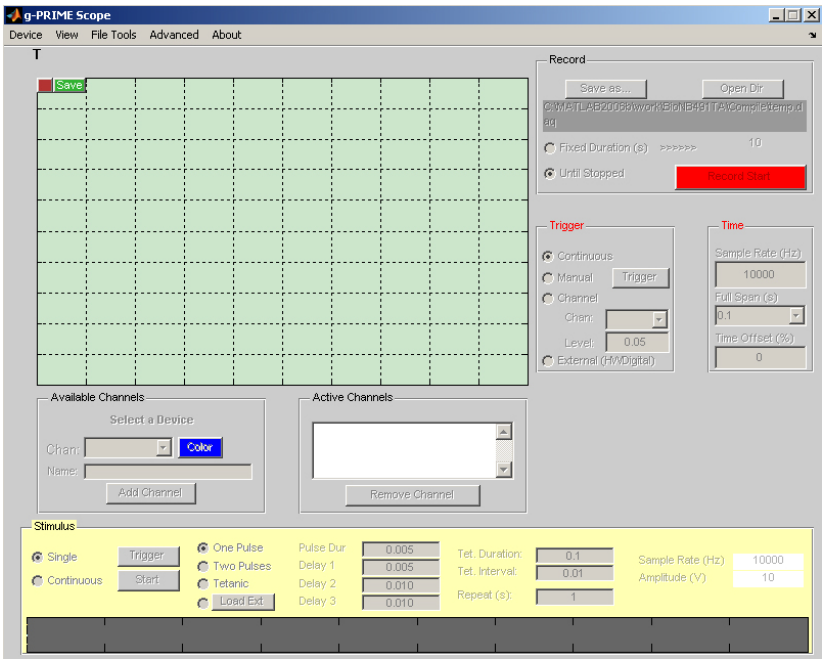

Figure A2.1 – The unconnected initial interface constructed by "makegui."

```
% Construct initial Graphic Interface
function makegui

if isdeployed
   %Register Interfaces on Stand-alone systems
   interfaces={'advantech','hpe1432','keithley','mcc','nidaq',...
      'parallel','winsound'};
   for i=1:length(interfaces)
      try; daqregister(interfaces{i});    end
   end
end
warning off MATLAB:Axes:NegativeDataInLogAxis
%Initialize some Variables
gui.ax=[]; gui.pl=[]; gui.tx=[];
gui.aLink(1:4)=0; gui.ChanControls=[]; gui.OverFlag=0;


%splash Screen
tempFig=figure('position',[0 0 400 200],'menubar','none','numbertitle','off','name',...
   ['g',char(180)],'tag','gSS07splash','resize','off');
centerfig(tempFig)
axes('position',[0 0 1 1],'color',[.8 .9 .8],'yticklabel',[]...
   ,'xticklabel',[],'xlim',[0 1],'ylim',[0 1],'xcolor',[.4 .4 .4],'ycolor',[.4 .4 .4]);
box on
```

```
text(0.5,0.7,'g-PRIME','fontsize',40,'fontweight','bold',...
   'horizontalalignment','center','fontname','Comic Sans MS')
text(.75,.35,'Gus Lott ','fontsize',15,'fontweight','bold','horizontalalignment','left')
text(.75,.25,'Cornell University','fontsize',15,'fontweight','bold',...
   'horizontalalignment','center','color',[.7 0 0])
a=text(.01,.1,'Scanning for Interfaces','fontsize',12,'fontweight','bold',...
   'horizontalalignment','left');
pause(0.2); set(a,'string','Scanning for Interfaces.')
pause(0.2); set(a,'string','Scanning for Interfaces..')
pause(0.2); set(a,'string','Scanning for Interfaces...')

% Scan system for installed interfaces while splash screen is active
drawnow; gTemp=daqhwinfo; set(a,'string','Scanning for Interfaces....')
pause(0.2); delete(tempFig)

%Construct Main Figure
gui.fig=figure('name','g-PRIME Scope','numbertitle','off','tag','gSS07',...
   'menubar','none','position',[0 0 800 600],'resize','off','deletefcn',...
   'gprime("DelFcn")','doublebuffer','on');
centerfig(gui.fig)
set(gui.fig,'windowbuttonupfcn',...
   'set(gcf,"windowbuttonmotionfcn",""); gprime("ChangeLevel");')

%Construct Axes for Display
gui.axtxt=uicontrol('style','text','units','normalized','position',...
   [0.2 0.96 0.3 0.03],'string','','fontunits','normalized','fontsize',1,...
   'backgroundcolor',get(gcf,'color'),'visible','off');
gui.backax=axes('position',[0.04 0.45 0.6 0.5],'xgrid','on','ygrid','on','color',...
   [.8 .9 .8],'yticklabel',[],'xticklabel',[],'XAxisLocation','top','ylim',[-1 1],...
   'box','on');
gui.timetext=text(0,1.15,'T','buttondownfcn','gprime("DragTime")',...
   'horizontalalignment','center','fontweight','bold');

gui.MainFigCapture=uicontrol('style','pushbutton','units','normalized',...
   'position',[0.04 0.925 0.02 0.025],'backgroundcolor',[.7 .2 .2],...
   'callback','gprime("CaptureScope")');
gui.MainFigSave=uicontrol('style','pushbutton','units','normalized',...
   'position',[0.06 0.925 0.04 0.025],'backgroundcolor',[.2 .7 .2],'string','Save',...
   'foregroundcolor','w','callback','gprime("CaptureScopeRaw")');

% Visualization Graphs.  Image for Spectrogram and Patches for FFT
gui.visAx=axes('position',[0.04 0.45 0.6 0.15],'color',[.8 .9 .8],'visible','off',...
   'color','none');
box on
Spect=zeros(100,100,3);
```

```
gui.SpectVisImage=image(Spect,'parent',gui.visAx,'visible','off','userdata',0);
NVisFFT=100;
for i=1:NVisFFT
   gui.fftVisPatch(i)=patch([-.5 .5 .5 -.5]+i,[0 0 1 1],'r','parent',gui.visAx,...
         'edgecolor','none','visible','off');
end
set(gui.visAx,'visible','off','ydir','normal','ylim',[0 1],'ytick',[],...
   'xtick',[],'xaxislocation','top','userdata',0,'tag','gfftScaleAx')
set([gui.visAx, gui.fftVisPatch, gui.SpectVisImage],'buttondownfcn',...
   ['set(findobj("tag","gfftScaleAx"),"userdata",',...
   '~get(findobj("tag","gfftScaleAx"),"userdata"))'])


%Frame for Channel Activation Controls
gui.ChannelAddPanel=uipanel('title','Available Channels','units','normalized',...
   'position',[0.04 0.24 0.28 0.2],'backgroundcolor',get(gcf,'color'));
gui.ChanDeviceName=uicontrol('parent',gui.ChannelAddPanel,'style','text','string',...
   'Select a Device',...
   'units','normalized','backgroundcolor',get(gcf,'color'),'position',...
   [0.05 0.75 0.9 0.17],'fontweight','bold');
gui.AvailableList=uicontrol('parent',gui.ChannelAddPanel,'style','popupmenu','units',...
   'normalized','backgroundcolor','w','position',[0.2 0.5 0.4 0.2],'string',' ');
uicontrol('parent',gui.ChannelAddPanel,'style','text','string','Chan:','units',...
   'normalized','backgroundcolor',get(gcf,'color'),'position',[0.01 0.54 0.19 0.12],...
   'fontunits','normalized','fontsize',1);
gui.ChanColorSelect=uicontrol('parent',gui.ChannelAddPanel,'style','pushbutton',...
   'units','normalized','backgroundcolor','b','position',[0.62 0.5 0.2 0.2],...
   'string','Color','callback','gprime("tracecolor")','foregroundcolor','w');
uicontrol('parent',gui.ChannelAddPanel,'style','text','string','Name:','units',...
   'normalized','backgroundcolor',get(gcf,'color'),'position',[0.01 0.34 0.19 0.10],...
   'fontunits','normalized','fontsize',1);
gui.ChanNameSet=uicontrol('parent',gui.ChannelAddPanel,'style','edit',...
   'backgroundcolor','w','units','normalized','position',[0.2 0.3 0.75 0.15],...
   'horizontalalignment','left');
gui.ChanAdd=uicontrol('parent',gui.ChannelAddPanel,'style','pushbutton','units',...
   'normalized','backgroundcolor',get(gcf,'color'),'position',[0.3 0.07 0.4 0.2],...
   'string','Add Channel','callback','gprime("ChannelAdd")');

%Frame for Channel Select Controls
gui.ChannelListPanel=uipanel('title','Active Channels','units','normalized',...
   'position',[0.36 0.24 0.28 0.2],'backgroundcolor',get(gcf,'color'));
gui.ChanList=uicontrol('parent',gui.ChannelListPanel,'style','listbox','units',...
   'normalized','position',[0.05 0.3 0.9 0.55],'backgroundcolor','w','callback',...
   'gprime("DisplayChanControls")');
gui.RemoveChan=uicontrol('parent',gui.ChannelListPanel,'style','pushbutton','units',...
```

```
    'normalized','position',[0.2 0.05 0.6 0.2],'string','Remove Channel',...
    'backgroundcolor',get(gcf,'color'),'callback','gprime("ChannelRemove")');

%Frame for Recording Controls
gui.RecControls=uipanel('title','Record','units','normalized','position',...
    [0.65 0.76 0.34 0.23],'backgroundcolor',get(gcf,'color'));
gui.RecBrowse=uicontrol('parent',gui.RecControls,'style','pushbutton','units',...
    'normalized','position',[0.1 0.75 0.3 0.15],'backgroundcolor',get(gcf,'color'),...
    'string','Save as...','callback','gprime("recbrowse")');
gui.RecDir=uicontrol('parent',gui.RecControls,'style','pushbutton','units',...
    'normalized','position',[0.6 0.75 0.3 0.15],'backgroundcolor',get(gcf,'color'),...
    'string','Open Dir','tag',pwd,'callback','gprime("openpwd");');
gui.RecFile=uicontrol('parent',gui.RecControls,'style','text','units','normalized',...
    'position',[0.02 0.48 0.96 0.26],'backgroundcolor',[0.6 0.6 0.6],...
    'string',[pwd,'\temp.daq'],'horizontalalignment','left');
gui.Rec(1)=uicontrol('parent',gui.RecControls,'style','radio','units','normalized',...
    'position',[0.02 0.25 0.96 0.2],'backgroundcolor',get(gcf,'color'),...
    'string','Fixed Duration (s)    >>>>>>');
    gui.RecDuration=uicontrol('parent',gui.RecControls,'style','text','units',...
        'normalized','position',[0.6 0.25 0.38 0.2],'backgroundcolor',...
        get(gui.RecControls,'backgroundcolor'),'foregroundcolor',...
    get(gui.RecControls,'backgroundcolor'),'string','10','userdata',1,'tag','gRecDur',...
        'callback',['if isnan(str2double(get(gcbo,"string"))); ',...
        'set(gcbo,"string",num2str(get(gcbo,"userdata"))); else;',...
        'set(gcbo,"userdata",str2double(get(gcbo,"string"))); end;'],...
        'userdata',0);
gui.Rec(2)=uicontrol('parent',gui.RecControls,'style','radio','units','normalized',...
    'position',[0.02 0.05 0.4 0.2],'backgroundcolor',get(gcf,'color'),...
    'string','Until Stopped','value',1);
set(gui.Rec(2),'userdata',gui.Rec,'callback',...
    ['set(get(gcbo,"userdata"),"value",0); set(gcbo,"value",1);',...
    ' set(findobj("tag","gRecDur"),"style","text","backgroundcolor",',...
    'get(get(gcbo,"parent"),"backgroundcolor"),"foregroundcolor",'...
    'get(get(gcbo,"parent"),"backgroundcolor"));'])
set(gui.Rec(1),'userdata',gui.Rec,'callback',...
    ['set(get(gcbo,"userdata"),"value",0); set(gcbo,"value",1);',...
    ' set(findobj("tag","gRecDur"),"style","edit","backgroundcolor",',...
    '"w","foregroundcolor","k");'])
gui.RecStartStop=uicontrol('parent',gui.RecControls,'style','toggle','units',...
    'normalized','position',[0.5 0.02 0.48 0.2],'backgroundcolor','r','string',...
    'Record Start','callback','gprime("GoStartStop")');

%Frame for Trigger Controls
gui.TrigControls=uipanel('title','Trigger','units','normalized','position',...
    [0.65 0.47 0.17 0.25],'backgroundcolor',get(gcf,'color'),'foregroundcolor','r');
```

```
gui.Trig(1)=uicontrol('parent',gui.TrigControls,'style','radio','units','normalized',...
   'position',[0.02 0.75 .96 0.15],'string','Continuous','backgroundcolor',...
   get(gui.TrigControls,'backgroundcolor'),'value',1,'callback',...
   'gprime("ChangeTrig")');
gui.Trig(2)=uicontrol('parent',gui.TrigControls,'style','radio','units','normalized',...
   'position',[0.02 0.6 .96 0.15],'string','Manual','backgroundcolor',...
   get(gui.TrigControls,'backgroundcolor'),'callback','gprime("ChangeTrig")');
   gui.TrigMan=uicontrol('parent',gui.TrigControls,'style','Pushbutton','units',...
      'normalized','position',[0.55 0.6 0.43 0.15],'string','Trigger',...
      'backgroundcolor',get(gui.TrigControls,'backgroundcolor'),'callback',...
      'trigger(get(gcbo,"userdata"))','enable','off');
gui.Trig(3)=uicontrol('parent',gui.TrigControls,'style','radio','units','normalized',...
   'position',[0.02 0.45 .96 0.15],'string','Channel','backgroundcolor',...
   get(gui.TrigControls,'backgroundcolor'),'callback','gprime("ChangeTrig")');
   uicontrol('parent',gui.TrigControls,'style','text','units','normalized',...
      'position',[0.2 0.33 0.3 0.10],'string','Chan:','backgroundcolor',...
      get(gcf,'color'),'horizontalalignment','left');
   gui.TrigChan=uicontrol('parent',gui.TrigControls,'style','popupmenu','units',...
      'normalized','position',[0.5 0.3 0.48 0.15],'string',' ','backgroundcolor',...
      'w','enable','off');
   uicontrol('parent',gui.TrigControls,'style','text','units','normalized',...
      'position',[0.2 0.15 0.3 0.10],'string','Level:','backgroundcolor',...
      get(gcf,'color'),'horizontalalignment','left');
   gui.TrigLevel=uicontrol('parent',gui.TrigControls,'style','edit','units',...
      'normalized','position',[0.5 0.13 0.48 0.15],'string','0.05 ',...
      'backgroundcolor','w','enable','off','userdata',0.05);
gui.Trig(4)=uicontrol('parent',gui.TrigControls,'style','radio','units','normalized',...
   'position',[0.02 0.01 .96 0.12],'string','External (HWDigital)',...
   'backgroundcolor',get(gui.TrigControls,'backgroundcolor'));
set(gui.Trig,'userdata',gui.Trig,...
   'callback',['set(get(gcbo,"userdata"),"value",0);',...
   'set(gcbo,"value",1); gprime("ChangeTrig");'])

%Frame for Time Controls
gui.TimeControls=uipanel('title','Time','units','normalized','position',...
   [0.86 0.47 0.13 0.25],'backgroundcolor',get(gcf,'color'),'foregroundcolor','r');
uicontrol('parent',gui.TimeControls,'style','text','string','Sample Rate (Hz)',...
   'backgroundcolor',get(gcf,'color'),'units','normalized','position',...
   [0.05 0.8 .9 0.1],'horizontalalignment','left');
gui.SRate=uicontrol('parent',gui.TimeControls,'style','Edit','string','10000',...
   'backgroundcolor','w','units','normalized','position',[0.05 0.6 .9 0.2],...
   'userdata',10000,'callback',...
   ['if isnan(str2double(get(gcbo,"string")));',...
   'set(gcbo,"string",num2str(get(gcbo,"userdata"))); else;',...
   'set(gcbo,"userdata",str2double(get(gcbo,"string"))); end;',...
```

```
    'gprime("initAI")']);
uicontrol('parent',gui.TimeControls,'style','text','string','Full Span (s)',...
    'backgroundcolor',get(gcf,'color'),'units','normalized','position',...
    [0.05 0.48 .9 0.1],'horizontalalignment','left');
gui.Refresh=uicontrol('parent',gui.TimeControls,'style','popupmenu','string',...
    strvcat('0.05','0.1','0.2','0.4','0.6','0.8','1','2','5','10','20','30'),...
    'backgroundcolor','w','units','normalized','position',[0.05 0.28 .9 0.2],...
    'userdata',0.2,'value',2,'callback','gprime("initAI")');
gui.UpdateRate=uicontrol('parent',gui.TimeControls,'style','text','string','0.1',...
    'backgroundcolor',get(gcf,'color'),'foregroundcolor',get(gcf,'color'),'units',...
    'normalized','position',[0.05 0.01 .9 0.17],'userdata',0.1);
uicontrol('parent',gui.TimeControls,'style','text','string','Time Offset (%)',...
    'backgroundcolor',get(gcf,'color'),'units','normalized','position',...
    [0.05 0.18 .9 0.1],'horizontalalignment','left');
gui.TriggerDelay=uicontrol('parent',gui.TimeControls,'style','edit','string','0',...
    'backgroundcolor','w','units','normalized','position',[0.05 0.01 .9 0.17],...
    'userdata',0,'callback',['if isnan(str2double(get(gcbo,"string")));',...
    'set(gcbo,"string",num2str(get(gcbo,"userdata"))); else;',...
    'set(gcbo,"userdata",str2double(get(gcbo,"string"))); end;',...
    'gprime("ChangeTrig"); ']);

%Frame For Stimulation Control ---------------------------------
gui.StimControls=uipanel('title','Stimulus','units','normalized','position',...
    [0.02 0.01 0.97 0.22],'backgroundcolor',[1 1 .7]);

%Mode Controls (Single/Continuous)
gui.StimMode(1)=uicontrol('parent',gui.StimControls,'style','radio','units',...
    'normalized','position',[0.01 0.7 0.11 0.2],'string','Single','backgroundcolor',...
    get(gui.StimControls,'backgroundcolor'),'value',1);
gui.StimModeTrig=uicontrol('parent',gui.StimControls,'style','pushbutton','units',...
    'normalized','position',[0.12 0.72 0.08 0.18],'string','Trigger',...
    'backgroundcolor',get(gui.fig,'color'),'tag','gSS07ao',...
    'callback','gprime("StimLoad"); start(get(gcbo,"userdata"))');
gui.StimMode(2)=uicontrol('parent',gui.StimControls,'style','radio','units',...
    'normalized','position',[0.01 0.5 0.11 0.2],'string','Continuous',...
    'backgroundcolor',get(gui.StimControls,'backgroundcolor'));
gui.StimModeStart=uicontrol('parent',gui.StimControls,'style','toggle','units',...
    'normalized','position',[0.12 0.52 0.08 0.18],'string','Start','backgroundcolor',...
    get(gui.fig,'color'),'enable','off','callback',['gprime("StimLoad");',...
    'if get(gcbo,"value")==1; start(get(gcbo,"userdata")); ',...
    'set(gcbo,"string","Stop"); else; stop(get(gcbo,"userdata")); ',...
    'set(gcbo,"string","Start"); end']);
set(gui.StimMode,'userdata',gui.StimMode,'callback',...
    ['set(get(gcbo,"userdata"),"value",0); ',...
    'set(gcbo,"value",1); gprime("StimParam")'])
```

```
%Waveform Shape (pulse duration,3x delays)
gui.StimType(1)=uicontrol('parent',gui.StimControls,'style','radio','units',...
   'normalized','position',[0.22 0.8 0.10 0.15],'string','One Pulse',...
   'backgroundcolor',get(gui.StimControls,'backgroundcolor'),'value',1);
gui.StimType(2)=uicontrol('parent',gui.StimControls,'style','radio','units',...
   'normalized','position',[0.22 0.65 0.10 0.15],'string','Two Pulses',...
   'backgroundcolor',get(gui.StimControls,'backgroundcolor'));
gui.StimType(3)=uicontrol('parent',gui.StimControls,'style','radio','units',...
   'normalized','position',[0.22 0.5 0.10 0.15],'string','Tetanic',...
   'backgroundcolor',get(gui.StimControls,'backgroundcolor'));
gui.StimType(4)=uicontrol('parent',gui.StimControls,'style','radio','units',...
   'normalized','position',[0.22 0.34 0.10 0.15],'string','',...
   'backgroundcolor',get(gui.StimControls,'backgroundcolor'));
gui.StimTypeLoad=uicontrol('parent',gui.StimControls,'style','PushButton','units',...
   'normalized','position',[0.24 0.35 0.08 0.15],'string','Load Ext',...
   'enable','off','callback','gprime("LoadAStim")');
set(gui.StimType,'userdata',gui.StimType,'callback',...
   ['set(get(gcbo,"userdata"),"value",0);',...
   ' set(gcbo,"value",1); gprime("StimParam")'])

uicontrol('parent',gui.StimControls,'style','text','backgroundcolor',...
get(gui.StimControls,'backgroundcolor'),'string','Pulse Dur','units','normalized',...
'position',[0.35 0.8 0.08 0.14],'horizontalalignment','left');
gui.StimShape(1)=uicontrol('parent',gui.StimControls,'style','edit','string','0.005',...
   'userdata',0.005,'units','normalized','position',[0.43 0.8 0.1 0.14],...
   'backgroundcolor',[1 .7 .7],'tag','PulseDur');
uicontrol('parent',gui.StimControls,'style','text','backgroundcolor',...
   get(gui.StimControls,'backgroundcolor'),'string','Delay 1','units','normalized',...
   'position',[0.35 0.65 0.1 0.14],'horizontalalignment','left');
gui.StimShape(2)=uicontrol('parent',gui.StimControls,'style','edit','string','0.005',...
   'userdata',0.005,'units','normalized','position',[0.43 0.65 0.1 0.14],...
   'backgroundcolor',[.7 1 .7],'tag','Delay1');
uicontrol('parent',gui.StimControls,'style','text','backgroundcolor',...
   get(gui.StimControls,'backgroundcolor'),'string','Delay 2','units','normalized',...
   'position',[0.35 0.5 0.1 0.14],'horizontalalignment','left');
gui.StimShape(3)=uicontrol('parent',gui.StimControls,'style','edit','string','0.010',...
   'userdata',0.010,'units','normalized','position',[0.43 0.5 0.1 0.14],...
   'backgroundcolor',[.9 .7 1],'tag','Delay2');
uicontrol('parent',gui.StimControls,'style','text','backgroundcolor',...
   get(gui.StimControls,'backgroundcolor'),'string','Delay 3','units','normalized',...
   'position',[0.35 0.35 0.1 0.14],'horizontalalignment','left');
gui.StimShape(4)=uicontrol('parent',gui.StimControls,'style','edit','string','0.010',...
   'userdata',0.010,'units','normalized','position',[0.43 0.35 0.1 0.14],...
   'backgroundcolor',[.7 1 1],'tag','Delay3');
```

```
set(gui.StimShape,'callback',...
   ['if isnan(str2double(get(gcbo,"string")));',...
   ' set(gcbo,"string",num2str(get(gcbo,"userdata"))); else; ',...
   'set(gcbo,"userdata",str2double(get(gcbo,"string"))); end; ',...
   'gprime("StimParam")'])

%Tetanus duration, Tetanic Interval
uicontrol('parent',gui.StimControls,'style','text','backgroundcolor',...
   get(gui.StimControls,'backgroundcolor'),'string','Tet. Duration:','units',...
   'normalized','position',[0.55 0.75 0.15 0.14],'horizontalalignment','left');
gui.StimTetDur=uicontrol('parent',gui.StimControls,'style','edit','string','0.1',...
   'userdata',0.1,'units','normalized','position',[0.65 0.75 0.1 0.14],...
   'backgroundcolor',[.9 .9 .6],'tag','TetDur');
uicontrol('parent',gui.StimControls,'style','text','backgroundcolor',...
   get(gui.StimControls,'backgroundcolor'),'string','Tet. Interval:','units',...
   'normalized','position',[0.55 0.6 0.15 0.14],'horizontalalignment','left');
gui.StimTetInt=uicontrol('parent',gui.StimControls,'style','edit','string','0.01',...
   'userdata',0.01,'units','normalized','position',[0.65 0.6 0.1 0.14],...
   'backgroundcolor',[1 .9 .4],'tag','TetInt');
uicontrol('parent',gui.StimControls,'style','text','backgroundcolor',...
   get(gui.StimControls,'backgroundcolor'),'string','Repeat (s):','units',...
   'normalized','position',[0.55 0.4 0.15 0.14],'horizontalalignment','left');
gui.StimRepeat=uicontrol('parent',gui.StimControls,'style','edit','string','1',...
   'userdata',1,'units','normalized','position',[.65 .4 .1 .14],'backgroundcolor','w');
set([gui.StimTetDur gui.StimTetInt],'callback',...
   ['if isnan(str2double(get(gcbo,"string")));',...
   ' set(gcbo,"string",num2str(get(gcbo,"userdata"))); else; ',...
   'set(gcbo,"userdata",str2double(get(gcbo,"string"))); end; ',...
   'gprime("StimParam")'])

%Output Controls (sample rate, amplitude) Repeat Interval
uicontrol('parent',gui.StimControls,'style','text','backgroundcolor',...
   get(gui.StimControls,'backgroundcolor'),'string','Sample Rate (Hz)','units',...
   'normalized','position',[0.78 0.7 0.15 0.14],'horizontalalignment','left');
gui.StimSRate=uicontrol('parent',gui.StimControls,'style','text','string','10000',...
   'userdata',2500,'units','normalized','position',[0.9 0.7 0.09 0.14],...
   'backgroundcolor','w','tag','gui.StimSRate');
uicontrol('parent',gui.StimControls,'style','text','backgroundcolor',...
   get(gui.StimControls,'backgroundcolor'),'string','Amplitude (V)','units',...
   'normalized','position',[0.78 0.55 0.15 0.14],'horizontalalignment','left');
gui.StimAmp=uicontrol('parent',gui.StimControls,'style','text','string','10',...
   'userdata',10,'units','normalized','position',[.9 .55 .09 .14],...
   'backgroundcolor','w','tag','gui.StimAmp');
set(gui.StimSRate,'buttondownfcn',...
   ['set(findobj("tag","gui.StimSRate"),"style","edit"); ',...
```

```
    'set(findobj("tag","gui.StimSRate"),"buttondownfcn","")'])
set(gui.StimAmp,'buttondownfcn',...
    ['set(findobj("tag","gui.StimAmp"),"style","edit"); ',...
    'set(findobj("tag","gui.StimAmp"),"buttondownfcn","")'])

set([gui.StimSRate gui.StimAmp gui.StimRepeat],'callback',...
    ['if isnan(str2double(get(gcbo,"string"))); ',...
    'set(gcbo,"string",num2str(get(gcbo,"userdata"))); else; ',...
    'set(gcbo,"userdata",str2double(get(gcbo,"string"))); end;',...
    ' gprime("StimParam")'])

%Graph for waveform
AxLoc=[0.005 0.005 .99 .3];
gui.StimBackAx=axes('parent',gui.StimControls,'color',[.4 .4 .4],'position',AxLoc,...
    'xticklabel',[],'ytick',[]);
box on
gui.StimAx=axes('parent',gui.StimControls,'color','none','position',AxLoc,'xtick',[],...
    'ytick',[],'ylim',[0 11],'xlim',[0 .1]);
gui.StimPl=line(0,5,'color','w','linewidth',2);
gui.StimPlPulseDur=line([0 0],[8.5 8.5],'color',get(gui.StimShape(1),...
    'backgroundcolor'),'linewidth',3);
gui.StimPlDelay1=line([0 0],[5.5 5.5],'color',get(gui.StimShape(2),...
    'backgroundcolor'),'linewidth',3);
gui.StimPlDelay2=line([0 0],[5.5 5.5],'color',get(gui.StimShape(3),...
    'backgroundcolor'),'linewidth',3);
gui.StimPlDelay3=line([0 0],[5.5 5.5],'color',get(gui.StimShape(4),...
    'backgroundcolor'),'linewidth',3);
gui.StimPlTetDur=line([0 0],[2.5 2.5],'color',get(gui.StimTetDur,...
    'backgroundcolor'),'linewidth',3);
gui.StimPlTetInt=line([0 0],[8.5 8.5],'color',get(gui.StimTetInt,...
    'backgroundcolor'),'linewidth',3);

gui.StimViolation=uicontrol('parent',gui.StimControls,'style','text',...
    'backgroundcolor',[.9 .9 .6],'units','normalized','position',AxLoc,...
   'foregroundcolor','r','string','Stimulus Parameter Error','fontunits','normalized',...
    'fontsize',.8,'fontweight','bold','visible','off');

gui.AIControls=[gui.ChannelAddPanel gui.ChannelListPanel gui.RecControls...
    gui.TrigControls gui.TimeControls gui.StimControls];

for i=gui.AIControls
    a=get(i,'children');
    b=findobj(a,'type','axes');
    for j=1:length(b)
        a(a==b(j))=[];
```

```
    end
    set(a,'enable','off')
end

%-----------------------------------------
%Construct menus for interface selection based on available hardware
gui.HWMenu=uimenu('Label','Device','userdata',0);
drawnow
j=1;
for i=1:length(gTemp.InstalledAdaptors)
    try
    gFoo=daqhwinfo(gTemp.InstalledAdaptors{i});
    if ~isempty(gFoo.ObjectConstructorName{1})

gui.HWSubMenu(j)=uimenu('parent',gui.HWMenu,'Label',gTemp.InstalledAdaptors{i
});
        for k=1:length(gFoo.InstalledBoardIds)
            uimenu('parent',gui.HWSubMenu(j),'Label',gFoo.BoardNames{k},'userdata',...
                gFoo.InstalledBoardIds{k},'callback','gprime("BoardSelect")')
        end
        j=j+1;
    end
    end

end

%Construct other Dropdown menus
gui.View(1)=uimenu('Label','View');
gui.View(2)=uimenu('parent',gui.View(1),'Label',...
    'Scope Visualization','separator','on');
    gui.VisMenu(1)=uimenu('parent',gui.View(2),'Label','None','checked','on');
    gui.VisMenu(2)=uimenu('parent',gui.View(2),'Label','FFT');
    gui.VisMenu(3)=uimenu('parent',gui.View(2),'Label','Spectrogram');
    set(gui.VisMenu,'userdata',gui.VisMenu,'callback',...
      ['set(get(gcbo,"userdata"),"checked","off"); set(gcbo,"checked",',...
      '"on"); gprime("ShowVisNow");']);

gui.View(end+1)=uimenu('parent',gui.View(1),'Label','Measure off Active Trace',...
    'callback','gprime("MeasureGo")','separator','on');

gui.DataMenu(1)=uimenu('Label','File Tools');
gui.DataMenu(end+1)=uimenu('parent',gui.DataMenu(1),'Label','Display .daq File',...
    'callback','gprime("LoadDaq")');
gui.DataMenu(end+1)=uimenu('parent',gui.DataMenu(1),'Label','Convert .daq to
.mat',...
```

```
   'callback','gprime("ConvertMAT")','separator','on');
gui.DataMenu(end+1)=uimenu('parent',gui.DataMenu(1),'Label','Convert .daq to
.txt',...
   'callback','gprime("ConvertTXT")');
gui.DataMenu(end+1)=uimenu('parent',gui.DataMenu(1),'Label',...
   'Convert .daq to .wav audio','callback','gprime("ConvertWAV")');

gui.AdvancedMenu(1)=uimenu('Label','Advanced');
gui.AdvancedMenu(2)=uimenu('parent',gui.AdvancedMenu(1),'Label',...
   'Data Analysis Mode','callback','gprime("TimeStamp")');
gui.AdvancedMenu(3)=uimenu('parent',gui.AdvancedMenu(1),'Label',...
   'Control Stimulus Sample Rate','separator','on','callback',...
   'set(get(gcbo,"userdata"),"style","edit")','userdata',gui.StimSRate);
gui.AdvancedMenu(4)=uimenu('parent',gui.AdvancedMenu(1),'Label',...
   'Control Stimulus Amplitude','callback',...
   'set(get(gcbo,"userdata"),"style","edit")','userdata',gui.StimAmp);
gui.AdvancedMenu(5)=uimenu('parent',gui.AdvancedMenu(1),'Label',...
   'Input Range (Gain) Control','callback','','separator','on');

gui.AboutMenu(1)=uimenu('Label','About');
gui.AboutMenu(end+1)=uimenu('parent',gui.AboutMenu(1),'Label',...
   'About g-PRIME','callback','gprime("Aboutgprime")');

set(gui.fig,'userdata',gui)
set(gui.DataMenu(1),'userdata',[pwd,'\'])
```

3. ChannelAdd

The “ChannelAdd” function is activated when the user wishes to connect a hardware channel of streaming data to the scope. This function creates a “uipanel” object containing the controls for an individual hardware channel, creates the analog input channel object, handles the contents of the available channel and active channel lists, and initializes data acquisition on the new channel (**fig A2.2**).

The user may select a color for the trace and this color is used as the background color for the “uipanel” object, the trace containing the channel data, and the axis containing the trace. Text color for the uipanel background is set to white or

black depending on how close the lightness or darkness of the selected color is to white or black in order to prevent some colors whipping out the interface elments from view.

State variables “gui.ax, gui.pl, and gui.ChanControls” are used to track the number of active channels throughout the code. Whenever the user selects a channel from the “Active Channels” list, the input control options for that channel are displayed in the channel controls area (**fig A2.2** right).

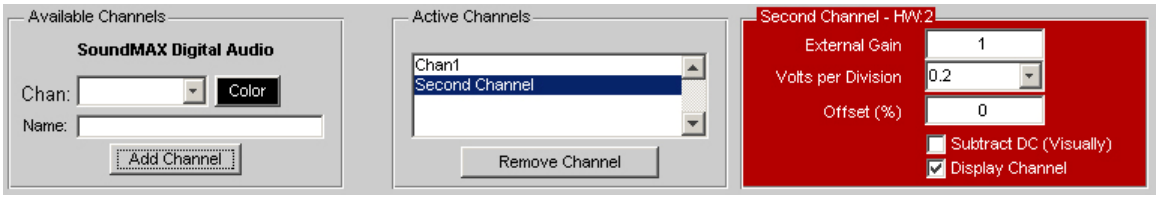

Figure A2.2 – The components involved in the ChannelAdd Interface. Available Channel List, Active Channel List, and the control interface for the selected channel are illustrated.

```
%_____________________________________________
%Create a New Channel for the AI
function ChannelAdd
gui=get(gcbf,'userdata');
gTemp=get(gui.AvailableList,'string');

%Gather Channel Information and Create it
ChanID=str2double(popupstr(gui.AvailableList));
if isnan(ChanID); return; end
stop(gui.ai)

%If channel name is empty, set default channel name
ChanName=get(gui.ChanNameSet,'string');
if isempty(ChanName); ChanName=['Chan',popupstr(gui.AvailableList)]; end

ChanColor=get(gui.ChanColorSelect,'backgroundcolor');

%Create Channel
gTempChan=addchannel(gui.ai,ChanID,ChanName);
%Add channel name to Active Channel List
if isempty(get(gui.ChanList,'string'))
```

```
    set(gui.ChanList,'string',ChanName)
else
    set(gui.ChanList,'string',strvcat(get(gui.ChanList,'string'),ChanName))
end

gTemp(get(gui.AvailableList,'value'),:)=[];
if isempty(gTemp); gTemp=' '; end
set(gui.AvailableList,'string',gTemp)

%Create Graph Elements
try; set(gui.ax,'visible','off'); end
gui.ax=[gui.ax,axes('position',get(gui.backax,'position'),'xgrid','off','ygrid','off',...
    'color','none','xtick',[],'tag',ChanName)];
set(gui.ax(end),'visible','on','ycolor',ChanColor,'userdata',0)
gui.pl=[gui.pl,line([0 1],[0 0],'color',ChanColor)];
gui.tx=[gui.tx,text(0,0,ChanName)];
set(gui.tx(end),'buttondownfcn',['gprime("DragTrace",'',ChanName,'')'],...
    'fontweight','bold','tag',ChanName)

%Create Control Elements
gui.ChanControls=[gui.ChanControls,...
    uipanel('title',[ChanName,' - HW:',num2str(ChanID)],...
    'units','normalized','position',[0.65 0.24 0.34 0.2],...
    'backgroundcolor',get(gcf,'color'),'tag',ChanName,'visible','off')];

%Display latest created channel controls
for i=1:length(gui.ChanControls); set(gui.ChanControls(i),'visible','off'); end
set(gui.ChanControls(end),'visible','on');
set(gui.ChanList,'value',length(gui.ChanControls));
set(gui.ChanControls(end),'backgroundcolor',ChanColor)

%Intelligent font color selection
if mean(get(gui.ChanControls(end),'backgroundcolor'))<0.5
        set(gui.ChanControls(end),'foregroundcolor','w');
else
        set(gui.ChanControls(end),'foregroundcolor','k');
end

gTemp=daqhwinfo(gui.ai);
Chan.ID=ChanID;
set(gui.ChanNameSet,'string','');

uicontrol('parent',gui.ChanControls(end),'string','External Gain','style','text',...
    'backgroundcolor',get(gui.ChanControls(end),'backgroundcolor'),'units',...
    'normalized','position',[0.01 0.8 0.38 0.15],'HorizontalAlignment','right')
```

```
Chan.ExternalGain=uicontrol('parent',gui.ChanControls(end),'style','edit',...
   'string','1','backgroundcolor','w','units','normalized','position',[.45 .8 .3 .2],...
   'userdata',1,'callback',['if isnan(str2double(get(gcbo,"string")));',...
   ' set(gcbo,"string",num2str(get(gcbo,"userdata"))); else; ',...
   'set(gcbo,"userdata",str2double(get(gcbo,"string"))); end;',...
   ' if str2num(get(gcbo,"string"))<1;',...
   ' set(gcbo,"string","1","userdata",1); end;']);
uicontrol('parent',gui.ChanControls(end),'string','Volts per Division','style','text',...
   'backgroundcolor',get(gui.ChanControls(end),'backgroundcolor'),'units',...
   'normalized','position',[0.01 0.6 0.38 0.15],'HorizontalAlignment','right')
Chan.VpD=uicontrol('parent',gui.ChanControls(end),'style','popupmenu','string',...
 strvcat('5','2','1','0.5','0.2','0.1','0.05','0.02','0.01','0.005','0.001','0.0001'),...
   'backgroundcolor','w','units','normalized','position',[.45 .6 .3 .2],'tag','VpD',...
   'userdata',1,'value',5);
uicontrol('parent',gui.ChanControls(end),'string','Offset (%)','style','text',...
   'backgroundcolor',get(gui.ChanControls(end),'backgroundcolor'),'units',...
   'normalized','position',[0.01 0.38 0.38 0.15],'HorizontalAlignment','right')
Chan.Offset=uicontrol('parent',gui.ChanControls(end),'style','edit','string','0',...
   'backgroundcolor','w','units','normalized','position',[.45 .38 .3 .2],...
   'userdata',0,'callback',['if isnan(str2double(get(gcbo,"string")));',...
   ' set(gcbo,"string",num2str(get(gcbo,"userdata"))); else; ',...
   'set(gcbo,"userdata",str2double(get(gcbo,"string"))); ',...
   'end; if str2num(get(gcbo,"string"))>100; set(gcbo,"string","100"); end;',...
   ' if str2num(get(gcbo,"string"))<-100; set(gcbo,"string","-100"); end;']);

Chan.Show=uicontrol('parent',gui.ChanControls(end),'style','Checkbox','string',...
   'Display
Channel','backgroundcolor',get(gui.ChanControls(end),'backgroundcolor'),...
   'units','normalized','position',[0.45 0.04 .5 0.14],'value',1,'userdata',...
   gui.pl(end),'callback','gprime("HidePlots")');
Chan.ACCouple=uicontrol('parent',gui.ChanControls(end),'style','Checkbox','string',...
   'Subtract DC (Visually)','backgroundcolor',...
   get(gui.ChanControls(end),'backgroundcolor'),'units',...
   'normalized','position',[0.45 0.2 .5 0.14],'value',0,'userdata',...
   [gui.pl(end) gui.tx(end)],'callback',['if get(gcbo,"value")==1; ',...
   'set(get(gcbo,"userdata"),"visible","on"); else; ',...
   'set(get(gcbo,"userdata"),"visible","off"); end;']);

%Set an appropriate text color based on background color
chil=get(gui.ChanControls(end),'children');
for i=1:length(chil)
   if mean(get(chil(i),'backgroundcolor'))<0.5; set(chil(i),'foregroundcolor','w'); end
end
set(gui.ChanControls(end),'userdata',Chan)
set(gui.HWMenu,'userdata',1)
```

```
if length(gui.ChanControls)==1
   set(gui.ChanColorSelect,'backgroundcolor',[.7 0 0])
elseif length(gui.ChanControls)==2
   set(gui.ChanColorSelect,'backgroundcolor',[0 0 0])
else
   set(gui.ChanColorSelect,'backgroundcolor',[0 0 .9])
end

temp=uimenu('parent',gui.AdvancedMenu(5),'label',ChanName,'tag','uimenu');
s=size(gTemp.InputRanges);
irange=gui.ai.channel(end).InputRange;
for i=1:s(1)
   subtemp=uimenu('parent',temp,'label',['[',num2str(gTemp.InputRanges(i,:)),']'],...
      'userdata',gui.ai.channel(end),'callback',...
      ['stop(get(get(gcbo,"userdata"),"parent")); ',...
      'set(get(gcbo,"userdata"),"inputrange",str2num(get(gcbo,"label"))); ',...
      'set(get(get(gcbo,"parent"),"children"),"checked","off"); ',...
      'set(gcbo,"checked","on"); gprime("ChangeTrig");']);
   if irange(1)==gTemp.InputRanges(i,1)&&irange(2)==gTemp.InputRanges(i,2)
      set(subtemp,'checked','on')
   end
end

gTemp=get(gui.ChanList,'string');
if isempty(gTemp)
   gTemp=' ';
end
 set(gui.TrigChan,'string',gTemp,...
   'callback','gprime("ChangeTrig")')
set(gui.TrigLevel,'callback',...
   ['if isnan(str2double(get(gcbo,"string"))); ',...
   'set(gcbo,"string",num2str(get(gcbo,"userdata"))); else; ',...
   'set(gcbo,"userdata",str2double(get(gcbo,"string"))); end; ',...
   'gprime("ChangeTrig")']);
 set(gui.fig,'userdata',gui)

initAI
```

4. MoveThresh

The MoveThresh function is an example of a frequently used process that I developed to handle a draggable line in an axis. The line object's "buttondownfcn" (button down function) property is set to call a function which activates a "windowbuttonmotionfcn" for the figure that contains the line. Whenever the mouse moves in the figure, the code detects if the mouse pointer is within the axis limits and sets the y value only (in this case) equal to that of the pointer location.

In this case, the channel name is stored in the root object's userdata for easy reference but other tools may be used to detect the axis in which the threshold line is located. For example, one could detect the parent of the threshold line (the axis).

The "windowbuttonupfcn" of the figure clears the "windowbuttonmotionfcn" when the user lets the mouse go and the trace stops updating its position. This action of the windowbuttonupfcn is set in the initialization of the figure (makegui). It is necessary to use these "windowbutton" functions as the trace itself contains only a "buttondownfcn" action.

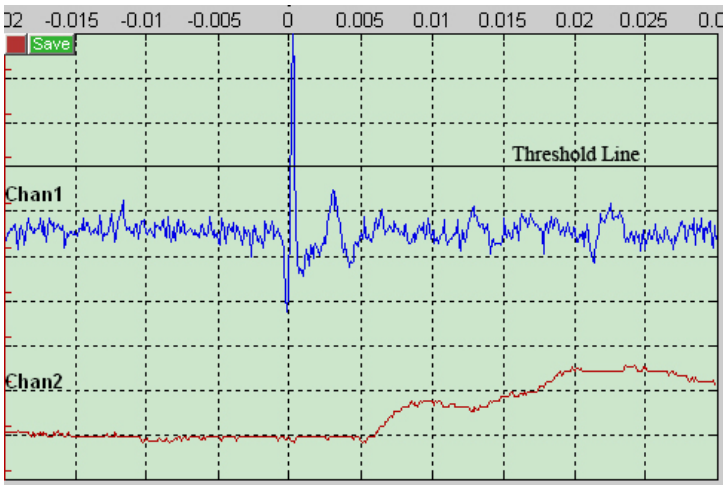

Figure A2.3 – An example of a system sweep triggered off of a threshold cross in the window. The threshold line is indicated and may be moved up or down relative to the blue trace by clicking it and dragging in the axis.

```
%_____________________________________________________________________
%Move the threshold bar
function MoveThresh(ChanName)
gui=get(findobj('tag','gSS07'),'userdata');
for i=1:length(gui.tx)
   if strcmpi(ChanName,get(gui.tx(i),'string'))
      ind=i; break;
   end
end
set(0,'userdata',ChanName)

set(gcf,'windowbuttonmotionfcn','gprime("ThreshOffset")')

function ThreshOffset
gui=get(findobj('tag','gSS07'),'userdata');
ChanName=get(0,'userdata');
thisAx=findobj(gui.ax,'tag',ChanName,'type','axes');
a=get(thisAx,'currentpoint');
ylim=get(thisAx,'ylim');
if a(1,2)>max(ylim)||a(1,2)<min(ylim); return; end
set(gui.TrigLevel,'string',num2str(round(10000*a(1,2))/10000),...
            'userdata',round(10000*a(1,2))/10000)
set(findobj('tag','gSS07thresh'),'ydata',[1 1]*get(gui.TrigLevel,'userdata'))
drawnow
```

5. ChangeTrig

This function acts as a "catch-all" reset for the analog input interface. It is called in many instances when the user makes a change to the behavior parameters for the analog input. This interface activates continuous, manual, channel voltage level, or external TTL (if available) trigger modes and controls sweep width.

For "Full Span" times greather than or equal to 200ms, the update period of the scope is equal to 200ms. A trace will be continuously displayed on the screen and extended in 200ms chunks. For traces shorter than 200ms, the update rate of the trace matches the scope width (i.e. 50ms or 100ms).

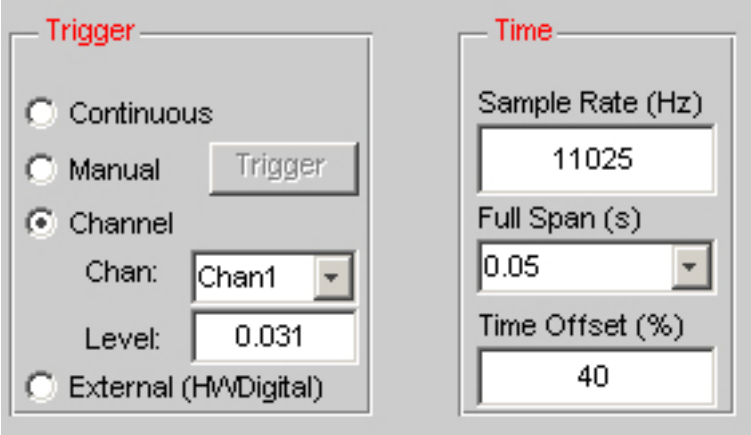

Figure A2.4 – The trigger mode and sweep time control panels. Changing these values calls the ChangeTrig function which resets the behavior of a single sweep of the analog input interface.

```
%_____________________________________________
%Called when Trigger State Changes (or to reset AI)
function ChangeTrig
gui=get(findobj('tag','gSS07'),'userdata');
for i=[gui.ChannelAddPanel gui.ChannelListPanel gui.RecControls...
       gui.TrigControls gui.TimeControls gui.ChanControls]
         try; set(get(i,'children'),'enable','on'); end
end

if ishandle(findobj('Name','g-PRIME Analysis'))
   s=get(gui.ChanList,'string');
   if ~isempty(s)
      set(gui.aSourceChan,'string',s)
      set(gui.aCSource,'string',strvcat('None',s))
      s=size(get(gui.aSourceChan,'string'));
      if get(gui.aSourceChan,'value')>s(1); set(gui.aSourceChan,'value',1); end
      if get(gui.aCSource,'value')>s(1); set(gui.aCSource,'value',1); end
   else
      set([gui.aSourceChan, gui.aCSource],'string',' ')
      s=size(get(gui.aSourceChan,'string'));
      if get(gui.aSourceChan,'value')>s(1); set(gui.aSourceChan,'value',1); end
      if get(gui.aCSource,'value')>s(1); set(gui.aCSource,'value',1); end
   end
end



if get(gui.TriggerDelay,'userdata')<0
```

```
    set(gui.TriggerDelay,'userdata',0,'string','0')
elseif get(gui.TriggerDelay,'userdata')>90
    set(gui.TriggerDelay,'userdata',90,'string','0')
end
    set(gui.timetext,'position',[round(get(gui.TriggerDelay,'userdata')/10)/10 1.15 0])

set([gui.TrigMan gui.TrigChan gui.TrigLevel],'enable','off')
if isempty(get(gui.ChanList,'string')); set(gui.Trig,'value',0);
    set(gui.Trig(1),'value',1); return; end
gui.ai.StopFcn='';
stop(gui.ai)
set(gui.HWMenu,'userdata',1)
set(gui.ai,'userdata',-1)
gui.ai.LoggingMode='Memory';
set(gui.RecStartStop,'value',0,'backgroundcolor','r','string','Record Start')
gui.ai.TriggerConditionValue=str2double(get(gui.TrigLevel,'string'));

delete(findobj('tag','gSS07thresh'))

%Continuous Triggering
if get(gui.Trig(1),'value')==1
    gui.ai.TriggerDelay=0;
    gui.ai.TriggerType='Immediate';
    gui.ai.TriggerRepeat=0;
    gui.ai.TriggerFcn='gprime("ClearPlot");';
    updateRate=get(gui.UpdateRate,'userdata');
    gui.ai.samplesacquiredfcncount=floor(gui.ai.samplerate*updateRate);
    gui.ai.samplespertrigger=inf;
    set(gui.fig,'userdata',gui)
    start(gui.ai)
end

%Manual Button Press Triggering
if get(gui.Trig(2),'value')==1
    set(gui.TrigMan,'enable','on')
    gui.ai.TriggerType='Manual';
    gui.ai.TriggerRepeat=inf;
    gui.ai.TriggerFcn='gprime("ClearPlot");';
    Span=get(gui.Refresh,'string');
    Span=str2num(Span(get(gui.Refresh,'value'),:));
    gui.ai.TriggerDelay=-.01*get(gui.TriggerDelay,'userdata')*Span;
    updateRate=get(gui.UpdateRate,'userdata');
    gui.ai.samplesacquiredfcncount=floor(gui.ai.samplerate*updateRate);
    gui.ai.samplespertrigger=floor(Span*gui.ai.samplerate);
    start(gui.ai)
```

```
end

%Level based Software Triggering
if get(gui.Trig(3),'value')==1
    set(gui.TrigChan,'string',get(gui.ChanList,'string'),'enable','on',...
        'callback','gprime("ChangeTrig")')
    set(gui.TrigLevel,'enable','on','callback',...
        ['if isnan(str2double(get(gcbo,"string"))); ',...
        'set(gcbo,"string",num2str(get(gcbo,"userdata"))); else; ',...
        'set(gcbo,"userdata",str2double(get(gcbo,"string"))); end; ',...
        'gprime("ChangeTrig")']);
    gui.ai.TriggerType='Software';
    ChanName=get(gui.TrigChan,'string');
    ChanName=ChanName(get(gui.TrigChan,'value'),:);
    %Display threshold Trace
    gTemp=get(findobj('tag',ChanName,'type','axes'),'xlim');
    gui.thresh=line(gTemp,[1 1]*str2num(get(gui.TrigLevel,'string')),'color','k',...
      'linewidth',1,'parent',findobj('tag',ChanName,'type','axes'),'tag',...
      'gSS07thresh','buttondownfcn',['gprime("MoveThresh",'',ChanName,'')']);

    Span=get(gui.Refresh,'string');
    Span=str2double(Span(get(gui.Refresh,'value'),:));
    updateRate=get(gui.UpdateRate,'userdata');
    gui.ai.samplesacquiredfcncount=floor(gui.ai.samplerate*updateRate);
    gui.ai.TriggerDelay=-.01*get(gui.TriggerDelay,'userdata')*Span;
    gui.ai.samplespertrigger=floor(gui.ai.samplerate*Span);

    for i=1:length(gui.ai.channel)
        if strcmpi(gui.ai.channel(i).ChannelName,ChanName)
            gui.ai.TriggerChannel=gui.ai.channel(i);
        end
    end
    gui.ai.TriggerCondition='Rising';
    gui.ai.TriggerConditionValue=str2double(get(gui.TrigLevel,'string'));
    gui.ai.TriggerRepeat=inf;
    gui.ai.TriggerFcn='gprime("ClearPlot");';
    start(gui.ai)
end

%Hardware Based digital triggering (throws error on winsound)
if get(gui.Trig(4),'value')==1
    gui.ai.TriggerDelay=0;
    try
        gui.ai.TriggerType='HwDigital';
    catch
```

```
        set(gui.Trig,'value',0)
        set(gui.Trig(1),'value',1)
        ChangeTrig
        return
    end
    gui.ai.TriggerRepeat=inf;
    Span=get(gui.Refresh,'string');
    Span=str2double(Span(get(gui.Refresh,'value'),:));
    updateRate=get(gui.UpdateRate,'userdata');
    gui.ai.TriggerFcn='gprime("ClearPlot");';
    gui.ai.samplesacquiredfcncount=floor(gui.ai.samplerate*updateRate);
    gui.ai.samplespertrigger=floor(Span*gui.ai.samplerate);
    gui.ai.TriggerConditionValue=2.5;

    start(gui.ai)
end
```

6. GoStartStop

The recording function handles the state of the big red "Record Start" button. When depressed, the button is colored green and a variety of tasks are carried out. The analog input interface is set to log data directly to disk (as well as to memory for visualization). The file name is indexed based on the user selected name if it is named something other than "temp.daq," and the entire scope interface is disabled so that the user can't make any changes to the display that would reset the recording session without first pushing the stop button.

If a fixed duration recording is set, the samplespertrigger property of the input device is set to the duration that the user has entered. In addition to the behavior of this function, the button's text is updated with the elapsed record time by the SAF function.

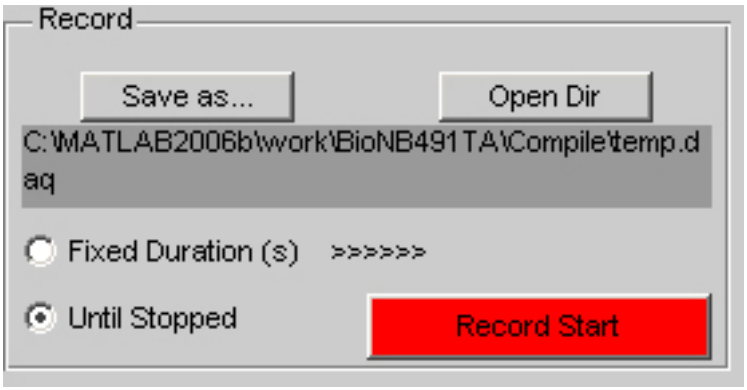

Figure A2.5 – The recording interface. The "Record Start" button calls "GoStartStop."

```
%_____________________________________________
%Control Recording
function GoStartStop
gui=get(findobj('tag','gSS07'),'userdata');
if isempty(gui.ChanControls);
   set(gui.RecStartStop,'value',0,'string','Record Start','backgroundcolor','r');...
      return;
end

switch get(gui.RecStartStop,'value')
   case 0
      for i=[gui.ChannelAddPanel gui.ChannelListPanel gui.RecControls...
            gui.TrigControls gui.TimeControls gui.ChanControls]
         set(get(i,'children'),'enable','on')
      end
      set(gui.RecStartStop,'backgroundcolor','r','string','Record Start')
      stop(gui.ai)
      ChangeTrig
   case 1
      for i=[gui.ChannelAddPanel gui.ChannelListPanel gui.RecControls...
            gui.TrigControls gui.TimeControls gui.ChanControls]
         set(get(i,'children'),'enable','off')
      end
      if get(gui.Trig(2),'value')==1; set(gui.TrigMan,'enable','on'); end
      set(gui.RecStartStop,'enable','on')
      set(gui.RecStartStop,'Backgroundcolor','g','string','Record Stop')
      stop(gui.ai)
      gui.ai.LoggingMode='Disk&Memory';

      fullstring=get(gui.RecFile,'string');
      slashes=strfind(fullstring,'\');
```

```
pname=fullstring(1:slashes(end));
fname=fullstring((slashes(end)+1):end);
%Itterate Files (other than temp.daq), matlab's analoginput log
%type doesn't seem to automatically index even when in index mode
if strcmpi(fname,'temp.daq')
   gui.ai.LogFileName=get(gui.RecFile,'string');
else
   fname=fname(1:end-4);
   tempfname=fname;
   gTemp=dir(pname);
   j=1;
   while 1==1
      p=0;
      for i=1:length(gTemp)
         if strcmpi([tempfname,'.daq'],gTemp(i).name)
            tempfname=[fname,num2str(j)];
            tempfname
            j=j+1;                    p=1;
         end
      end
      if p==0
         break
      end
   end
   gui.ai.LogFileName=[pname,tempfname,'.daq'];
   get(gui.ai,'LogFileName')
end
switch get(gui.Rec(1),'value')
   case 0  %Indefinite
      %gui.ai.samplespertrigger=inf;
   case 1  %Fixed Duration
      duration=str2double(get(gui.RecDuration,'string'));
      updateRate=get(gui.UpdateRate,'userdata');
      duration=ceil(duration/updateRate)*updateRate;
      set(gui.RecDuration,'string',num2str(duration),'userdata',duration)

      gui.ai.samplespertrigger=floor(duration*gui.ai.samplerate/...
         gui.ai.samplesacquiredfcncount)*gui.ai.SamplesAcquiredFcnCount;
      gui.ai.StopFcn='gprime("ChangeTrig")';
end
set(gui.HWMenu,'userdata',1)
gui.OverFlag=0;
set(gui.ai,'userdata',-1)
set(gui.fig,'userdata',gui)
start(gui.ai)
```

```
end
```

7. StimParam

StimParam is a fairly detailed function that acts as a "catch-all" for any changes to the waveform shape interface elements in the stimuluation panel. This function acquires all of the parameters from the UI, checks the state of the waveform and detects any parameter mismatches. The waveform is then generated based on the user supplied input parameters. The final waveform is not displayed as all samples, but simply as samples around the changing points (i.e. the derivative). Displaying an extended signal in the stimulus axis could cause graphics loading that would delay the behavior of the interface. This method of signal representation provides all of the useful information for signal reconstruction using the following code when the signal is prepared for output. This is effectively an integration of the derivative of the signal where only non-zero values are stored.

```
if get(gui.StimType(4),'value')~=1
   sig=get(gui.StimPl,'ydata');
   time=get(gui.StimPl,'xdata');

   rawt=linspace(0,time(end),gui.ao.samplerate*time(end));
   raws=zeros(length(rawt),1);
   for i=2:length(sig)
      raws((rawt>time(i-1))&rawt<time(i))=sig(i);
   end
   sig=raws;
end
```

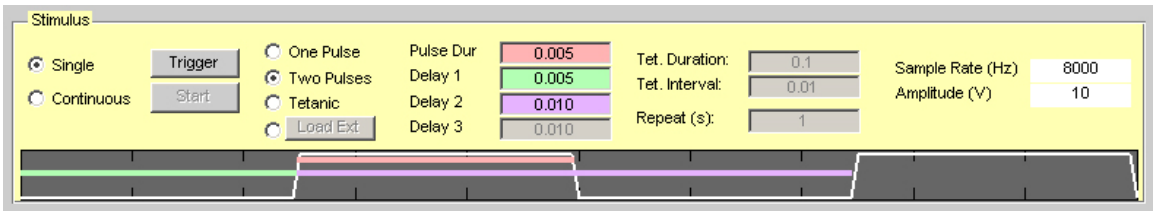

Figure A2.6 The Stimulus interface. Each element of this panel calls the "StimParam" function to check for parameter errors and construct a waveform based on the user specified inputs.

```
%_____________________________________________
%Handle Bad Values in Stimulus parameters
%Build Stimulus and Prep AO
function StimParam
gui=get(findobj('tag','gSS07'),'userdata');
stop(gui.ao)


daqinfo=daqhwinfo(gui.ao);
if get(gui.StimSRate,'userdata')<daqinfo.MinSampleRate
   gui.ao.samplerate=daqinfo.MinSampleRate;
elseif get(gui.StimSRate,'userdata')>daqinfo.MaxSampleRate
   gui.ao.samplerate=daqinfo.MaxSampleRate;
else
   gui.ao.samplerate=get(gui.StimSRate,'userdata');
end
set(gui.StimSRate,'string',num2str(gui.ao.samplerate),'userdata',gui.ao.samplerate)

if get(gui.StimMode(1),'value')==1
   set(gui.StimModeTrig,'enable','on')
   set(gui.StimModeStart,'enable','off','value',0,'string','Start')
else
   set(gui.StimModeTrig,'enable','off')
   set(gui.StimModeStart,'enable','on','value',0,'string','Start')
end

vals=[gui.StimSRate,gui.StimShape(1:4),gui.StimTetDur,gui.StimTetInt,gui.StimRepe
at,...
   gui.StimAmp];

if get(gui.StimType(4),'value')==1 %Load Arbitrary Stimulus
   %disable Signal Gen Interface
   set(vals,'enable','off')
   set(gui.StimTypeLoad,'enable','on')
```

```
    AllTraces=[gui.StimPlPulseDur gui.StimPlDelay1 gui.StimPlDelay2
gui.StimPlDelay3,...
        gui.StimPlTetDur gui.StimPlTetInt];
    set(AllTraces,'visible','off');
    return
else
    set(vals,'enable','on')
    set(gui.StimTypeLoad,'enable','off')
end

    set(gui.StimShape,'enable','off')
    set(gui.StimTetDur,'enable','off')
    set(gui.StimTetInt,'enable','off')

if get(gui.StimType(1),'value')==1
    set(gui.StimShape(1:2),'enable','on')
elseif get(gui.StimType(2),'value')==1
    set(gui.StimShape(1:3),'enable','on')
elseif get(gui.StimType(3),'value')==1
    set(gui.StimShape,'enable','on')
    set(gui.StimTetDur,'enable','on')
    set(gui.StimTetInt,'enable','on')
end

    set(gui.StimRepeat,'enable','off')
if get(gui.StimMode(2),'value')==1
    set(gui.StimRepeat,'enable','on')
end

set(gui.StimViolation,'visible','off')

%Check Parameters %Sample Rate %Delays
%StimShape(1:4) = dur +3x delay
%StimTetDur & StimTetInt = Tetanic Params
%StimRepeat = Total Signal Duration
PulseDur=get(gui.StimShape(1),'userdata');
Delay1=get(gui.StimShape(2),'userdata');
Delay2=get(gui.StimShape(3),'userdata');
Delay3=get(gui.StimShape(4),'userdata');
TetDur=get(gui.StimTetDur,'userdata');
TetInt=get(gui.StimTetInt,'userdata');
Amp=get(gui.StimAmp,'userdata');
Repeat=get(gui.StimRepeat,'userdata');

AllVals=[PulseDur, Delay1, Delay2, Delay3, TetDur, TetInt, Repeat];
```

```
violation=0;

if sum(AllVals([1,3:end])<=(1/gui.ao.samplerate))>0; violation=1; end

if PulseDur>Delay2&&get(gui.StimType(1),'value')==0; violation=1; end
if PulseDur>Delay3&&get(gui.StimType(3),'value')==1;  violation=1; end
if PulseDur>TetInt&&get(gui.StimType(3),'value')==1;  violation=1; end
if (Delay1+PulseDur)>Repeat&&get(gui.StimType(1),'value')==1&&...
      get(gui.StimMode(2),'value')==1; violation=1; end
if (Delay1+Delay2+PulseDur)>Repeat&&get(gui.StimType(2),'value')==1&&...
      get(gui.StimMode(2),'value')==1; violation=1; end
if
(Delay1+Delay2+TetDur+Delay3+PulseDur)>Repeat&&get(gui.StimType(3),'value')
==1&&...
      get(gui.StimMode(2),'value')==1; violation=1; end
if TetInt>TetDur&&get(gui.StimType(3),'value')==1;  violation=1; end

if Amp>10||isempty(Amp); set(gui.StimAmp,'string','10','userdata',10); Amp=10; end
if Amp<=0; set(gui.StimAmp,'string','0','userdata',0); Amp=0; end

if violation==1
   set(gui.StimViolation,'visible','on','string','Stimulus Parameter Error',...
      'fontsize',.7)
   return
end

%Construct Signal

if get(gui.StimType(1),'value')==1
      sig=zeros(1,round(gui.ao.samplerate*(Delay1+PulseDur)));
elseif get(gui.StimType(2),'value')==1
      sig=zeros(1,round(gui.ao.samplerate*(Delay1+Delay2+PulseDur)));
elseif get(gui.StimType(3),'value')==1
      sig=zeros(1,round(gui.ao.samplerate*(Delay1+Delay3+TetDur+PulseDur)));
end

PulseDur=round(PulseDur*gui.ao.samplerate);
Delay1=round(Delay1*gui.ao.samplerate)+1;
Delay2=round(Delay2*gui.ao.samplerate);
Delay3=round(Delay3*gui.ao.samplerate);
TetDur=round(TetDur*gui.ao.samplerate);
TetInt=round(TetInt*gui.ao.samplerate);

if get(gui.StimType(1),'value')==1 %Create Single Pulse Signal
   sig(Delay1:(Delay1+PulseDur))=Amp;
```

```
end

if get(gui.StimType(2),'value')==1 %Create Double Pulse Signal

   sig(Delay1:(Delay1+PulseDur))=Amp;
   sig((Delay1+PulseDur):(Delay1+PulseDur+Delay2))=0;
   sig((Delay1+Delay2):(Delay1+Delay2+PulseDur))=Amp;
end

if get(gui.StimType(3),'value')==1 %Create Double + Tetanic Pulse Signal
   sig(Delay1:(Delay1+PulseDur))=Amp;
   sig((Delay1+PulseDur):(Delay1+Delay2))=0;

   tetsig=zeros(1,TetInt);
   tetsig(1:PulseDur)=Amp;
   tetsig=repmat(tetsig,[1,floor(TetDur/TetInt)]);
   sig((Delay1+Delay2):(Delay1+Delay2+length(tetsig)-1))=tetsig;
   finalval=max(find(sig~=0));
   sig(finalval:(finalval+Delay3))=0;
   sig((finalval+Delay3):(finalval+Delay3+PulseDur))=Amp;
   finalval2=max(find(sig~=0));
   sig=sig(1:finalval2);
end

Repeat=get(gui.StimRepeat,'userdata');
if get(gui.StimMode(2),'value')==1&get(gui.StimType(4),'value')==0
   sig=[sig,zeros(1,round(Repeat*gui.ao.SampleRate-length(sig)))];
end


%Update Display and Parameter Bars
sig(end)=0;
time=linspace(0,length(sig)/gui.ao.samplerate,length(sig));

dsig=[0,diff(sig)];
critPoints=find(dsig~=0);
Points(1:3:(length(critPoints)*3))=critPoints-1;
Points(2:3:(length(critPoints)*3))=critPoints;
Points(3:3:(length(critPoints)*3))=critPoints+1;
if Points(1)~=1; Points=[1,Points]; end
if Points(end)~=length(sig); Points=[Points,length(sig)]; end

Points(Points>length(sig))=[];
Points(Points<1)=[];
```

```
time=time(Points);
sig=sig(Points);

set(gui.StimPl,'xdata',time,'ydata',sig,'visible','on')

set(gui.StimAx,'xlim',[0 max(time)],'ylim',[-.1 11])
if get(gui.StimMode(2),'value')==1
   set(gui.StimAx,'xlim',[0 Repeat])
end

PulseDur=get(gui.StimShape(1),'userdata');
Delay1=get(gui.StimShape(2),'userdata');
Delay2=get(gui.StimShape(3),'userdata');
Delay3=get(gui.StimShape(4),'userdata');
TetDur=get(gui.StimTetDur,'userdata');
TetInt=get(gui.StimTetInt,'userdata');

AllTraces=[gui.StimPlPulseDur gui.StimPlDelay1 gui.StimPlDelay2
gui.StimPlDelay3...
   gui.StimPlTetDur gui.StimPlTetInt];
set(AllTraces,'visible','off');

if get(gui.StimType(1),'value')==1 %Single Pulse Signal
   set(gui.StimPlDelay1,'xdata',[0 Delay1],'visible','on')
   set(gui.StimPlPulseDur,'xdata',[Delay1 Delay1+PulseDur],'visible','on')
end

if get(gui.StimType(2),'value')==1 %Double Pulse Signal
   set(gui.StimPlDelay1,'xdata',[0 Delay1],'visible','on')
   set(gui.StimPlPulseDur,'xdata',[Delay1 Delay1+PulseDur],'visible','on')
   set(gui.StimPlDelay2,'xdata',[Delay1 Delay1+Delay2],'visible','on')
end

if get(gui.StimType(3),'value')==1 %Tetanic Pulse Signal
   set(gui.StimPlDelay1,'xdata',[0 Delay1],'visible','on')
   set(gui.StimPlPulseDur,'xdata',[Delay1 Delay1+PulseDur],'visible','on')
   set(gui.StimPlDelay2,'xdata',[Delay1 Delay1+Delay2],'visible','on')
   set(gui.StimPlTetInt,'xdata',[Delay1+Delay2 Delay1+Delay2+TetInt],'visible','on')
   set(gui.StimPlTetDur,'xdata',[Delay1+Delay2 finalval/gui.ao.samplerate],...
      'visible','on')
   set(gui.StimPlDelay3,'xdata',...
      [finalval/gui.ao.samplerate finalval/gui.ao.samplerate+Delay3],'visible','on')
end
```

8. SAF

"SAF" stands for "Samples Acquired Function" and is the main visualization segment of the code responsible for reading data from the analog input object and displaying it on the screen. Given the state of the interface, this function may also call the real time analysis calculation function or the FFT/Spectrogram visualization function.

This function simply reads the sweep data from the interface, cycles through the plots associated with each channel and updates the data. If the sweep length is longer than 200ms, the interface will extract the current trace and tack onto the trace in 200ms increments.

```
%_____________________________________________
%Update Graphics with new data
function SAF
gui=get(findobj('tag','gSS07'),'userdata');
if isempty(gui); return; end
[data t abstime]=getdata(gui.ai,gui.ai.samplesacquiredfcncount);

if ishandle(findobj('name','g-PRIME Analysis'))&...
   ishandle(findobj('Label','Scope Channel'))
      if ishandle(gui.aLink(3))&gui.aLink(3)~=0
      if strcmpi(get(gui.aLink(3),'checked'),'on')
         corrInd=get(gui.aCSource,'value');
         corrInd=(corrInd-1)*strcmpi(get(gui.aResults(8),'checked'),'on');
         SelInd=get(gui.aSourceChan,'value');
         if corrInd>0
            s=size(data);
            if corrInd>s(2)
               set(gui.aCSource,'value',1)
               corrInd=1;
            end
            aRTCalc(data(:,SelInd), t, SelInd,data(:,corrInd));
         else
```

```
            s=size(data);
            if SelInd>s(2)
               set(gui.aSourceChan,'value',1)
               SelInd=1;
            end
            aRTCalc(data(:,SelInd), t, SelInd);
         end
      end
      end

end

Span=get(gui.Refresh,'string');
Span=str2double(Span(get(gui.Refresh,'value'),:));
updateRate=get(gui.UpdateRate,'userdata');

%Real Time Visualization Tools
if strcmpi(get(gui.VisMenu(2),'checked'),'on')|...
   strcmpi(get(gui.VisMenu(3),'checked'),'on')
   %Activate Axes and Call Visualization Function
   SelInd=get(gui.ChanList,'value');
   SAFVisualization(data(:,SelInd), t, SelInd)
end

%Display Data
for i=1:length(gui.pl)
   Chan=get(gui.ChanControls(i),'userdata');

   if get(Chan.ACCouple,'value')==1; data(:,i)=data(:,i)-mean(data(:,i)); end
   oldData=get(gui.pl(i),'ydata');
   extGain=get(Chan.ExternalGain,'userdata');

   if length(oldData)<(Span*gui.ai.samplerate)&get(gui.HWMenu,'userdata')==0
      newData=[oldData';(data(:,i)/extGain)];
   else
      newData=data(:,i)/extGain;
   end

   set(gui.pl(i),'ydata',newData,'xdata',1:length(newData),'visible','on');
   set(gui.tx(i),'position',[1 0 0],'visible','on');
   if get(Chan.Show,'value')==0; set([gui.pl(i) gui.tx(i)],'visible','off'); end

   offset=str2double(get(Chan.Offset,'string'));
   if isnan(offset); offset=get(Chan.Offset,'userdata'); end
   VpD=get(Chan.VpD,'string');
```

```
    VpD=str2double(VpD(get(Chan.VpD,'value'),:));
    offset=.01*offset*VpD*5;

    TempRange=VpD*[-5 5]-offset;
    set(gui.ax(i),'ylim',TempRange,'xlim',[0 1]*Span*gui.ai.samplerate)
end

set(gui.backax,'xlim',[0 1],'xtick',linspace(-1,1,21),'xticklabel',...
    round((linspace(-Span,Span,21)-...
    Span*get(gui.TriggerDelay,'userdata')/100)*5000)/5000,'ylim',[-1 1])


set(gui.HWMenu,'userdata',0)

if get(gui.RecStartStop,'value')==1
    set(gui.RecStartStop,'string',...
        ['Record Stop (',num2str(floor(gui.ai.SamplesAcquired/gui.ai.SampleRate)),')']);
end
set(findobj('tag','gSS07thresh'),'xdata',...
    get(get(findobj('tag','gSS07thresh'),'parent'),'xlim'));

drawnow
```

9. SAFVisualization

This is the visualization code for a spectral representation of the signal. It either appends the decimated FFT to a running spectrogram or it displays the FFT in a bar graph style plot. The FFT of the signal is calculated and 100 log or linear spaced points are interpolated out of the amplitude values. The Spectrogram is shifted one column left and the new column of data is inserted. In order to auto-scale the axis to eliminate the need for axis controls, the axis auto-sets to the maximum amplitude value of the FFT if the value is greater than the current axis range or the axis range decreases to 95% of its previous value per sweep. This provides a smooth adaption over time to the current frequency content of the signal.

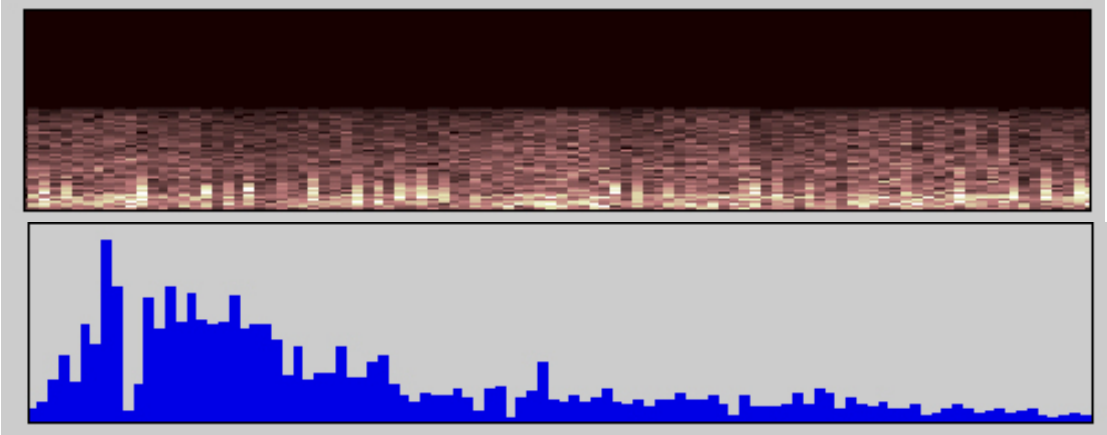

Figure A2.7 An illustration of the two visualization options. The spectrogram (top) consists of 100 sweeps (width) of a 100 point FFT (height). The sweep FFT (bottom) illustrates the immediate FFT for a sweep only.

```
%_____________________________________________________________________
% Handle Updating the Scope Visualizations (FFT/Spectrogram)
function SAFVisualization(data, time, SelInd)
gui=get(findobj('tag','gSS07'),'userdata');

    fsig=abs(fft((data-mean(data)).*hamming(length(data))));
    fsig=fsig(1:ceil(end/2));
    freqs=linspace(0,gui.ai.samplerate/2,length(fsig));
    if get(gui.visAx,'userdata')==0
        ifsig=interp1(freqs,fsig,...
            logspace(2,log10(max(freqs)),length(gui.fftVisPatch)),'linear');
    else
        ifsig=interp1(freqs,fsig,...
            linspace(min(freqs),max(freqs),length(gui.fftVisPatch)),'linear');
    end

    ifsig(isnan(ifsig))=10^-7;
    ifsig(ifsig<=0)=10^-7;

%Raw FFT Visualization
if strcmpi(get(gui.VisMenu(2),'checked'),'on')
    for i=1:length(ifsig)
        set(gui.fftVisPatch(i),'ydata',[0 0 1 1]*ifsig(i),...
            'facecolor',get(gui.pl(SelInd),'color'))
    end
    ylim=get(gui.visAx,'ylim');
    if max(ylim)<max(ifsig)
```

```
        ylim=[0 max(ifsig)];
    else
        ylim=[0 max(ylim)*.95];
    end
   set(gui.visAx,'xlim',[.5 length(ifsig)+.5],'ylim',ylim,'color','none','xdir','normal')
end

%Spectrogram Visualization
if strcmpi(get(gui.VisMenu(3),'checked'),'on')
    premax=get(gui.SpectVisImage,'userdata');
    cmap=pink(100);
    maxv=premax;
    maxs=max(ifsig);
    maxuse=maxv*(maxv>=maxs)+maxs*(maxs>maxv)+1*(maxs==0|maxv==0);
    set(gui.SpectVisImage,'userdata',maxuse*.95);

    Spect=get(gui.SpectVisImage,'cdata');
    Spect(:,2:end,:)=Spect(:,1:(end-1),:);
    ifsig=ceil(99.1*ifsig/maxuse);
    ifsig=cat(3,cmap(ifsig,1),cmap(ifsig,2),cmap(ifsig,3));
    Spect(:,1,:)=ifsig;

    set(gui.SpectVisImage,'cdata',Spect)

end
```

10. ConvertTXT

ConvertTXT is a brief example of a variety of data saving functions available to the user. In this case, a previously recorded raw .daq file is converted into a multi-column, comma delimited, text file for offloading to another analysis program such as Microsoft Excell.

```
%_____________________________________________________________________
%Export a .daq file into a comma delimited text file
function ConvertTXT
%dlmwrite
[fname, pname] = uigetfile('*.daq', 'File to be Converted (*.daq to *.txt)',...
    get(gui.DataMenu(1),'userdata'));
```

```
if isequal(fname,0) || isequal(pname,0)
  return
end
   set(gui.DataMenu(1),'userdata',pname)
[fname2, pname2] = uiputfile('*.txt', 'Target Text File Name (*.daq to *.txt)',...
   get(gui.DataMenu(1),'userdata'));
if isequal(fname2,0) || isequal(pname2,0)
  return
end
   set(gui.DataMenu(1),'userdata',pname2)

[data time abstime ev daqinfo]=daqread([pname,fname]);
out=[time,data];
dlmwrite([pname2,fname2],out,'precision','%.12f')
```

11. aConnect

This function acts in response to user input in the analysis "source" drop down menu. The function is a switch yard which sets analysis interface components into appropriate states for the selected mode (real time or offline) and handles loading a variety of file types into the interface (offline mode).

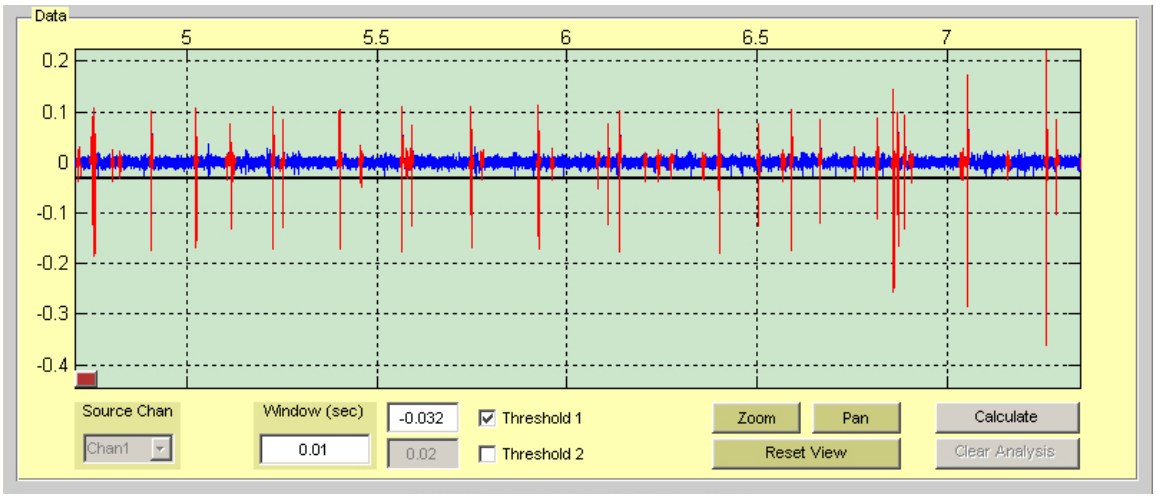

Figure A2.8 A file loaded into the analysis interface. All options in the "Source" menu in the analysis window call aConnect. The function is responsible for linking data to the analysis window and for setting the analysis window state based on Real-Time or Offline analysis state.

```
%_____________________________________________________________________
%Link the analysis interface to a data source
function aConnect
gui=get(findobj('tag','gSS07'),'userdata');

delete(findobj('tag','timeTrace'))

if strcmpi(get(gui.aLink(3),'checked'),'on')&length(gui.pl)~=0
   set(gui.aSourceAx,'xlim',[1 gui.ai.samplesacquiredfcncount],...
      'ylim',gui.ai.channel(get(gui.ChanList,'value')).InputRange)
elseif strcmpi(get(gui.aLink(4),'checked'),'on')
   set(gui.aSourceAx,'xlimmode','auto','ylimmode','auto')
end

if strcmpi(get(gui.aLink(2),'checked'),'on')
   set(gui.aTitle,'string','Select A Signal Source','fontsize',.8)
   set(gui.aSourcePl,'xdata',nan,'ydata',nan)
   set(gui.aAnalPl,'xdata',nan,'ydata',nan)
   set([gui.aHPMenu, gui.aLPMenu],'checked','off')
   set([gui.aHPMenu(1), gui.aLPMenu(1)],'checked','on')
   set(gui.aMasterCalc,'enable','off')
   set(gui.aMasterRESET,'enable','off')
   set(gui.aResults(7),'enable','off')
   delete(findobj('tag','gSS07aTrace'))
   set(gui.aResults(8),'checked','off','enable','off')
   set(gui.aAnalAx,'userdata',[]);
   set([gui.aEDenThresh gui.aFreqThresh],'visible','off','value',0)
   aEDenThresh; aFreqThresh;
   aChangeAnalysisDisplay
   aShowCorr
   set(gui.aSourceChan,'enable','off')
   set([gui.aSquareButton gui.aPolygonButton gui.aPolyInverse],'visible','off')
   set(gui.aMasterDisplayA,'enable','off')
   return
end

if strcmpi(get(gui.aLink(3),'checked'),'on')
   try
      set(gui.aSourceAx,'ylim',gui.channel(get(gui.ChanList,'value')).InputRange)
   end
   set(gui.aMasterCalc,'enable','off')
```

```
    set(gui.aMasterRESET,'enable','on')
    set(gui.aSourcePl,'xdata',nan,'ydata',nan)
    set([gui.aHPMenu, gui.aLPMenu],'checked','off')
    set([gui.aHPMenu(1), gui.aLPMenu(1)],'checked','on')
    delete(findobj('tag','gSS07aTrace'))
    set(gui.aResults(7),'enable','off')
    set(gui.aResults(8),'checked','off','enable','on')
    set(gui.aSourceChan,'enable','on')
    set(gui.aAnalAx,'userdata',[]);
    set([gui.aEDenThresh gui.aFreqThresh],'visible','off','value',0)
    aEDenThresh; aFreqThresh;
    aChangeAnalysisDisplay
    aShowCorr
    set([gui.aSquareButton gui.aPolygonButton gui.aPolyInverse],'visible','off')
    set(gui.aMasterDisplayA,'enable','on')

    return
end

if strcmpi(get(gui.aLink(4),'checked'),'on')

    filterspec={'*.daq','(*.daq) Native Format';...
    '*.wav','(*.wav) Wave Audio';...
    '*.mat','(*.mat) "data" and "time"';...
    '*.txt','(*.txt) Columns: time, data1, data2,...';...
    '*.nbb','(*.nbb) StimScope File'};
    [fname, pname, filterindex] = uigetfile(filterspec,'Load Arbitrary Data File',...
        get(gui.DataMenu(1),'userdata'));
    if isequal(fname,0) | isequal(pname,0)
      return
    end
    set(gui.DataMenu(1),'userdata',pname)
    set(gui.aResults(8),'checked','off','enable','off')
    switch filterindex  %Need Data (one column), time, sRate for each
       case 2
          [data sRate]=wavread([pname,fname]);
          s=size(data);
          if s(2)>1
           [j v]=listdlg('PromptString','Select a Channel','SelectionMode','Single',...
                'ListString',{'Left','Right'});
             if v==0; return; end
             data=data(:,j);
          end
          time=linspace(0,length(data)/sRate,length(data));
          set(gui.aSourceAx,'xlim',[min(time) max(time)],'ylim',...
```

```
        [-1.1 1.1],'xticklabelmode','auto')
case 1
    daqinfo=daqread([pname,fname],'info');
    sRate=daqinfo.ObjInfo.SampleRate;
    j=1; p=1;
    if length(daqinfo.ObjInfo.Channel)>1
     [j v]=listdlg('PromptString','Select a Channel','SelectionMode','Single',...
           'ListString',{daqinfo.ObjInfo.Channel.ChannelName});
        if v==0; return; end

    end
    if daqinfo.ObjInfo.TriggersExecuted>1
        for ii=1:daqinfo.ObjInfo.TriggersExecuted
            trigList{ii}=num2str(ii);
        end
     [p v]=listdlg('PromptString','Select a Trigger','SelectionMode','Single',...
           'ListString',trigList);
    end
 [data time abstime ev daqinfo]=daqread([pname,fname],'channels',j,'triggers',p);
    set(gui.aSourceAx,'xlim',[min(time) max(time)],'ylim',...
        daqinfo.ObjInfo.Channel(1).InputRange,'xticklabelmode','auto')
case 3
    load([pname,fname])
    s=size(data);
    if s(2)>1
        for i=1:s(2)
            Lstring{i}=['Data Column ',num2str(i)];
        end
     [j v]=listdlg('PromptString','Select a Channel','SelectionMode','Single',...
          'ListString',Lstring);
        if v==0; return; end
        data=data(:,j);
    end
    sRate=round(1/(time(2)-time(1)));
    set(gui.aSourceAx,'xlim',[min(time) max(time)],'ylim',...
        [min(data) max(data)],'xticklabelmode','auto')

case 4
    temp=load([pname,fname]);
    time=temp(:,1);
    data=temp(:,2:end);
    sRate=round(1/(temp(2,1)-temp(1,1)));
    s=size(data);
    if s(2)>1
        for i=1:s(2)
```

```
            Lstring{i}=['Data Column ',num2str(i)];
          end
        [j v]=listdlg('PromptString','Select a Channel','SelectionMode','Single',...
            'ListString',Lstring);
          if v==0; return; end
          data=data(:,j);
      end
      set(gui.aSourceAx,'xlim',[min(time) max(time)],'ylim',...
        [min(data) max(data)],'xticklabelmode','auto')
  case 5
      load([pname,fname],'-mat')
      [j v]=listdlg('PromptString','Select a Channel','SelectionMode','Single',...
          'ListString',{'Channel 1','Channel 2'});
        if v==0; return; end
      sRate=round(1/abs(diff(data.x(1:2))));
      time=data.x;
      data=data.y(:,j);

      set(gui.aSourceAx,'xlim',[min(time) max(time)],'ylim',...
        [min(data) max(data)],'xticklabelmode','auto')
end
set(gui.aAnalAx,'userdata',[]);
set([gui.aEDenThresh gui.aFreqThresh],'visible','off','value',0)
aEDenThresh; aFreqThresh;
aChangeAnalysisDisplay

if min(time)<0; time=time+abs(min(time)); end

set([gui.aThresh1Pl gui.aThresh2Pl],'visible','off')
set(gui.aMasterCalc,'userdata',sRate)
set(gui.aTitle,'string',[pname,fname],'fontsize',.5,...
  'userdata',[pname,fname])
set(gui.aSourcePl,'xdata',time,'ydata',data,'userdata',data)
set(gui.aAnalPl,'xdata',NaN,'ydata',NaN)
set([gui.aHPMenu, gui.aLPMenu],'checked','off')
set([gui.aHPMenu(1), gui.aLPMenu(1)],'checked','on')
set(gui.aMasterCalc,'enable','on')
set(gui.aResults(7),'enable','on')
delete(findobj('tag','gSS07aTrace'))
set(gui.aSourceChan,'enable','off')
set(gui.aMasterDisplayA,'enable','off')
set([gui.aSquareButton gui.aPolygonButton gui.aPolyInverse],'visible','on')
set(gui.aSourceAx,'xlimmode','auto','xtickmode','auto','xticklabelmode','auto',...
  'ylimmode','auto','ytickmode','auto','yticklabelmode','auto')
ranges=max(time);
```

```
    set(gui.aSourceAx,'xlim',[0 max(time)],'userdata',ranges)

    %Draw Time Borders
    gTemp(1)=line([0 0],get(gui.aSourceAx,'ylim'),'parent',gui.aSourceAx,'color','r',...
        'tag','timeTrace','userdata',1,'linewidth',1,'marker','o');
    gTemp(2)=line([1
1]*max(time),get(gui.aSourceAx,'ylim'),'parent',gui.aSourceAx,...
        'color','r','tag','timeTrace','userdata',2,'linewidth',1,'marker','o');
    set(gTemp(1),'buttondownfcn','gprime("aDragTimeBar",1)')
    set(gTemp(2),'buttondownfcn','gprime("aDragTimeBar",2)')
    aShowCorr
    analysisThreshSet
    aFiltCalc
end
```

12. aFiltCalc

In order to apply a filter to a signal in real time or during offline analysis, the filter coefficients are calculated in advance. A fairly continuous logarithmic spread of frequencies are available for high-pass or low-pass filtering. The filter coefficients are calculated using a 3rd order Butterworth representation with critical frequency at the selected value from the drop-down "Analysis" menu.

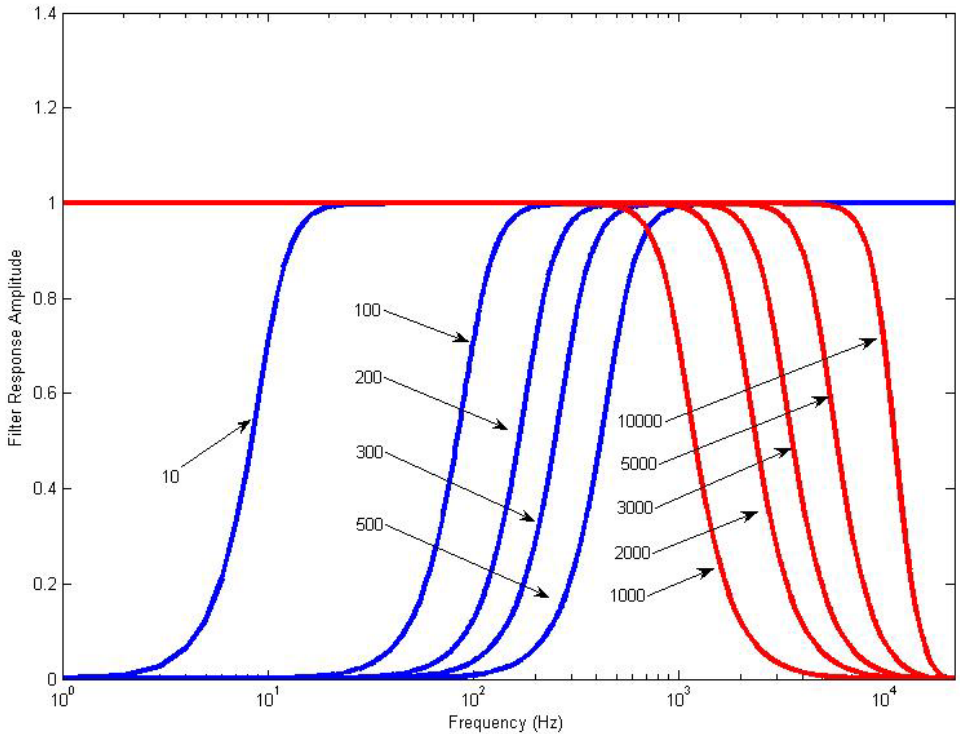

Figure A2.9 The transfer functions for all third order Butterworth filters available in the analysis interface. Band-pass filters of this kind have a spectrum similar in shape to an action potential pulse. A Buttorworth band-pass acts similarly to a match filter for a group of spikes. High-pass filters are blue (left) and low pass filters are red (right).

```
%_____________________________________________________________
%Precalculate Filter Coefficients
function aFiltCalc(varargin)
gui=get(findobj('tag','gSS07'),'userdata');
sRate=get(gui.aMasterCalc,'userdata');
filt.sRate=sRate;
if strcmpi(get(gui.aLink(3),'checked'),'on');
   sRate=gui.ai.Samplerate; set(gui.aMasterCalc,'userdata',gui.ai.Samplerate);
end
if isempty(sRate); return; end

for i=1:length(gui.aHPMenu)
   if strcmpi(get(gui.aHPMenu(i),'checked'),'on')
      fHP=get(gui.aHPMenu(i),'label');
   end
end

fHP=str2double(fHP);
```

```
if isnan(fHP)
   filt.b=1; filt.a=1;
else
     if fHP/(sRate/2)>=1;
       set(gui.aHPMenu,'checked','off')
       set(gui.aHPMenu(1),'checked','on')
       aFiltCalc; return;
     end
     [filt.b filt.a]=butter(3,fHP/(sRate/2),'high');
end
set(gui.aHP,'userdata',filt)

for i=1:length(gui.aLPMenu)
   if strcmpi(get(gui.aLPMenu(i),'checked'),'on')
      fLP=get(gui.aLPMenu(i),'label');
   end
end
fLP=str2double(fLP);
if isnan(fLP)
   filt.b=1; filt.a=1;
else
   if fLP/(sRate/2)>=1;
      set(gui.aLPMenu,'checked','off')
      set(gui.aLPMenu(1),'checked','on')
      aFiltCalc; return;
   end
      [filt.b filt.a]=butter(3,fLP/(sRate/2),'low');
end
set(gui.aLP,'userdata',filt)
```

13. aChangeAnalysisDisplay

The analysis parameters, when calculated, are stored in a struct and are accessed by this function for display. Based on the user selected options in the "Events" panel of the analysis window, a certain set of parameters is selected from the analysis metric struct. If a histogram is desired, the function will activate and calculate the display. All axes limits and scale (log/linear) are set based on specifications in the analysis window.

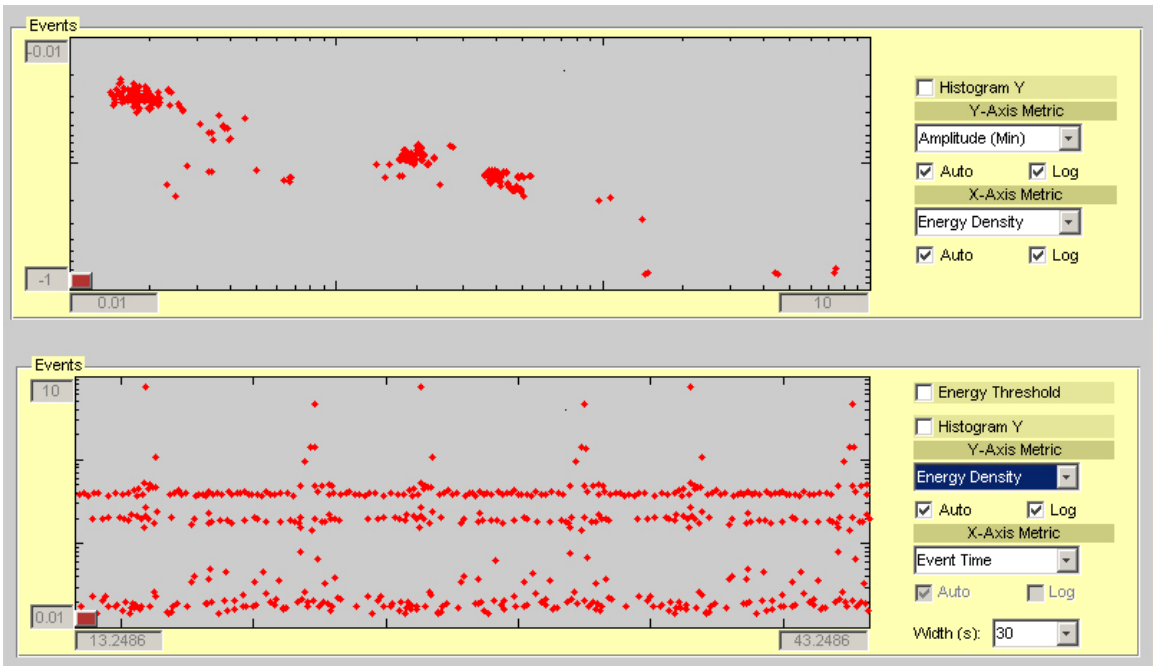

Figure A2.10 Two different data displays in real time analysis mode with a repeating, pre-recorded, extracellular nerve recording. Clusters corresponding to multiple action potentials can be seen clearly in the Energy Density vs Minimum Amplitude display (top) and amplitude levels of multiple spikes can clearly be visualized in the Energy Density vs Time plot (bottom)

```
%_____________________________________________________________
%Function called to update the Analysis Plots
function aChangeAnalysisDisplay
gui=get(findobj('tag','gSS07'),'userdata');
analysis=get(gui.aAnalAx,'userdata');
if isempty(analysis);
   set(gui.aYhist,'value',0)
   for i=1:length(gui.aHistPatch)
     set(gui.aHistPatch(i),'xdata',[0 1 1 0]*0,'ydata',...
        [-.5 -.5 .5 .5]*0)
   end
   set(gui.aAnalAx,'position',[0.05 0.1 0.71 0.9])
   set(gui.aAnalAx2,'visible','off')
   set(gui.aHistPatch,'visible','off')
   set(gui.aXt,'position',[0.68 0.01 0.08 0.08])
   set(gui.aAnalPl,'xdata',NaN,'ydata',NaN)
   return;
end

% ('Event Time','Rate','Interval','Amplitude(Min)','Amplitude(Max)',
```

```
% 'FFT Peak','FFT Sum')
switch get(gui.aXval,'value')
   case 1; xdat=analysis.threshcross;
   case 2;
         if length(analysis.threshcross)<2;
            xdat=nan*ones(length(analysis.threshcross),1);
         else xdat=1./diff(analysis.threshcross(:));
         end
   case 3;
         if length(analysis.threshcross)<2;
            xdat=nan*ones(length(analysis.threshcross),1);
         else xdat=diff(analysis.threshcross(:));
         end
   case 4; xdat=analysis.minamps;
   case 5; xdat=analysis.maxamps;
   case 6; xdat=analysis.fpeaks;
   case 7; xdat=analysis.fsum;
end
switch get(gui.aYval,'value')
   case 1; ydat=analysis.threshcross;
   case 2;
         if length(analysis.threshcross)<2;
            ydat=nan*ones(length(analysis.threshcross),1);
         else ydat=1./diff(analysis.threshcross(:));
         end
   case 3;
         if length(analysis.threshcross)<2;
            ydat=nan*ones(length(analysis.threshcross),1);
         else ydat=diff(analysis.threshcross(:));
         end
   case 4; ydat=analysis.minamps;
   case 5; ydat=analysis.maxamps;
   case 6; ydat=analysis.fpeaks;
   case 7; ydat=analysis.fsum;
end
if length(xdat)>length(ydat);  xdat=xdat(2:end);   end
if length(ydat)>length(xdat);  ydat=ydat(2:end);   end

set(gui.aAnalPl,'xdata',xdat,'ydata',ydat)
%Set axes scale and limits
if get(gui.aXlog,'value'); xscale='log'; else xscale='linear'; end
if get(gui.aYlog,'value'); yscale='log'; else yscale='linear'; end
set(gui.aAnalAx,'yscale',yscale,'xscale',xscale)

if get(gui.aXauto,'value');
```

```
    set(gui.aAnalAx,'xlimmode','auto');
    set(gui.aXb,'string',num2str(min(get(gui.aAnalAx,'xlim'))),'enable','off')
    set(gui.aXt,'string',num2str(max(get(gui.aAnalAx,'xlim'))),'enable','off')
else
    set(gui.aAnalAx,'xlimmode','manual')
    xrange=sort([str2double(get(gui.aXb,'string')) str2double(get(gui.aXt,'string'))]);
    set(gui.aAnalAx,'xlim',xrange);
    set(gui.aXb,'string',num2str(xrange(1)),'userdata',xrange(1),'enable','on')
    set(gui.aXt,'string',num2str(xrange(2)),'userdata',xrange(2),'enable','on')
end

if isempty(analysis.threshcross); analysis.threshcross=0; end

if get(gui.aXval,'value')==1&strcmpi(get(gui.aLink(3),'checked'),'on')
    set([gui.aXwidthT,gui.aXwidth],'visible','on')
    set(gui.aXauto,'enable','off','value',1)
    set(gui.aXlog,'enable','off','value',0)
    xrange=[-str2double(popupstr(gui.aXwidth)) 0]+max(analysis.threshcross);
    set(gui.aAnalAx,'xlim',xrange)
    set(gui.aXb,'string',num2str(xrange(1)),'userdata',xrange(1))
    set(gui.aXt,'string',num2str(xrange(2)),'userdata',xrange(2))
else
    set([gui.aXwidthT,gui.aXwidth],'visible','off')
    set([gui.aXauto gui.aXlog],'enable','on')
end

if get(gui.aYauto,'value');
    set(gui.aAnalAx,'ylimmode','auto');
    set(gui.aYb,'string',num2str(min(get(gui.aAnalAx,'ylim'))),'enable','off')
    set(gui.aYt,'string',num2str(max(get(gui.aAnalAx,'ylim'))),'enable','off')
else
    set(gui.aAnalAx,'ylimmode','manual')
    yrange=sort([str2double(get(gui.aYb,'string')) str2double(get(gui.aYt,'string'))]);
    set(gui.aAnalAx,'ylim',yrange);
    set(gui.aYb,'string',num2str(yrange(1)),'userdata',yrange(1),'enable','on')
    set(gui.aYt,'string',num2str(yrange(2)),'userdata',yrange(1),'enable','on')
end

if get(gui.aYval,'value')==4|get(gui.aYval,'value')==5
    set(findobj('tag','gSS07threshAnalysis','userdata',1),'ydata',...
        [1 1]*get(gui.aTH1Val,'userdata'),'xdata',get(gui.aAnalAx,'xlim'))
    set(findobj('tag','gSS07threshAnalysis','userdata',2),'ydata',...
        [1 1]*get(gui.aTH2Val,'userdata'),'xdata',get(gui.aAnalAx,'xlim'))
end
```

```
%Update Histogram
if get(gui.aYhist,'value')
   set(gui.aAnalAx,'position',[0.05 0.1 0.55 0.9])
   set(gui.aAnalAx2,'visible','on')
   set(gui.aHistPatch,'visible','on')
   set(gui.aXt,'position',[0.52 0.01 0.08 0.08])
   [N X]=hist(ydat,length(gui.aHistPatch));
   for i=1:length(gui.aHistPatch)
     set(gui.aHistPatch(i),'xdata',[0 1 1 0]*N(i),'ydata',...
        [-.5 -.5 .5 .5]*mean(diff(X))+X(i))
   end
  set(gui.aAnalAx2,'ylim',get(gui.aAnalAx,'ylim'),'yscale',get(gui.aAnalAx,'yscale'))
else
   set(gui.aAnalAx,'position',[0.05 0.1 0.71 0.9])
   set(gui.aAnalAx2,'visible','off')
   set(gui.aHistPatch,'visible','off')
   set(gui.aXt,'position',[0.68 0.01 0.08 0.08])
end

if strcmpi(get(gui.aLink(3),'checked'),'on')
set(findobj('tag','gFreqThresh'),'visible','off','xdata',get(gui.aAnalAx,'xlim'))
set(findobj('tag','gEDenThresh'),'visible','off','xdata',get(gui.aAnalAx,'xlim'))
if get(gui.aYval,'value')==6
   set([gui.aEDenThresh gui.aFreqThresh],'visible','off')
   set(gui.aFreqThresh,'visible','on')
   set(findobj('tag','gFreqThresh'),'visible','on')
elseif get(gui.aYval,'value')==7
   set([gui.aEDenThresh gui.aFreqThresh],'visible','off')
   set(gui.aEDenThresh,'visible','on')
   set(findobj('tag','gEDenThresh'),'visible','on')
else
   set([gui.aEDenThresh gui.aFreqThresh],'visible','off')
end
else
   set([gui.aEDenThresh gui.aFreqThresh],'visible','off')
end
```

14. aRTCalc

This real-time parameter calculation function is the work-horse of the entire analysis component of the program. It handles signal conditioning, event detection, event extraction, and analysis parameter calculation.

The function must deal with data in sweep sized chunks and must retain information about sweeps that overlapped the ending edge of the previous sweep in order to extract information about the event in the subsequent sweep when the remainder of the required samples are available.

This function applies the band pass filters selected by the user and then detects events based on the lowest absolute amplitude threshold selected by the user. Events are detected and all analysis parameters are calculated. Once analysis is complete, further decimation of acquired signals is carried out based on a second amplitude threshold or on the other available parameter thresholds (peak frequency or energy density).

aRTCalc also carries out real time correlation of signals with a target source. Window widths are acquired from a target channel (with overlap handling code) and a running mean is carried out and displayed in the target axis. This function is called by the samples acquired function of the analog input object if real time analysis mode is selected by the user.

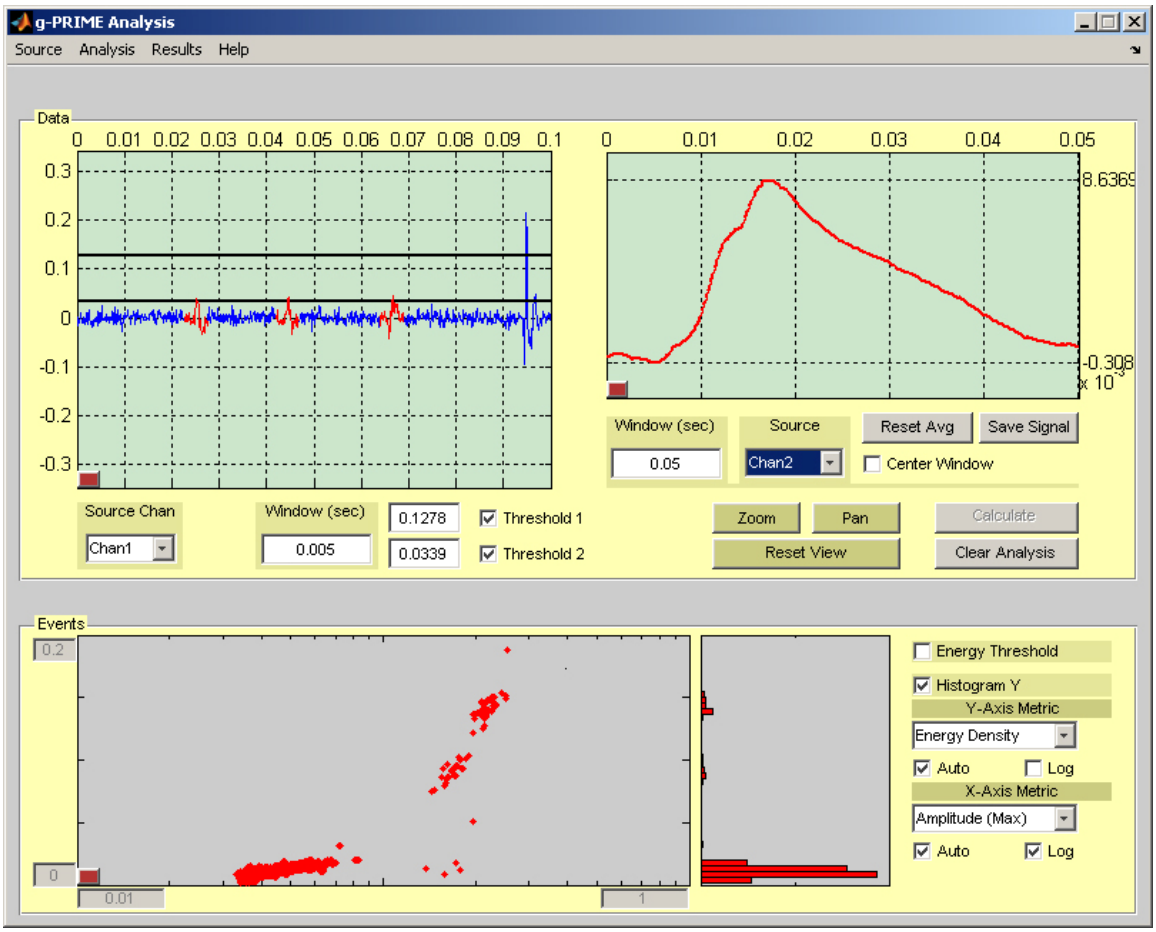

Figure A2.11 The fully active real time analysis window. aRTCalc handles the per-sweep event detection, signal correlation, and metric display update. This is the main work-horse of the real time analysis process and is called by the scope samples acquired function (SAF).

```
%_____________________________________________________________
% Function to handle real time analysis and display of raw data in the
% analysis window
function aRTCalc(varargin)
gui=get(findobj('tag','gSS07'),'userdata');
data=varargin{1};
time=varargin{2};
SelInd=varargin{3};

sRate=gui.ai.samplerate;

for i=1:4
   if strcmpi(get(gui.aWindowCent(i),'checked'),'on')
```

```
        WindWidth=i;
    end
end
wind=get(gui.aWindow,'userdata');
wind=ceil(wind*sRate/4)*4;
switch WindWidth
    case 1; prewind=0; postwind=wind;
    case 2; prewind=wind*.25; postwind=wind*.75;
    case 3; prewind=wind*.5; postwind=wind*.5;
    case 4; prewind=wind*.75; postwind=wind*.25;
end

hamwind=hamming(wind);
raw.data=data;
raw.time=time;
delete(findobj('tag','gSS07aTrace'))

%previous trace raw data
oldraw=get(gui.aSourcePl,'userdata');
if ~isstruct(oldraw); oldraw=[]; oldraw.time=inf; end

%previous Analysis Data
oldAnalysis=get(gui.aAnalAx,'userdata');
if ~isstruct(oldAnalysis)
    oldAnalysis=[];
    oldAnalysis.maxes=[]; oldAnalysis.maxamps=[]; oldAnalysis.mins=[];
    oldAnalysis.minamps=[]; oldAnalysis.fpeaks=[]; oldAnalysis.fsum=[];
    oldAnalysis.threshcross=[]; oldAnalysis.sRate=gui.ai.SampleRate;
    oldAnalysis.OverThresh=[];
end
if oldAnalysis.sRate~=gui.ai.SampleRate||raw.time(1)<oldraw.time(end)
    oldAnalysis.maxes=[]; oldAnalysis.maxamps=[]; oldAnalysis.mins=[];
    oldAnalysis.minamps=[]; oldAnalysis.fpeaks=[]; oldAnalysis.fsum=[];
    oldAnalysis.threshcross=[]; oldAnalysis.sRate=gui.ai.SampleRate;
    oldraw=[]; oldAnalysis.OverThresh=[];
end

%if filters were calculated on a different SR, recalculate them
LPfilt=get(gui.aLP,'userdata');
if isempty(LPfilt)
    aFiltCalc
    LPfilt=get(gui.aLP,'userdata');
end
if LPfilt.sRate~=gui.ai.samplerate; aFiltCalc; end
LPfilt=get(gui.aLP,'userdata');
```

```
HPfilt=get(gui.aHP,'userdata');

%Apply Filter(s)
if isstruct(oldraw)
   data=[oldraw.data; data];
if ~strcmpi(get(gui.aLPMenu(1),'checked'),'on')
   data=filter(LPfilt.b,LPfilt.a,data);
end
if ~strcmpi(get(gui.aHPMenu(1),'checked'),'on')
   data=filter(HPfilt.b,HPfilt.a,data);
end
   oldraw.data=data(1:(end-length(raw.data)));
   data=data((end-length(raw.data)+1):end);
end
avdat=mean(data);

%Calculate Previous threshold crosses to define a start point for the
%events in this sweep.
if ~isempty(oldAnalysis.OverThresh)
   %analysis.OverThresh is indexes in oldraw.data for threshold crosses
   minind=oldAnalysis.OverThresh-prewind;
   maxind=oldAnalysis.OverThresh+postwind-1;
   overlap=maxind>length(oldraw.data);
   if overlap
      tempdat=[oldraw.data(minind:end); data(1:(maxind-length(oldraw.data)))];
      temptime=[oldraw.time(minind:end); time(1:(maxind-length(oldraw.data)))];
   else
      tempdat=oldraw.data(minind:maxind);
      temptime=oldraw.time(minind:maxind);
   end
   %Do Analysis and then tack it onto the analysis vector
   overmax=find(max(tempdat)==tempdat,1,'first');
   overmaxamp=tempdat(overmax);
   overmin=find(min(tempdat)==tempdat,1,'first');
   overminamp=tempdat(overmin);

   overfsig=abs(fft((tempdat-mean(tempdat)).*hamwind));
   overfsig=overfsig(1:ceil(end/2));
   freq=linspace(0,sRate/2,length(overfsig));
   ffpeak=find(overfsig==max(overfsig),1,'first');
   overfpeak=freq(ffpeak);
   overfsum=sum(overfsig.^2)/length(overfsig);
   overtime=oldraw.time(oldAnalysis.OverThresh);

   %reject threshcross outside of 2 thresh range if active
```

```
discardindex=0;
if get(gui.aThresh1,'value')&get(gui.aThresh2,'value')
   %Double Thresh Mode
    thresh=[get(gui.aTH1Val,'userdata') get(gui.aTH2Val,'userdata')];
    [y,i]=max(abs(thresh));
    maxthresh=y*sign(thresh(i));
    if maxthresh>avdat
       discardindex=overmaxamp>maxthresh;
    else
       discardindex=overminamp<maxthresh;
    end
end

 if get(gui.aEDenThresh,'value')
   EDenLim=sort([mean(get(findobj('tag','gEDenThresh','userdata',1),'ydata')),...
       mean(get(findobj('tag','gEDenThresh','userdata',2),'ydata'))]);
    discardindex=discardindex|(overfsum<EDenLim(1)|overfsum>EDenLim(2));
 end
 if get(gui.aFreqThresh,'value')
   FreqLim=sort([mean(get(findobj('tag','gFreqThresh','userdata',1),'ydata')),...
       mean(get(findobj('tag','gFreqThresh','userdata',2),'ydata'))]);
    discardindex=discardindex|(overfpeak<FreqLim(1)|overfpeak>FreqLim(2));
 end

%tack overmax, overmaxamp, overmin, overminamp, overfpeak, overfsum,
%overtime onto the analysis list.
if discardindex==0
   oldAnalysis.maxes(end+1)=overmax;
   oldAnalysis.maxamps(end+1)=overmaxamp;
   oldAnalysis.mins(end+1)=overmin;
   oldAnalysis.minamps(end+1)=overminamp;
   oldAnalysis.fpeaks(end+1)=overfpeak;
   oldAnalysis.fsum(end+1)=overfsum;
   oldAnalysis.threshcross(end+1)=overtime;


minind=1;
maxind=oldAnalysis.OverThresh+postwind-1-length(oldraw.data);
if strcmpi(get(gui.aAnalysisMenu(5),'checked'),'on')
   line(minind:maxind,abs(data(minind:maxind))','tag','gSS07aTrace',...
      'color','r','parent',gui.aSourceAx);
else
   line(minind:maxind,data(minind:maxind)','tag','gSS07aTrace',...
      'color','r','parent',gui.aSourceAx);
end
```

```
    end
end


thresh=[get(gui.aTH1Val,'userdata') get(gui.aTH2Val,'userdata')];
threshon=[get(gui.aThresh1,'value') get(gui.aThresh2,'value')]==1;

if strcmpi(get(gui.aAnalysisMenu(5),'checked'),'on')
    analysisdata=data;
    data=abs(data);
end

if (get(gui.aThresh1,'value')|get(gui.aThresh2,'value'))==0
    %No threshold
    threshcross=[];
elseif xor(get(gui.aThresh1,'value'),get(gui.aThresh2,'value'))
    %single Threshold
    thresh=thresh(threshon);
    if thresh>=avdat
        threshcross=find(data>=thresh);
    else
        threshcross=find(data<=thresh);
    end
elseif get(gui.aThresh1,'value')&get(gui.aThresh2,'value')
    [y,i]=min(abs(thresh));
    minthresh=y*sign(thresh(i));
    if minthresh>=avdat
        threshcross=find(data>=minthresh);
    else
        threshcross=find(data<=minthresh);
    end
    %double Threshold
end

if strcmpi(get(gui.aAnalysisMenu(5),'checked'),'on')
    data=analysisdata;
end

%Detect events
OverThresh=[];
if ~isempty(threshcross)
    if ~isempty(oldAnalysis.threshcross)

threshcross(time(threshcross)<(oldAnalysis.threshcross(end)+postwind/sRate))=[];
    end
```

```
    threshcross(time(threshcross)<(prewind/sRate))=[];
end

j=1;
while j<=length(threshcross)
    threshcross((threshcross-threshcross(j)<postwind)&(threshcross-
threshcross(j)>0))=[];
    j=j+1;
end

fullThresh=[];
if ~isempty(threshcross)
    OverThresh=threshcross(threshcross>(length(data)-postwind));
    fullThresh=threshcross;
    threshcross(threshcross>(length(data)-postwind))=[];
end

maxes=zeros(1,length(threshcross));
mins=zeros(1,length(threshcross));
fpeaks=zeros(1,length(threshcross));
fsum=zeros(1,length(threshcross));
maxamps=zeros(1,length(threshcross));
minamps=zeros(1,length(threshcross));

for i=1:length(threshcross)
    if (threshcross(i)-prewind)>=1
        tempdata=data((1:wind)-prewind+threshcross(i));
        temptime=time((1:wind)-prewind+threshcross(i));
    else
        ind=threshcross(i)-prewind;
        tempdata=[oldraw.data((end+ind):end); data(1:(postwind-1+threshcross(i)))];
        temptime=[oldraw.time((end+ind):end); time(1:(postwind-1+threshcross(i)))];
    end
    %Interval Max
    maxes(i)=find(max(tempdata)==tempdata,1,'first');
    maxamps(i)=tempdata(maxes(i));
    maxes(i)=temptime(maxes(i));
    %Interval Min
    mins(i)=find(min(tempdata)==tempdata,1,'first');
    minamps(i)=tempdata(mins(i));
    mins(i)=temptime(mins(i));
    fsig=abs(fft((tempdata-mean(tempdata)).*hamwind));
    fsig=fsig(1:ceil(end/2));
    freq=linspace(0,sRate/2,length(fsig));
    ffpeak=find(fsig==max(fsig),1,'first');
```

```
    fpeaks(i)=freq(ffpeak);
    fsum(i)=sum(fsig.^2)/length(fsig);
end
%maxes= time of max amplitudes
%maxamps= peak value in window
%mins= time of min amplitudes
%minamps= min value in window
%fpeaks= peak frequency component
%fsum= sum of FFT components

%reject threshcross outside of 2 thresh range if active
discardindex=logical(zeros(1,length(maxes)));
if get(gui.aThresh1,'value')&get(gui.aThresh2,'value')
    %Double Thresh Mode
    thresh=[get(gui.aTH1Val,'userdata') get(gui.aTH2Val,'userdata')];
    [y,i]=max(abs(thresh));
    maxthresh=y*sign(thresh(i));
    if maxthresh>avdat
        discardindex=maxamps>maxthresh;
    else
        discardindex=minamps<maxthresh;
    end
end

 if get(gui.aEDenThresh,'value')
   EDenLim=sort([mean(get(findobj('tag','gEDenThresh','userdata',1),'ydata')),...
        mean(get(findobj('tag','gEDenThresh','userdata',2),'ydata'))]);
    discardindex=discardindex|(fsum<EDenLim(1)|fsum>EDenLim(2));
 end
 if get(gui.aFreqThresh,'value')
   FreqLim=sort([mean(get(findobj('tag','gFreqThresh','userdata',1),'ydata')),...
        mean(get(findobj('tag','gFreqThresh','userdata',2),'ydata'))]);
    discardindex=discardindex|(fpeaks<FreqLim(1)|fpeaks>FreqLim(2));
 end

    maxes(discardindex)=[];
    maxamps(discardindex)=[];
    mins(discardindex)=[];
    minamps(discardindex)=[];
    fpeaks(discardindex)=[];
    fsum(discardindex)=[];
    threshcross(discardindex)=[];

%Plot Red overlays on trace
for i=1:length(threshcross)
```

```
    minind=-prewind+threshcross(i);
    maxind=postwind-1+threshcross(i);
    if minind<1; minind=1; end
    if strcmpi(get(gui.aAnalysisMenu(5),'checked'),'on')
       line(minind:maxind,abs(data(minind:maxind))','tag','gSS07aTrace',...
          'color','r','parent',gui.aSourceAx);
    else
        line(minind:maxind,data(minind:maxind)','tag','gSS07aTrace',...
          'color','r','parent',gui.aSourceAx);
    end
end
if ~isempty(OverThresh)
    minind=-prewind+OverThresh;
    maxind=length(data);
    if strcmpi(get(gui.aAnalysisMenu(5),'checked'),'on')
       line(minind:maxind,abs(data(minind:maxind))','tag','gSS07aTrace',...
          'color','r','parent',gui.aSourceAx);
    else
        line(minind:maxind,data(minind:maxind)','tag','gSS07aTrace',...
          'color','r','parent',gui.aSourceAx);
    end
end

%Update Analysis Display
if ~isempty(oldAnalysis.maxes)
    analysis.maxes=[oldAnalysis.maxes(:); maxes(:)];
    analysis.maxamps=[oldAnalysis.maxamps(:); maxamps(:)];
    analysis.mins=[oldAnalysis.mins(:); mins(:)];
    analysis.minamps=[oldAnalysis.minamps(:); minamps(:)];
    analysis.fpeaks=[oldAnalysis.fpeaks(:); fpeaks(:)];
    analysis.fsum=[oldAnalysis.fsum(:); fsum(:)];
    analysis.threshcross=[oldAnalysis.threshcross(:); time(threshcross)];
    analysis.sRate=sRate;
    analysis.OverThresh=OverThresh;
else
    analysis.maxes=maxes(:);
    analysis.maxamps=maxamps(:);
    analysis.mins=mins(:);
    analysis.minamps=minamps(:);
    analysis.fpeaks=fpeaks(:);
    analysis.fsum=fsum(:);
    analysis.threshcross=time(threshcross);
    analysis.sRate=sRate;
    analysis.OverThresh=OverThresh;
end
```

```
if strcmpi(get(gui.aMasterDisplayA,'checked'),'off');
   N=1000;
   if length(analysis.maxes)>N
      analysis.maxes=analysis.maxes((end-N):end);
      analysis.maxamps=analysis.maxamps((end-N):end);
      analysis.mins=analysis.mins((end-N):end);
      analysis.minamps=analysis.minamps((end-N):end);
      analysis.fpeaks=analysis.fpeaks((end-N):end);
      analysis.fsum=analysis.fsum((end-N):end);
      analysis.threshcross=analysis.threshcross((end-N):end);
   end
end
set(gui.aAnalAx,'userdata',analysis)


% Real Time Triggered Average
if nargin==4
   RunningMean=get(gui.aCorrPl,'ydata');
   Corr.data=varargin{4};
   Corr.time=time;
   oldCorr=get(gui.aCorrPl,'userdata');
   corrInit=0;
   wind=round(get(gui.aCWindow,'userdata')*sRate);
   if (wind/sRate)>get(gui.UpdateRate,'userdata')
      set(gui.aCWindow,'string',get(gui.UpdateRate,'string'),...
         'userdata',get(gui.UpdateRate,'userdata'))
      wind=round(get(gui.UpdateRate,'userdata')*sRate);
   end
   wind=round(wind/2)*2;

   %Store First Sweep
   if isempty(oldCorr)
      oldCorr.data=Corr.data;
      oldCorr.time=time;
      oldCorr.num=0;
      oldCorr.Thresh=[];
      oldCorr.center=get(gui.aCenterWindow,'value');
      set(gui.aCorrPl,'userdata',oldCorr)
      corrInit=1;
   end
   if oldCorr.time(end)>time(1)||...
     oldCorr.center~=get(gui.aCenterWindow,'value');
      oldCorr.data=Corr.data;
      oldCorr.time=time;
```

```
    oldCorr.num=0;
    oldCorr.Thresh=[];
    oldCorr.center=get(gui.aCenterWindow,'value');
    set(gui.aCorrPl,'userdata',oldCorr)
    corrInit=1;
end
    Corr.num=oldCorr.num;
    Corr.Thresh=[];
    Corr.center=oldCorr.center;
if corrInit==0
    %     oldCorr.Thresh;
    %complete value from previous sweep
    if ~isempty(oldCorr.Thresh)
        for i=1:length(oldCorr.Thresh)
            if get(gui.aCenterWindow,'value')
                minind=oldCorr.Thresh(i)-wind/2;
                maxind=oldCorr.Thresh(i)+wind/2;
            else
                minind=oldCorr.Thresh(i);
                maxind=oldCorr.Thresh(i)+wind;
            end
            overlap=maxind>length(oldCorr.data);
            if overlap
                tempdat=[oldCorr.data(minind:end);...
                    Corr.data(1:(maxind-length(oldCorr.data)))];
            else
                tempdat=oldCorr.data(minind:maxind);
            end

            tempdat=tempdat-tempdat(1);

            if length(RunningMean)~=length(tempdat)
                RunningMean=tempdat'; Corr.num=1;
            else
                RunningMean=(RunningMean*Corr.num+tempdat')/(Corr.num+1);
                Corr.num=Corr.num+1;
            end
        end
    end

    % fullThresh is all threshold crosses (index) from this sweep
    CorrThreshcross=fullThresh;
    CorrThreshcross(time(CorrThreshcross)<(1+wind/2)/sRate)=[];
    if get(gui.aCenterWindow,'value')
```

```
Corr.Thresh=CorrThreshcross((CorrThreshcross+wind/2+1)>length(Corr.data));
    CorrThreshcross((CorrThreshcross+wind/2+1)>length(Corr.data))=[];
  else
    Corr.Thresh=CorrThreshcross((CorrThreshcross+wind)>length(Corr.data));
    CorrThreshcross((CorrThreshcross+wind)>length(Corr.data))=[];
  end

  if ~isempty(CorrThreshcross)
    for i=1:length(CorrThreshcross)
      if get(gui.aCenterWindow,'value')
        minind=CorrThreshcross(i)-wind/2;
        maxind=CorrThreshcross(i)+wind/2;
        if minind<1
          tempdat=[oldCorr.data((end+minind):end);...
            Corr.data(1:maxind)];
        else
          tempdat=Corr.data(minind:maxind);
        end
        tempdat=tempdat-tempdat(wind/2);
      else
        minind=CorrThreshcross(i);
        maxind=CorrThreshcross(i)+wind;
        tempdat=Corr.data(minind:maxind);
        tempdat=tempdat-tempdat(1);
      end

      if length(RunningMean)~=length(tempdat)
        RunningMean=tempdat'; Corr.num=1;
      else
        RunningMean=(RunningMean*Corr.num+tempdat')/(Corr.num+1);
        Corr.num=Corr.num+1;
      end
    end
  end


  if get(gui.aCenterWindow,'value')
    set(gui.aCorrPl,'userdata',Corr,'ydata',RunningMean,...
      'xdata',(1:length(RunningMean))/gui.ai.samplerate-...
      length(RunningMean)/(2*gui.ai.samplerate))
    set(gui.aCorrAx,'xlim',[-wind/(2*gui.ai.samplerate)...
      wind/(2*gui.ai.samplerate)],'ylimmode','auto',...
      'ytick',sort([min(RunningMean) max(RunningMean)]))
```

```
        else
            set(gui.aCorrPl,'userdata',Corr,'ydata',RunningMean,...
              'xdata',(1:length(RunningMean))/gui.ai.samplerate)
            set(gui.aCorrAx,'xlim',[0 wind/gui.ai.samplerate],'ylimmode','auto',...
               'ytick',sort([min(RunningMean) max(RunningMean)+10^-7]))
        end
    end
end
 if strcmpi(get(gui.aAnalysisMenu(5),'checked'),'on')
    data=abs(data);
end

set(gui.aThresh1Pl,'xdata',get(gui.aSourceAx,'xlim'),'ydata',...
    [1 1]*str2double(get(gui.aTH1Val,'string')))
set(gui.aThresh2Pl,'xdata',get(gui.aSourceAx,'xlim'),'ydata',...
    [1 1]*str2double(get(gui.aTH2Val,'string')))
set(gui.aTitle,'string',[popupstr(gui.ChanList),'  -  ',num2str(gui.ai.SampleRate),...
    'Hz - ',datestr(now)],'fontsize',.5)
set(gui.aSourcePl,'xdata',1:length(data),'ydata',data,'userdata',raw);
% set(gui.aSourceAx,'ylim',get(gui.ax(SelInd),'ylim'))
span=str2double(popupstr(gui.Refresh)); if span>1; span=1; end
if get(gui.aZoom,'value')==0;
    set(gui.aSourceAx,'xlim',[1 length(data)],...
        'xtick',linspace(1,length(data),11),'xticklabel',...
        round(linspace(0,span,length(get(gui.aSourceAx,'xtick')))*1000)/1000);
end
 aChangeAnalysisDisplay
```

15. aSetPolyRegion

This function illustrates how I have implemented the drawing of an arbitrary polygon in an axis to select points. As with the threshold dragging, this function utilizes the windowbuttonmotionfcn property of the analysis window. If the user extracts a square, only one point after the initial "buttondown" event is required. For polygons, the function is called multiple times until the user right-clicks the mouse in the analysis axis.

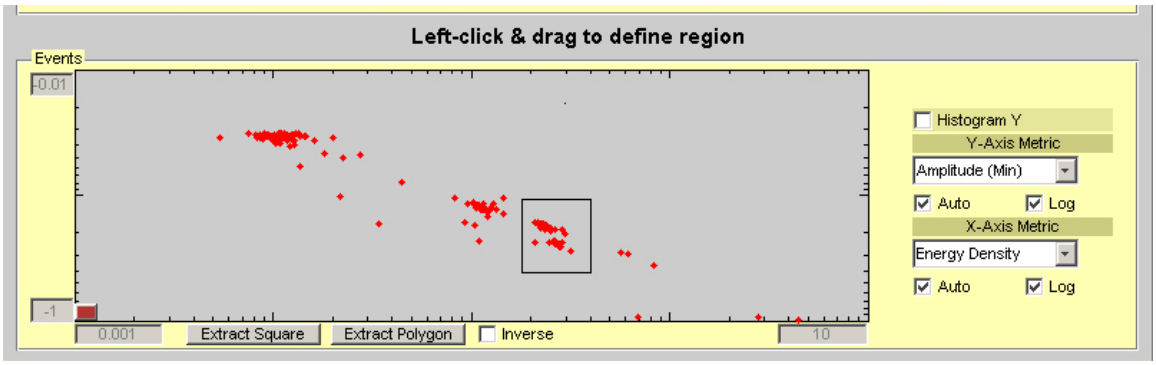

Figure A2.12 A polygon actively drawn on the analysis window to select a cluster of points for offline analysis, grooming, and extraction.

```
%_____________________________________________________________________
%Draw Points for polygon
function aSetPolyRegion(shape)
gui=get(findobj('tag','gSS07'),'userdata');
points=get(gui.aPanel,'userdata');
newpoint=get(gui.aAnalAx,'currentpoint');
clicktype=get(gui.aFig,'SelectionType');

%if right click, link to first point (error on only 1 line)
if ~strcmpi(clicktype,'normal')
   set([gui.aAnalAx gui.aAnalPl],'buttondownfcn','');
   if length(points)<2; delete(points); set(gui.aDirections,'string',''); return; end
   switch shape
      case 'Polygon'
         aDrawSubEvents
   end
   return
end

if strcmpi(shape,'Polygon')
   set(gui.aDirections,'string',...
      'Left-click and hold to draw side.  Right-Click to complete.')
end

%Draw initial line from this point to last point
switch shape
   case 'Square'
      %draw 4 lines
      for i=1:4
         points(i)=line([1 1]*newpoint(1,1),[1 1]*newpoint(1,2),'color','k');
      end
   case 'Polygon'
```

```
        if isempty(points)
           points(1)=line([1 1]*newpoint(1,1),[1 1]*newpoint(1,2),'color','k');
           x=get(points(end),'xdata'); y=get(points(end),'ydata');
        else
           x=get(points(end),'xdata'); y=get(points(end),'ydata');
           points(end+1)=line([x(2) newpoint(1,1)],[y(2) newpoint(1,2)],'color','k');
        end
end
set(points,'buttondownfcn',['gprime("aSetPolyRegion",'',shape,'')']);
%Set figure function to continue drawing while button down
set(gui.aFig,'windowbuttonmotionfcn','gprime("PolyTraceDraw")')

while ~isempty(get(gui.aFig,'windowbuttonmotionfcn'))
a=get(gui.aAnalAx,'currentpoint');
switch shape
   case 'Square'
      %update 4 lines anchored at newpoint
      set(points(1),'xdata',[newpoint(1,1) a(1,1)],'ydata',[1 1]*newpoint(1,2))
      set(points(2),'xdata',[1 1]*a(1,1),'ydata',[newpoint(1,2) a(1,2)])
      set(points(3),'xdata',[a(1,1) newpoint(1,1)],'ydata',[1 1]*a(1,2))
      set(points(4),'xdata',[1 1]*newpoint(1,1),'ydata',[a(1,2) newpoint(1,2)])
   case 'Polygon'
      set(points(end),'xdata',[x(2) a(1,1)],'ydata',[y(2) a(1,2)])
end
drawnow
end

set(gui.aPanel,'userdata',points);
switch shape
   case 'Square'
      %Call analysis function
      aDrawSubEvents
      set(gui.aAnalAx,'buttondownfcn','');
   case 'Polygon'
end

function PolyTraceDraw
```

16. aDrawSubEvents

Once a polygon is defined, the enclosed data points are extracted and displayed in a separate figure. This allows the user to groom the data set and save publication quality graphics containing their analysis results. The figure created by the function contains menus that allow for display of the subset and for the user to save both the analysis results and the raw traces to a file.

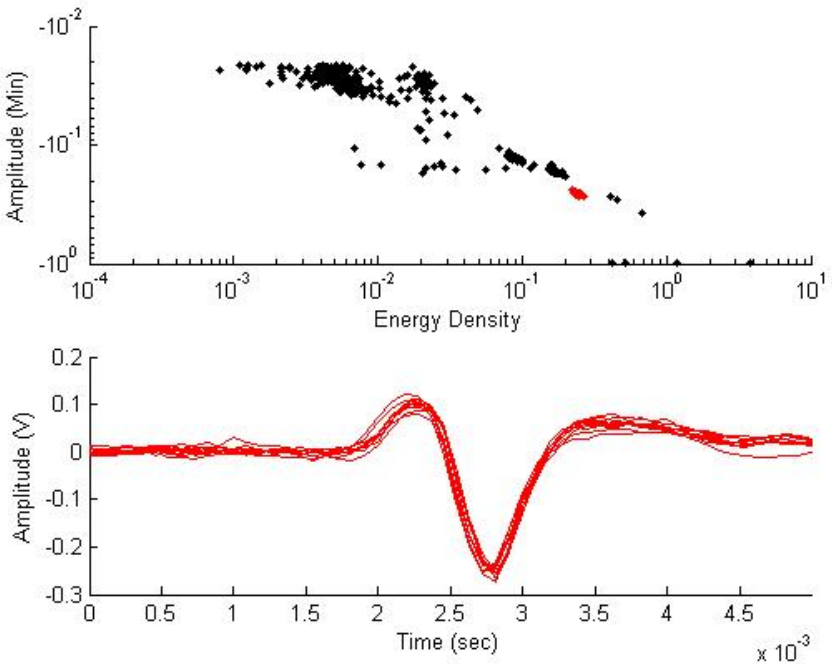

Figure A2.13 A single selected cluster of points in a data set. The user may visualize the subset of events selected in the total analysis input space. The user has access to raw trace display with only the subset displayed and rate plots for the individual subgroup only.

```
%_____________________________________________________________________
%Extract Subevents and display graph
function aDrawSubEvents
gui=get(findobj('tag','gSS07'),'userdata');
%Determine Analysis Parameters and find which are located w/in polygon
set(gui.aDirections,'string','')
analysis=get(gui.aAnalAx,'userdata');
sRate=get(gui.aMasterCalc,'userdata');
wind=get(gui.aWindow,'userdata');
```

```
wind=ceil(wind*sRate/2)*2;
points=get(gui.aPanel,'userdata');
endX=get(points(end),'xdata');
endY=get(points(end),'ydata');
oneX=get(points(1),'xdata');
oneY=get(points(1),'ydata');
points(end+1)=line([endX(2) oneX(1)],[endY(2) oneY(1)],'color','k');
drawnow

for i=1:length(points)
   tempX=get(points(i),'xdata');
   xv(i)=tempX(1);
   tempY=get(points(i),'ydata');
   yv(i)=tempY(1);
end


X=get(gui.aAnalPl,'xdata');
Y=get(gui.aAnalPl,'ydata');

if length(X)~=length(analysis.threshcross)
  %Rate/Interval is a differential so misses the first value
  X=[nan, X];
  Y=[nan, Y];
end
inside=inpolygon(X,Y,xv,yv);
if get(gui.aPolyInverse,'value'); inside=~inside; end
intimes=round(analysis.threshcross(inside)*sRate);
delete(points)

data=get(gui.aSourcePl,'ydata');
time=get(gui.aSourcePl,'xdata');


thisfig=figure('numbertitle','off','name','Analysis Subset');
set(thisfig,'keypressfcn','gprime("aDelSelectTrace")')

analysis.inside=inside;
inds=find(inside==1);
set(gcf,'userdata',analysis)

ax=axes('position',[0 0 1 1],'tag','source');
line(time,data,'parent',ax,'color','k')
spikes=analysis.threshcross(inside);
for i=1:length(spikes)
```

```
    ind=(time>(spikes(i)-wind/(2*sRate)))&(time<(spikes(i)+wind/(2*sRate)));
    line(time(ind),data(ind),'color','r','parent',ax,'userdata',inds(i),'tag','Gspikes');
end
xlabel('Time (sec)')
ylabel('Amplitude (V)')
set(get(ax,'children'),'visible','off')
set(ax,'visible','off')

ax=axes('position',[0 0 1 1],'tag','analysis','buttondownfcn',...
    'gprime("aResetTracesSubShow")');
for i=1:length(inds)
    line(X(inds(i)),Y(inds(i)),'color','r','marker','.','linestyle','none',...
        'parent',ax,'userdata',inds(i),'buttondownfcn',...
        ['gprime("aSubSingleShow","',num2str(inds(i)),'");'])
end
line(X(~inside),Y(~inside),'color','k','marker','.','linestyle','none','parent',ax)
set(ax,'xscale',get(gui.aAnalAx,'xscale'),'yscale',get(gui.aAnalAx,'yscale'))
set(ax,'xlim',get(gui.aAnalAx,'xlim'),'ylim',get(gui.aAnalAx,'ylim'))
xlabel(popupstr(gui.aXval))
ylabel(popupstr(gui.aYval))
set(get(ax,'children'),'visible','off')
set(ax,'visible','off')

ax=axes('position',[0 0 1 1],'tag','events','buttondownfcn',...
    'gprime("aResetTracesSubShow")');
for i=1:length(intimes)
    line(((1:wind)-1)/sRate,data(((1-wind/2):(wind/2))+intimes(i)),'color','r',...
        'userdata',inds(i),'buttondownfcn',...
        ['gprime("aSubSingleShow","',num2str(inds(i)),'");']);
end
set(ax,'xlim',[0 wind-1]/sRate)
xlabel('Time (sec)')
ylabel('Amplitude (V)')
set(get(ax,'children'),'visible','off')
set(ax,'visible','off')

ax=axes('position',[0 0 1 1],'tag','rate');
line(spikes(2:end),1./diff(spikes),'marker','.','linestyle','none','color','b')
ylabel('Inter-Spike Rate (Hz)')
xlabel('Event Time')
set(get(ax,'children'),'visible','off')
set(ax,'visible','off')


ax=axes('position',[0 0 1 1],'tag','interval');
```

```
line(spikes(2:end),diff(spikes),'marker','.','linestyle','none','color','b')
ylabel('Inter-Spike Rate (Hz)')
xlabel('Event Time')
set(get(ax,'children'),'visible','off')
set(ax,'visible','off')

h=uimenu('label','Options','foregroundcolor','b','tag','gAMen');
uimenu('label','Save Analysis Values (*.txt)','parent',h,...
   'callback','gprime("aSaveSubset")');
uimenu('label','Save Raw Traces (*.txt)','parent',h,...
   'callback','gprime("aSaveRawSubset")');
uimenu('label','Display Source','checked','off','separator','on','parent',h,...
   'callback','gprime("aDispSub")');
uimenu('label','Display Analysis','checked','on','parent',h,...
   'callback','gprime("aDispSub")');
uimenu('label','Display Events','checked','on','parent',h,...
   'callback','gprime("aDispSub")');
uimenu('label','Display Event Rate','checked','off','parent',h,...
   'callback','gprime("aDispSub")');
uimenu('label','Display Event Interval','checked','off','parent',h,...
   'callback','gprime("aDispSub")');
aDispSub(gcf)
```

17. aSubSingleShow

This function handles probably the most intuitive and powerful component of the analysis interface. Clusters may be groomed visually to make distinctions where parameters in the two dimensional analysis space failed. The "Buttondownfcn" of each selected event are set to link their entry in the data set. When an event is clicked on in the raw trace view or in the analysis metric space, the event highlights blue. When the user hits the "delete" key (key code 127), the event is extracted from the data set.

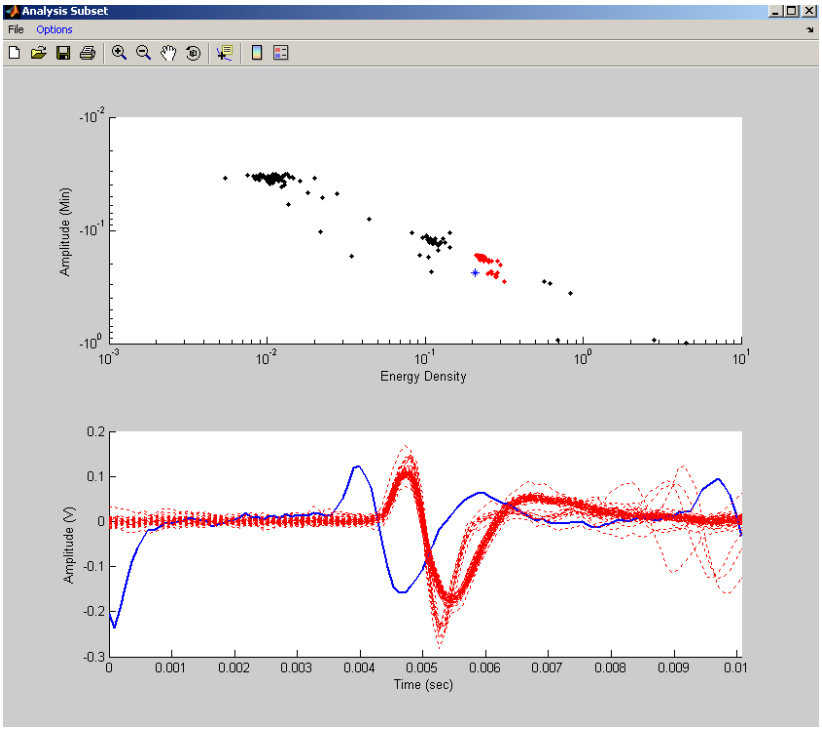

Figure A2.14 aSubSingleShow handles the selection and removal of individual traces in a selected subset of points from an offline analysis sub-set of detected events.

```
%Highlight a trace when a selected dot is clicked on
function aSubSingleShow(ind)
ind=str2double(ind);
analysisAx=findobj('parent',gcf,'type','axes','tag','analysis');
traceax=findobj('parent',gcf,'type','axes','tag','events');
rawax=findobj('parent',gcf,'type','axes','tag','source');
set(findobj('tag','Gspikes'),'color','r')
events=get(traceax,'children');
target=findobj(events,'userdata',ind);
aDot=findobj('parent',analysisAx,'markeredgecolor','b');
set(aDot,'markeredgecolor','r')
aDot=findobj('parent',analysisAx,'userdata',ind);
set(events,'linewidth',1,'color','r','visible','on','linestyle',':')
set(target,'linewidth',2,'color','b','linestyle','-')
set(aDot,'markeredgecolor','b','marker','*')
set(findobj('parent',rawax,'userdata',get(aDot,'userdata')),'color','b')

%_____
%Reset event traces when clicking on the analysis axis
function aResetTracesSubShow
traceax=findobj('parent',gcf,'type','axes','tag','events');
```

```
set(get(traceax,'children'),'linestyle','-','linewidth',1,'color','r')
analysisAx=findobj('parent',gcf,'type','axes','tag','analysis');
aDot=findobj('parent',analysisAx,'markeredgecolor','b');
set(aDot,'markeredgecolor','r','marker','.')
set(findobj('tag','Gspikes'),'color','r')

%_____________________________________________________________________
%if delete is pressed, delete a selected trace from the grouping
function aDelSelectTrace
if double(get(gcf,'CurrentCharacter'))~=127; return; end

analysisAx=findobj('parent',gcf,'type','axes','tag','analysis');
traceax=findobj('parent',gcf,'type','axes','tag','events');
rawax=findobj('parent',gcf,'type','axes','tag','source');
selTrace=findobj('parent',traceax,'color','b');
selDot=findobj('parent',analysisAx,'markeredgecolor','b');
if isempty(selTrace); return; end

analysis=get(gcf,'userdata');
analysis.inside(get(selDot,'userdata'))=0;
delete(findobj('parent',rawax,'userdata',get(selDot,'userdata')))
set(gcf,'userdata',analysis)
delete(selTrace)
set(selDot,'MarkerEdgeColor','k','buttondownfcn','','marker','.')
aResetTracesSubShow

spikes=analysis.threshcross(analysis.inside);

rateax=findobj('parent',gcf,'type','axes','tag','rate');
delete(get(rateax,'children'))
line(spikes(2:end),1./diff(spikes),'marker','.','linestyle','none',...
   'color','b','parent',rateax,'visible',get(rateax,'visible'))

intax=findobj('parent',gcf,'type','axes','tag','interval');
delete(get(intax,'children'))
line(spikes(2:end),diff(spikes),'marker','.','linestyle','none',...
   'color','b','parent',intax,'visible',get(intax,'visible'))
```

18. LoadSubEvents

Like with the online correlation function carried out in "aRTCalc," and offline correlation may be carried out with a data set. Values extracted through subset selection and grooming may be saved and then reloaded. The reloaded values are used to trigger a window in the currently loaded trace (correlating time events). Since all traces are available, the median may be compared to the mean value of the data set to determine the statistical quality of the resulting correlation.

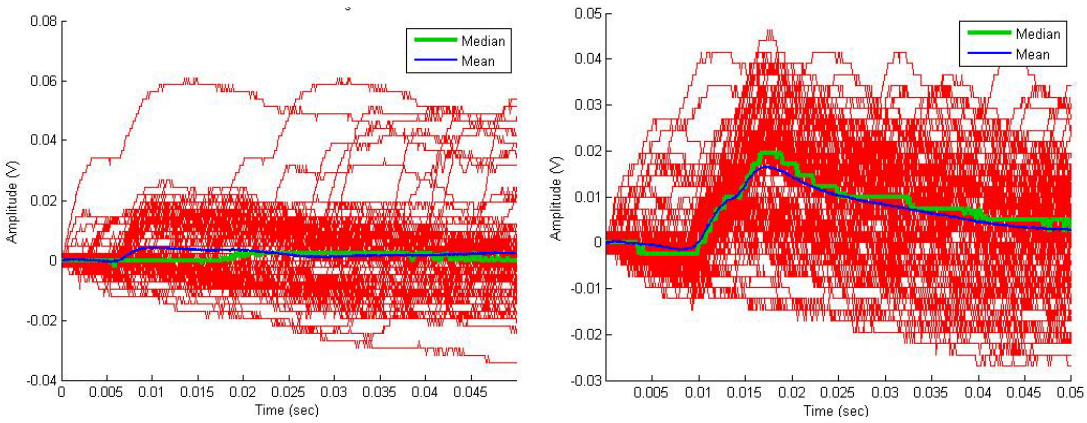

Figure A2.15 Two examples of offline correlated data. Action potentials from one channel were selected using the aSubSingleShow functionality and loaded back using "LoadSubEvents" to visualize a correlation (if any) between the data sets. Event series at left indicates low correlation (i.e. significant median divergence from mean) while the event series at right shows a strong correlation between the two channels.

```
%_____________________________________________________________________
%Load a stored set of subevents (text file) and extract the results from
%the window width in front of the event on a new data set
function LoadSubEvents
gui=get(findobj('tag','gSS07'),'userdata');

[fname2, pname2] = uigetfile('*.txt', 'Load Analysis Subset',...
    get(gui.DataMenu(1),'userdata'));
if isequal(fname2,0)||isequal(pname2,0)
  return
end
```

```
set(gui.DataMenu(1),'userdata',pname2)

wind=inputdlg('Post Event Window Width? (sec)','Enter a Value',1,{'0.05'});
wind=str2double(wind);
if isnan(wind)||isempty(wind); return; end

centerq=questdlg('Location of Window Center on Event','Window Center',...
   'After','Center','After');
if strcmpi(centerq,'after')|strcmpi(centerq,'center')
   centerq=strcmpi(centerq,'center');
else
   return
end

traceq=questdlg('Display All Traces in Background?','Display Raw Traces?','Yes');
if strcmpi(traceq,'yes')|strcmpi(traceq,'no')
   traceq=strcmpi(traceq,'yes');
else
   return
end

temp=load([pname2,fname2]);
threshcross=temp(:,1);
data=get(gui.aSourcePl,'ydata');
if isempty(data)||isempty(threshcross); return; end

sRate=get(gui.aMasterCalc,'userdata');
wind=ceil(wind*sRate/2)*2;
threshcross(threshcross>((length(data)/sRate)-wind/sRate))=[];

delete(findobj('tag','gSS07aTrace'))
line(threshcross,data(round(threshcross*sRate)),'color','r','tag','gSS07aTrace',...
   'linestyle','none','marker','*','markersize',10,'parent',gui.aSourceAx);
figure('numbertitle','off','name','Analysis Values Correlation')
sumd=zeros(wind,length(threshcross));

for i=1:length(threshcross)
   index=round((1:wind)+threshcross(i)*sRate-centerq*wind/2);
   if traceq
      line((-centerq*wind/2+(1:length(index)))/sRate,data(index)-
data(min(index)+centerq*wind/2),...
         'color','r');
   end
   sumd(:,i)=data(index)-data(min(index));
end
```

```
a1=gca;
a2=axes;
meansumd=mean(sumd,2);
mediansumd=median(sumd,2);
line((-centerq*wind/2+(1:length(index)))/sRate,mediansumd,'color',[0 .8
0],'linewidth',3)
line((-centerq*wind/2+(1:length(index)))/sRate,meansumd,'color','b','linewidth',2)

legend('Median','Mean')

set([a1 a2],'xlim',([1 wind]-centerq*wind/2)/sRate)
if traceq
   set(a2,'ylim',get(a1,'ylim'),'color','none')
end
thistitle=get(gui.aTitle,'string');
ind=strfind(thistitle,'\');
title([thistitle((ind(end)+1):end),' Correlated w/ ',fname2])
xlabel('Time (sec)')
ylabel('Amplitude (V)')
```

REFERENCES

[1] Cade W, "Acoustically Orienting Parasitoids: Fly Phonotaxis to Cricket Song." *Science* 190:1312-1313, 1975

[2] Doherty JA. Phonotaxis in the cricket, "Gryllus bimaculatus De Geer: Comparisons of choice and no-choice paradigms." *J Comp Physiol A* 157:279-289., 1985

[3] Doherty JA, Pires A. "A new microcomputer-based method for measuring walking phonotaxis in field crickets (Gryllidae)." *J Exp Biol* 130:425-432., 1987

[4] Haggerty H.S., Lusted H.S., "Histological reaction to polyimide films in the cochlea." *Acta Otolaryngol*, 107(1-2):13-22, 1989

[5] Hedwig B, Poulet JF. "Complex auditory behaviour emerges from simple reactive steering." *Nature* 430:781-785., 2004

[6] Hoy RR, Paul RC. "Genetic control of song specificity in crickets." *Science*180:82-83., 1973

[7] James C.D., Spence A.J.H., Dowell-Mesfin, N.M., Hussain R.J, Smith K.L., Craighead H.G., Isaacson M.S., Shain W., Turner J.N, "Extracellular Recordings from Patterned Neuronal Networks Using Planar Microelectrode Arrays." *IEEE Trans. Biomed. Eng.* 51(9):1640-1648, 2004

[8] Kennedy D., and Takeda K., "Reflex Control of Abdominal Flexor Muscles in the Crayfish: II. The Tonic System," *J. Exp. Biol*., Vol. 43, pp. 229-246, 1965.

[9] Kramer E. "The orientation of walking honeybees in odour fields with small concentration gradients." *Physiological Entomology* 1:27-37., 1976

[10] Mason AC, Oshinsky ML, Hoy RR. "Hyperacute directional hearing in a microscale auditory system." *Nature* 410:686-690., 2001

[11] McCarthy BJ, McMillan DL. "The role of the muscle receptor organ in the control of abdominal extension in the crayfish *Cherax destructor*," *J. Exp. Biol.* 198:2253-2259, 1995

[12] Muller P, Robert D, "A Shot in the Dark: The Silent Quest of a Free-Flying Phonotactic Fly." *J. Exp Biology* 204:1039-1052, 2001

[13] Murphey RK, Zaretsky MD. "Orientation to calling song by female crickets, Scapsipedus marginatus (Gryllidae)." *J Exp Biol* 56:335-352., 1972

[14] Naples G.G., Mortimer J.T., Scheiner A., and Sweeney J.D., "A Spiral Nerve Cuff Electrode for Peripheral Nerve Stimulation," *IEEE Trans. Biomed. Eng.*, Vol. 35, no. 11, pp. 905-916, 1988.

[15] Oshinsky M.L., Hoy R.R., "Physiology of the Auditory Afferents in an Acoustic Parasitoid Fly." *J. Neurosci* 22(16):7254-7263, 2002

[16] Pearce J, Govind CK. "Remodeling of the Proximal Segment of Crayfish Motor Nerves Following Transection," *J Comparative Neurology*. 450:61-72., 2002

[17] Pearce J, Lnenicka GA, Govind CK. "Regenerating Crayfish Motor Axons Assimilate Glial Cells and Sprout in Cultured Explants," *J Comparative Neurology*. 464:449-462., 2003

[18] Pires A, Hoy RR. "Temperature coupling in cricket acoustic communication. I. Field and laboratory studies of temperature effects on calling song production and recognition in Gryllus firmus." *J Comp Physiol* [A] 171:69-78., 1992a

[19] Pires A, Hoy RR. "Temperature coupling in cricket acoustic communication. II. Localization of temperature effects on song production and recognition networks in Gryllus firmus." *J Comp Physiol* [A] 171:79-92., 1992b

[20] Pollack G, Huber F, Weber T. "Frequency and temporal pattern-dependent phonotaxis of crickets (Teleogryllus oceanicus) during tethered flight and compensated walking." *J Comp Physiol A* 154:13-26., 1984

[21] Pollack GS, Hoy RR. "Temporal pattern as a cue for species-specific calling song recognition in crickets." *Science* 204:429-432., 1979

[22] Popović D.B., Stein R.B., Jovanović K.Lj., Dai R., and Armstrong W.W., "Sensory Nerve Recording for Closed-Loop Control to Restore Motor Functions," *IEEE Trans. Biomed. Eng.*, Vol 40, no. 10, pp. 1024-1031, 1993.

[23] Ramsauer N, Robert D. "Free-flight phonotaxis in a parasitoid fly: behavioural thresholds, relative attraction and susceptibility to noise." *Naturwissenschaften* 87:315-319., 2000

[24] Richardson Jr. R.R., Miller J.A., and Reichert W.M., "Polyimides as biomaterials: preliminary biocompatibility testing," *Biomaterials*, Vol.14, no. 8, pp. 627-635, 1993.

[25] Robert D., Amoroso J., Hoy R.R., "The Evolutionary Convergence of Hearing in a Parasitoid Fly and Its Cricket Host." *Science*, 258 (5085):1135-1137, 1992

[26] Robert D., Miles R.N., Hoy R.R., "Tympanal Mechanics in the Parasitoid fly *Ormia ochracea*: Intertympanal coupling during mechanical vibration." *J. Comp. Physiol A* 183:443-452, 1998

[27] Robert D., Miles R.N., Hoy R.R., "Tympanal Hearing in the Sarcophagid Parasitoid Fly *Emblemasoma* SP.: The Biomechanics of Directional Hearing." *J. Exp Biology* 202:1865-1876, 1999

[28] Robert D, Willi U, "The Histological Architecture of the Auditory Organs in the Parasitoid Fly *Ormia ochracea*." *Cell Tissue Res* 301:447-457, 2000

[29] Rodríguez F.J., Ceballos D., Schüttler M., Valero A., Valderrama E., Stieglitz T., and Navarro X., "Polyimide cuff electrodes for peripheral nerve stimulation," *J. Neurosci. Methods*, Vol. 98, pp. 105-118, 2000

[30] Rousche P.J., Pellinen D.S., Pivin Jr. D.P., Williams J.C., Vetter R.J., and Kipke D.R., "Flexible Polyimide-Based Intracortical Electrode Arrays with Bioactive Capability," *IEEE Trans. Biomed. Eng.*, Vol. 48, no. 3, pp. 361-371, 2001.

[31] Seo J., Kim S.J., Chung H., Kim E.T., Yu, H.G., Yu Y.S., "Biocompatibility of Polyimide Microelectrode Array for Retinal Stimulation." *Materials Science and Engineering: C*, 24(1-2):185-189, 2004

[32] Shamma-Donoghue S.A., May G.A., Cotter N.E., White R.L., Simmons F.B., "Thin-Film Multielectrode Arrays for a Cochlear Prosthesis." *IEEE Trans. Electron Devices* ED-29(1):136-144 (1982)

[33] Spence A.J., Hoy R.R., Isaacson M.S., "A Micromachined Silicon Multielectrode for Multiunit Recording." *J Neurosci Methods* 126(3):119-26, 2003

[34] Spence A.J., Neeves KB, Murphy D, Sponberq S, Land BR, Hoy RR, Isaacson M.S., "Flexible Multielectrodes can Resolve Multiple Muscles in an Insect Appendage." *J. Neurosci Methods* 159(1):116-24, 2007

[35] Swerup C. "On the ionic mechanisms of adaptation in an isolated mechanoreceptor –an electrophysiological study," *Acta Physiol Scand Suppl.* 520:1-43., 1983

[36] Tarler M.D. and Mortimer J.T., "Selective and Independent Activation of Four Motor Fascicles Using a Four Contact Nerve-Cuff Electrode," *IEEE Trans. Neur. Sys. And Rehab. Eng.*, Vol. 12, no. 2, pp. 251-257, 2004.

[37] van der Puije P.D., Pon C.R., Robillard H., "Cylindrical Cochlear Electrode Array for Use in Humans." *Ann Otol Rhinol Laryngol*, 98(6):466-71, 1989

[38] van Harreveld A., "A physiological solution for fresh-water crustaceans." *Proc. Soc. Exp. Biol*., (N.Y.) Vol. 34, 428-432 (1936)

[39] Vélez SJ, Wyman RJ. "Synaptic Connectivity in a Crayfish Neuromuscular System. I. Gradient of Innervation and Synaptic Strength," *J Neurophys.* 41(1):75-84., 1978

[40] Vélez SJ, Wyman RJ. "Synaptic Connectivity in a Crayfish Neuromuscular System. II. Nerve-Muscle Matching and Nerve Branching Patterns," *J Neurophys.* 41(1):85-96., 1978

[41] Veraart C., Raftopoulos C., Mortimer J.T., Delbeke J., Pins D., Michaux G., Vanlierde A., Parrini S., Wanet-Defalque M., "Visual sensations produced by optic nerve stimulation using an implanted self-sizing spiral cuff electrode," *Brain Res.*, Vol 813, pp. 181-186, 1998

[42] Walker TJ, Wineriter SA. "Rearing phonotactic parasitoid flies (Diptera: Tachinidae, Orminii, Ormia spp.)." *Entomophaga* 35:621-632., 1990

[43] Weber T, Thorson J, Huber F. "Auditory behavior of the cricket. I. Dynamics of compensated walking and discrimination paradigms on the Kramer treadmill." *J Comp Physiol A* 141:215-232., 1981

[44] Wine J.J., Mittenthal J.E., and Kennedy D., "The Structure of Tonic Flexor Motoneurons in Crayfish Abdominal Ganglia," *J. comp. Physiol.*, Vol. 93, pp 315-335, 1974.

[45] Wyttenbach RA, Johnson BR, Hoy RR. "Crawdad: A CD-ROM Lab Manual for Neurophysiology," Sinauer Associates Inc, Sunderland, MA, 2002

[46] Yoo P.B. and Durand D.M., "Selective Recording of the Canine Hypoglossal Nerve Using a Multicontact Flat Interface Nerve Electrode," *IEEE Trans. Biomed. Eng.*, Vol. 52, no. 8, pp. 1461-1469, 2005.