Robust, Adaptive Stimulus Artifact Cancellation Enabling Fully Bidirectional Neural Interfaces

John Patrick Uehlin

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Electrical Engineering

University of Washington

2017

Committee:

Jacques Christophe Rudell, Chair

Visvesh Sathe

Program Authorized to Offer Degree:

Electrical Engineering

University of Washington

Abstract

Robust, Adaptive Stimulus Artifact Cancellation Enabling Fully Bidirectional Neural Interfaces

John Patrick Uehlin

Chair of the Supervisory Committee:
Jacques Christophe Rudell
Department of Electrical Engineering

This work explores the design and implementation of a neural stimulus artifact cancellation system that enables reconfigurable bidirectional interfaces by allowing simultaneous electrical stimulation and electrical recording in the same region of neural tissue. This system is designed to be compatible with any stimulator or recording front end, and its efficacy is demonstrated and challenged by implementation with an H-Bridge stimulator and highly multiplexed mixed-mode feedback recording front-end. The canceller topology is adaptive, learning the stimulus artifact and tracking changes in stimulation or neural tissue characteristics. The adaptation algorithm and canceller topology is highly scalable to any length of stimulation artifact or number of simultaneous recording channels. The canceller has been implemented in CMOS and further prototyped on an FPGA. Real-time system validation has been performed with separate recording and stimulator chips, demonstrating compatibility with various front-end topologies. Voltage suppression capabilities are demonstrated up to 44dB, with signal-to-noise ratio improvements of up to 66dB possible by bringing the recording channel back into its linear range.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would first like to thank Dr. Jacques Christophe Rudell for the guidance and encouragement he has given over the course of my research. Dr. Rudell has been a consistent influence for me to better myself and the quality of my work. Acceptance into the Future Analog Systems and Technologies gave me the opportunity to research neural interface circuits, an exciting and challenging field, and it has opened up countless opportunities for academic and industry exposure. I would also like to acknowledge Dr. Rudell's contribution of advice and expertise throughout the course of my design.

I would also like to thank Dr. Visvesh Sathe for his close technical guidance and direct support throughout the course of this design. Dr. Sathe is always ready for a detailed technical discussion, which has been invaluable for hashing out the ideas presented in this work. Additionally, his tape-out support and expertise on digital circuits have been greatly appreciated at design time.

Thank you to Dr. Steve Perlmutter and Brian Mogen for committing time and lab resources to supporting this work. Neural interface work is nothing without in-vivo experimentation, and Dr. Perlmutter has graciously opened up his lab and test animals for our use in testing our neural interface circuits. Also, Dr. Perlmutter and Mr. Mogen have provided unique insight into the real-life efficacy of implantable neural systems. It has been extremely valuable to know the candid views and observations of practicing neuroscientists.

This collaboration with neuroscience experts was made possible by the UW Center for Sensorimotor Neural Engineering (CSNE), which brings together researchers from various scientific and engineering disciplines to develop cutting-edge neurological research. The learning

and exposure opportunities provided by the CSNE have been a source of motivation and inspiration for this work.

I would like to thank Eric Pepin for his mentorship and guidance during the early stages of my work. My experience with neural circuits started with developing a test platform for Eric's stimulator. Without Eric's expertise on the neural-electrical interface, this project would still be in its infancy. Work done on Eric's stimulator has kick-started our exposure in the neural engineering community and bolstered our legitimacy in the research world. Additionally, I appreciate Eric showing me the ropes in the EE graduate school and making my first months in the FAST lab welcome ones.

Thank you to William Anthony Smith for his hard work, mentorship, and advice. Working closely with Anthony on a full tape-out and two publications has given me a solid understanding of how to succeed as a research engineer. He has always been available for discussions regarding research, work life, and personal life. I am proud to call him a role model and a friend.

I would also like to thank members of the FAST and Sathe labs for their input, support, and hard work. Thanks goes to Tong Zhang, Andrew Chen, Ali Najafi, Samrat Dey, Rajesh Pamula, Patrick Howe, and Sung Kim. This research would not have been possible without their technical support and willingness for discussion.

Chapter 1. INTRODUCTION

Neural interfaces have been invaluable tools in increasing our understanding of the human nervous system and treating chronic neurological disorders. Stimulation of nervous tissue allows medical practitioners to reanimate paralyzed limbs, regulate malfunctioning endocrine systems, and halt or slow destructive neural processes (i.e. Parkinson's disease, depression, OCD) [1], [2]. In conjunction, neural recording enables early detection of similar conditions and gives us a much-needed window into the intricate workings of the human body.

Electrical neuromodulation is the most straightforward method of interfacing with the nervous system. Neurons naturally respond to and generate electrical potentials. Additionally, electrodes are relatively simple to fabricate and implant [3]. System complexity is pushed to the electronics side of the interface, an engineering realm accustomed to highly intricate designs. However, neuroscientists now require more sophisticated "closed-loop" interfaces to enable complex feedback networks, and a problem of orthogonality arises. In order to reliably influence and monitor the nervous system at the same time, the modulation tools must not interfere with any of the activity detection hardware. In purely electrical neuromodulation systems, the direct injection of current into the neural tissue can completely obscure any underlying, natural neural potentials. The simplest solution is to use different media for modulation and detection. Most modern "bi-directional" interfaces use a combination of optical stimulation or recording with an electrical counterpart. While this method is a quick fix for research environments, the use of two very different types of hardware and the necessary gene therapy for optical neuromodulation make this approach cumbersome in clinical settings. The desired tool for chronically implantable neural interfaces is a monolithic, small form-factor device that can interface with many individual

neurons in a bidirectional capacity. To achieve this goal, the interfaces will have to be entirely electrical.

Stimulating nervous tissue by injecting electrical current emulates the action potentials naturally propagated by neurons. When a neuron "fires" due to some input, signals are sent as waves of electrical potential differences across an insulated membrane. The membrane is selectively permeable to charged ions, with permeability influenced by the potential across the membrane. By artificially depolarizing this membrane with an injection of charge, electrical stimulation causes the selectively permeable membrane to allow ions to flow and start a cascade of action potentials [3]. This is typically done with a biphasic current pulse consisting of equal injection of positive and negative charge, as seen in Figure 1. Only one polarity of injection is necessary for neuromodulation, but the biphasic pulse prevents long-term buildup of charge and resulting tissue damage. Current flows through an electrode in physical contact with the nervous tissue and out through a differential electrode or some grounding contact. The electrode-tissue interface presents a resistive and capacitive component caused by charge transfer through ionic media (R_{CT}), insulating double-layer cell membranes (C_{DL}), and ionic solution resistance (R_S) [4]. The configuration of these impedances with respect to stimulation and recording is shown in Figure 1. As electrodes get smaller to increase neuron selectivity, the interface impedance increases. This trend leads to larger and larger voltages across the interface for a given amount of charge injection. These stimulation voltages seen at the electrodes (on the order of one to ten volts), propagate throughout the surrounding nervous tissue to varying degrees, dependent on current injection topologies and electrode placement. It is this propagation of stimulation voltages that has previously made electrical stimulation and recording incompatible in the same region of neural tissue.

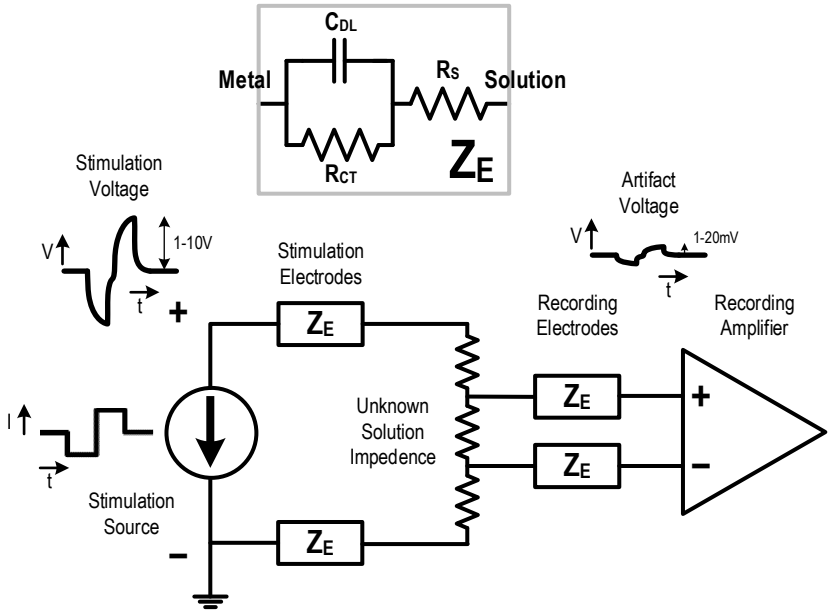

Figure 1: General Neural Interface Block Diagram

Electrical recording employs the same type of electrode-tissue interface to detect action potentials generated by individual neurons. As waves of electrical potential flow across neuron membranes, the resulting voltage difference can be detected with differential sensing between two positions in tissue. The problem is with regards to signal magnitude. Across any neuron's cell membrane itself, the action potential voltage never exceeds 120 millivolts peak-to-peak. This signal attenuates dramatically on propagation through surrounding tissue and out through an electrode. Typical action potential magnitudes at a recording electrode (a ten micrometer-wide platinum-iridium electrocorticography wire in this example) can be as small as one microvolt. It is already a significant challenge to record these signals in the presence of biological, environmental and electrical noise. This s exacerbated by the presence of large stimulation voltages, termed stimulation artifacts, caused by current injection anywhere else in the body. For example, stimulating the motor cortex of a Rhesus macaque with ±700 microamperes of locally applied current causes approximately ±10 millivolts of residual stimulation artifact at all locations

in the brain, with several millivolts propagating as far away as the arm. Several methods have been previously developed to protect recording electronics from these voltages, which will be described in Chapter 2.

Modern CMOS integrated technologies allows combination of stimulation and recording electronics on the same implantable device. Nanometer-scale lithography allows low-cost scaling to thousands of interface sites with integrated high-fidelity analog components and compact, low-power digital processing. This thesis proposes a highly scalable artifact cancellation method that allows simultaneous operation of stimulation and recording electronics on the same CMOS die. Leveraging the information processing capabilities of bulk CMOS, the system gathers information about the stimulation artifact and allows predictive subtraction of the stimulation voltage from the input of the recording amplifier.

Chapter 2. EXISTING METHODS

Electrical neural stimulation and recording have always been used in conjunction, as it is relatively uninformative to actuate without visibility or observe without manipulation. Accordingly, many efforts have been made to solve the stimulation artifact problem. Stimulators have been configured to reduce artifact amplitude via pulse shaping, power domain isolation, and use of orthogonal stimulation modes. Several recording systems have been designed to be artifact-immune with fast-restart input amplifiers and increased dynamic range. Finally, artifact cancellation methods have been developed in post-processing and in silicon. This chapter will enumerate recent efforts in artifact suppression that represent the state-of-the-art and identify the limitations and shortcomings of existing solutions.

2.1 STIMULATOR MODIFICATIONS

2.1.1 *Pulse Shaping*

The goal of an electrical neural stimulator is to accumulate charge over a certain threshold on neuron membranes near the stimulated electrode. The only critical specifications are the amount of charge on the membrane (enough to trip the action potential threshold) and that the charge is eventually removed to prevent chronic membrane depolarization or cell death. The amount of time it takes to deliver this charge (i.e. current amplitude) and its transient characteristics have minimal effects on the efficacy of the stimulation pulse. However, different stimulation waveforms have different energy loss profiles and are filtered differently by propagation through electrodes and tissue. Therefore, some stimulation pulse shapes more effectively deliver charge to cell membranes. One way to reduce artifact amplitudes seen at a recording input is to shape the stimulation pulses to better suit the frequency-domain characteristics of the electrode-tissue channel, reducing overall stimulation and artifact voltages.

Various stimulator pulse shapes have been quantified with respect to required stimulation energy, percent neuron recruitment, and optimal pulse timing. Examples of commonly studied pulse shapes are shown in Figure 2 [5]–[7]. It has been found that a Gaussian-shaped pulse requires less overall energy (the integrated current-voltage product) to achieve an appropriate stimulation threshold [6]–[8]. Based on the author's experience, parasitic capacitance in the interconnects, electrodes and tissue will shunt more energy from high-frequency pulse content, explaining why the pulse with the fewest sharp transitions (Gaussian) would be the most efficient. Stimulation pulse optimization has not been studied from an artifact reduction perspective, but a more efficient waveform would equate to a corresponding reduction in stimulation voltage and artifact amplitude. However, the energy and voltage savings are only approximately 40% with a Gaussian pulse, which is short several orders of magnitude from the necessary artifact suppression [8]. Additionally, generating arbitrary stimulation waveforms requires a high-voltage-compliant current DAC with greater precision and speed than those currently produced in silicon stimulation systems. The stimulator pulse-shaping approach to artifact suppression has minimal returns and is therefore not feasible.

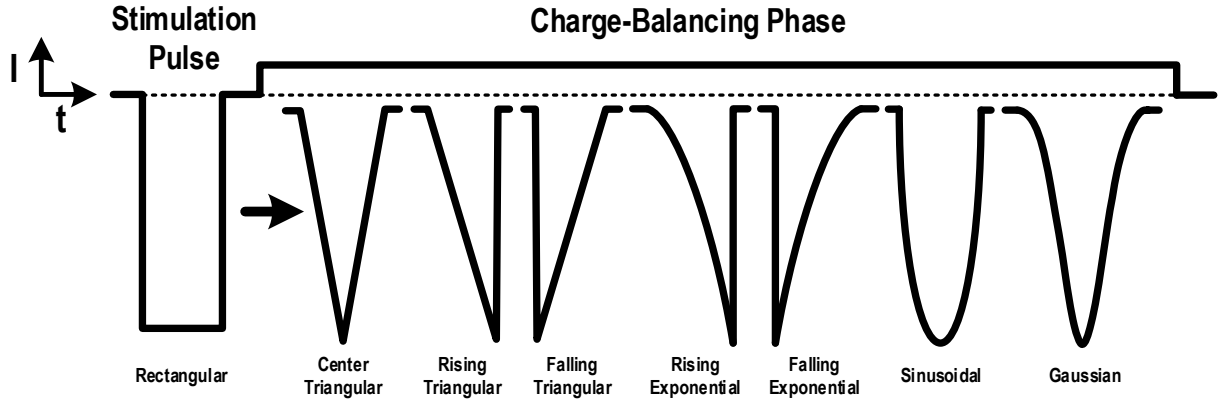

Figure 2: Commonly Studied Stimulator Pulse Shapes

2.1.2 *Power Domain Isolation*

In implantable neural systems, power is supplied by battery or wireless transfer. In either of these cases, there are several different ways to couple the stimulator and recording system power domains. Engineers at Medtronic have developed a method for reducing apparent stimulation artifact amplitudes by approximately 20dB by AC-coupling the recording and stimulation power domains [9]; a basic schematic of the configuration is shown in Figure 3. By AC-coupling the two power domains, the recording supply is free to move with the stimulation voltage. The recording ground reference is taken from the stimulated neural tissue, which sees similar artifact amplitudes to the differential recording inputs. This approach works for Medtronic's specific application. They are using separate stimulator and recording chips, each with a dedicated rectifier [9]. In the ideal single-chip solution, the stimulator and recording system will be sharing a common silicon substrate and most likely a common set of power rails and grounds. If the recording system ground were allowed to float with respect to the stimulator's "absolute" ground, there would be breakdown and reliability issues on chip.

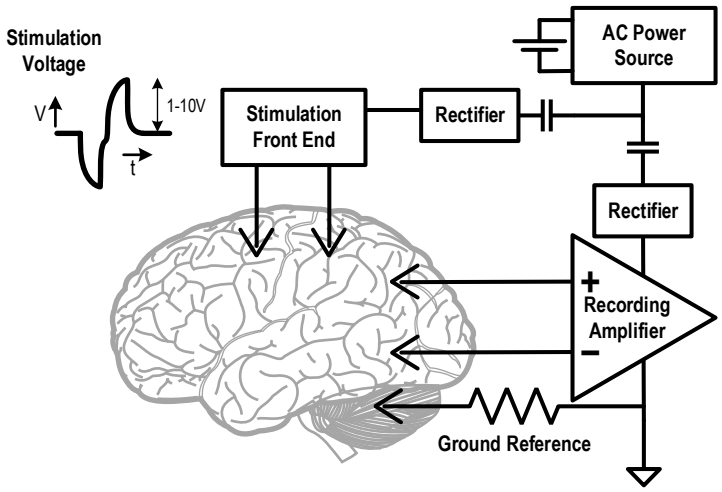

Figure 3: Power Supply Decoupling Block Diagram

2.1.3 *Optical Stimulation*

The stand-in solution to artifact suppression, used by most neuroscientists today, is optogenetic neural stimulation, first proposed by Francis Crick in 1979. Through transgenic conditioning, neurons can be made to produce light-sensitive ion channels, which cause the neuron to generate an action potential upon exposure to high-intensity light. This is done by genetically engineering viral carriers with opsin plasmids and injecting them into a living subject. Different opsins can be used to create ion channels sensitive to different wavelengths of light, allowing for several orthogonal stimulation modes [10]. These transgenic practices are used extensively in test rodents, where genetically-altering side effects can be ignored. However, transgenic gene therapy is a young technology. Optogenetics have been demonstrated in cultured human cells, but it is not yet ready for use in human patients; the danger of cancer or genome damage is too great.

Optical stimulation is used in conjunction with electrical recording to create many of the commercially produced BCIs available today. Light pulses do not induce any artificial electrical signals, so the two interface modes are completely orthogonal. State-of-the-art designs integrate fiber optic cables with electrodes in a single shank. These bidirectional setups are a sufficient placeholder for rodent-based in-vivo research, but optogenetic therapy is infeasible for human and primate use (the test subjects aren't expendable). The efficacy of optical stimulation has been demonstrated in rodents with artificially-induced epileptic seizures [11]. The action potentials induced by optical stimulation still slightly obscure electrical recordings, but their relative magnitude is the same as the surrounding signals of interest, meaning that the recording front end will not saturate.

2.2 RECORDING MODIFICATIONS

2.2.1 *Input Saturation Recovery*

Early bidirectional systems accepted amplifier saturation during stimulation and focused on decreasing saturation recovery times after stimulation [12]. This means that all neural signals occurring during stimulation and shortly afterwards are lost. This was considered acceptable because the latency between stimulation and resultant neural activity is typically on the order of milliseconds. The recovery time from amplifier saturation varies, depending on the amplifier topology. Consider the traditional closed-loop neural amplifier, shown in Figure 4. This amplifier topology is well optimized and ideal for matching gain and bandwidth between multiple channels. Matching integrated resistors and capacitors is much easier than matching process-sensitive amplifiers.

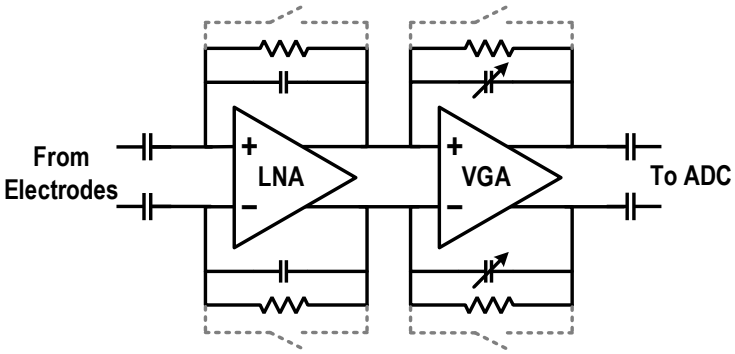

Figure 4: A Traditional Close-Loop Neural Amplifier (Feedback Reset Switches Shown in Grey)

The bandwidth of ECoG and neural acquisition systems is typically 1Hz to 2kHz or 32kHz. This means that the loop bandwidth of the closed-loop amplifier in Figure 4 extends down to 1Hz. Therefore, recovery from amplifier saturation will take full seconds. To speed up the process, reset switches are placed across the feedback caps (gray dotted lines in Figure 4), temporarily speeding up the amplifier response time during stimulation. This zeros out the differential amplifier gain during stimulation, but recording can resume as soon as the switches are opened once again. Open-

loop amplifiers used for chopper-based recording configurations have much lower settling times, so extra methods are not necessary for rapid amplifier recovery.

2.2.2 *Increased Input Dynamic Range*

Recent silicon front-ends have been implemented with dynamic ranges that can tolerate full-scale stimulation artifacts. The first implementation, from UCLA, relaxes front-end gain to keep stimulation artifacts within the linear range [13]. The system is chopper-stabilized to shape the heavy flicker noise at low frequencies, compensating for low gain. All filtering and noise shaping is done in the analog domain, with transconductance stages and switched capacitor/resistor networks for gain and filtering. This implementation trades an increase in power consumption for increased dynamic range, active filtering, and an input impedance boosted through feedback.

Another novel approach has been proposed by the same group at UCLA, incorporating voltage-to-frequency conversion through a high dynamic-range voltage controlled oscillator (VCO) [14]. The input voltage is buffered with a moderate amount of gain and then used to bias a ring oscillator. This approach has highly nonlinear gain which is compensated by estimating and cancelling the frequency-to-voltage transfer function nonlinearity with a 5^{th}-order polynomial fit. High frequency artifacts would create many intermodulation products and harmonics when passed through a nonlinear transfer function, which may not be completely suppressed by a 5^{th}-order inverse function.

2.3 ARTIFACT CANCELLATION

2.3.1 *Post-Processing*

Digital artifact cancellation was first developed as a post processing method to remove artifacts from pre-recorded data using a computer. Most neural recordings from clinical settings are taken with robust benchtop equipment that can easily handle and record the multiple millivolts

presented by stimulation artifacts. In this case, because the front-end amplifiers are not saturated, the stimulation artifacts are merely superimposed on the underlying neural signal in the final recording. After determining the shape of the artifact, one can easily subtract it from the recording at each stimulation instant. The art in post-processing is developing an automatic method for determining the artifact shape, rather than manually creating a "template" by visual inspection, a method called template subtraction.

Because the stimulator current waveform is always tightly controlled by the user, there are several different ways to infer the artifact shape from a recording. Local curve-fitting methods automatically generate accurate templates for each stimulation instance, relying on the known stimulation waveform as a boundary condition for the solution space [15]. This approach is computationally expensive, which is negligible in benchtop applications. More subtle methods have been developed that rely on the spectral properties of the artifact and the underlying neural signal. Neural signal power is highly concentrated below 2kHz and artifacts can have content in the hundreds of kilohertz. By selectively bandpass filtering the corrupted recording, decomposing, weighting, and summing its parts, an automatic template subtraction scheme can be created with relatively inexpensive digital signal processing (DSP) techniques [16]. Knowing the stimulation amplitude and timing, the artifact shape can be "learned" with filter-training regression techniques made popular in inter-symbol interference (ISI) cancellers for communications circuits. Discrete implementations of this training method can use complex self-training filters, such as the Weiner filter, having no computational bottleneck [17]. However, in integrated implementations, every operation costs power and area, so complex DSP algorithms are not appropriate.

2.3.2 *Front-End Cancellation*

An efficient integrated neural amplifier is not going to have a wide enough dynamic range to accurately record the neural signal and the superimposed artifact. Therefore, the only reasonable approach is to cancel the voltage artifact in the analog front-end before any gain is applied. A front-end artifact canceller was implemented at the University of Michigan, integrating 4-channel (8-channel) stimulation (recording) interface in 0.18µm CMOS with an adaptive equalizer [18]. Each recording channel has a dedicated input decoupling network, LNA, ADC, simplified LMS equalizer, and DAC for equalizer output. The area per recording channel is large, with several multi-picofarad integrated capacitors per channel. This system also falls short in voltage compliance in both stimulation and recording. The stimulators are limited to ±0.5V and artifact suppression is performed on single-millivolt-scale artifacts, rendering the system unusable for a majority of neural stimulation applications.

Chapter 3. METHODS

This section outlines the theoretical methods employed in creating an adaptive, integrated artifact cancellation system. First, there is discussion regarding the nature of the stimulation voltage artifact, from the perspective of the stimulator and the neural channel. Then interactions with the recording system front end are considered. Next, the nature of adaptive equalization and the least mean squared algorithm are explored. Finally, there is discussion regarding practical concerns and methods regarding system integration.

3.1 THE STIMULATION ARTIFACT

There are several different methods for injecting current into neural tissue to perform stimulation. The resulting current flow through the neural tissue is the same, but varying current source configurations and voltage supply placements affect the voltage artifact. There are a few key features common to each stimulator. First, each stimulator has a current-controlled sink to ground for the leading phase of the current pulse. Where the current is sourced varies among topologies. Second, a high-voltage supply sources current through a current-controlled source. High voltages are necessary to source up to milliamps of current through relatively high electrode impedances. HVDD voltage supplies are typically 10V to supply milliamps through electrodes with tens of kilo-ohms of DC impedance. Differing HVDD placements affect the DC bias of the neural tissue and the appearance of voltage artifacts. This section discusses how each stimulator topology generates artifacts at the electrode interfaces. Propagation of artifacts through neural tissue will be discussed later.

3.1.1 *Ground-Return Stimulators*

The ground-return stimulator is the simplest stimulator configuration. There is an active electrode that performs the controlled current sink and source. A common return electrode is held

at a DC voltage to provide the current return path for both phases. In benchtop configurations, the return is typically held at ground, while the current sources are operated on positive negative supplies. However, in integrated systems, a single supply necessitates that the return electrode is held at HVDD/2, as shown in Figure 3.1 [19], [20]. This topology is preferred for high-channel count stimulation, as each channel only requires one dedicated electrode, with a shared return electrode. Integrated ground-return topologies bias the neural tissue to an intermediate DC voltage, preventing DC-coupled recording front-ends. This configuration creates a voltage artifact at the active electrode similar to the voltage waveform in Figure 5.

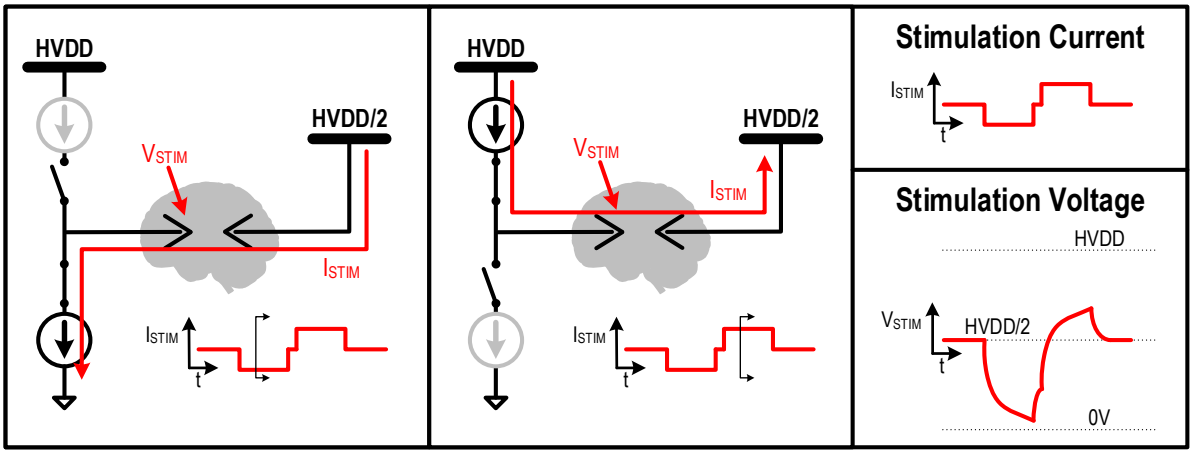

Figure 5: The Ground-Return Stimulator Topology

3.1.2 *Differential Stimulators*

A fully differential stimulator connects a regulated current source to each active and return electrode, one side sourcing and the other sinking for the first phase and visa-versa for the second phase, as shown in Figure 6. This is a common configuration for single-channel stimulators, as charge balancing is easily accomplished with standard CMOS matching practices. Additionally, the effective voltage compliance of the differential stimulator topology is double that of the ground-return configuration. Each electrode sees an equal and opposite voltage change for all

stages of stimulation, with the range of HVDD to ground for each electrode, as can be seen in Figure 6. The main difficulty in designing differential stimulators is matching source-mode and sink-mode current sources [12]. This is especially difficult for high-voltage-compliant stimulators, as the source-mode current source will be biased above the normal voltage rails of a given process, modulating impedances and increasing control complexity. Additionally, both electrodes must be pre-charged to a mid-rail DC value to ensure reliable current source operation. This presents the same DC-coupling issue as with the ground-return topology.

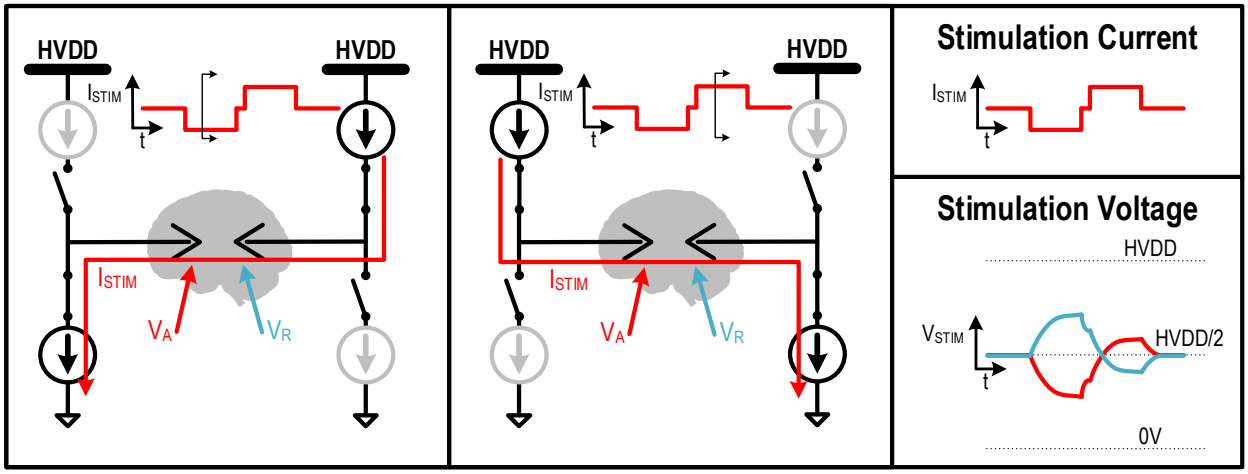

Figure 6: The Fully Differential Stimulator Topology

3.1.3 *H-Bridge Stimulators*

The stimulator topology used for the experiments in this work is the H-Bridge, specifically the circuit design presented in [21], [22]. The H-Bridge uses only sink or source-regulated current supplies, easing the matching concerns of the differential stimulator. The high-voltage supply is used to directly bias the neural tissue during stimulation, decreasing system complexity by eliminating the mid-rail supply. During stimulation, one electrode is connected to HVDD while the other sink-regulates current to ground. The opposite could be done, connecting to ground and source-regulating from HVDD, but it is much easier to design and control ground-connected

current sources. Like the differential stimulator, the effective voltage compliance is doubled as compared to the ground-return configuration. The H-Bridge stimulator does generate unique artifacts, as the electrode voltages are monopolar and unbalanced. The next section will discuss the exact effects in detail.

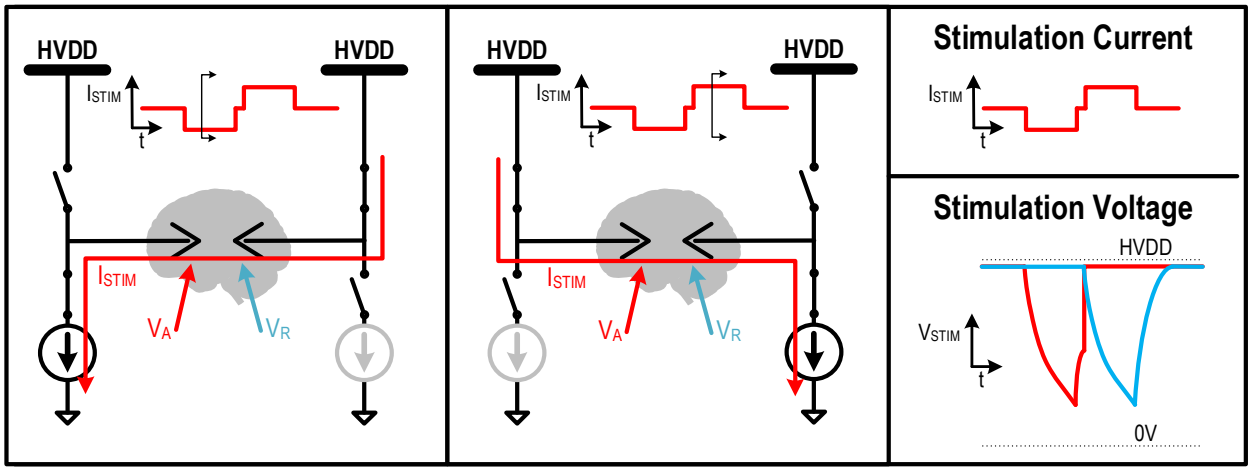

Figure 7: The H-Bridge Stimulator Topology

3.1.4 *Artifact Propagation*

There are two components to the stimulation voltage artifact: common-mode voltages that propagate to all connected neural tissue due to stimulation voltages at the electrode-tissue interface and voltage differences between two locations in neural tissue due to current flowing through the tissue impedance. Both types of artifacts affect the recording front end: common-mode artifacts change input biasing conditions (especially for open-loop amplifier topologies) and differential-mode artifacts will saturate any high-gain amplifier.

The common-mode artifact is highly dependent on stimulator topology. Once past the electrode-tissue interface, neural tissue is relatively low impedance, consisting of a fairly homogenous, low-impedance distributed series-R, shunt-C network. The common-mode artifact seen at all terminals of the recording architecture is then a superposition of the stimulator's active and return electrodes, depending on the particular recording electrode's position with respect to

the stimulator electrodes. Typically, the electrodes for differential recording are close enough together that this “common-mode” signal will propagate to the positive and negative inputs equally. However, if there is any difference in distance between the two recording electrodes and the stimulation electrodes, the common-mode superposition of the two stimulator electrode voltages will be imbalanced and manifest itself as a differential signal.

Figure 8 shows an example of the superposition of the two electrode voltages as a resulting common-mode artifact. For the ground-return stimulator topology, a slightly attenuated version of the single active electrode voltage is seen at all points in the neural tissue. As the recording electrodes approach the active electrode, the signal will get larger. The opposite is true for the AC-grounded return electrode. The differential stimulator ideally generates no common-mode artifact, however any non-symmetrical electrode configuration will still result in a small common-mode artifact. The H-bridge stimulator produces a unique artifact shape. As each of the electrodes is driven up (or down) in voltage, a monophasic common-mode voltage artifact is generated. The other electrode is held at HVDD for the other electrode’s stimulation phase, so the common-mode artifact should be half the size of each electrode’s peak voltage (assuming the recording electrodes are placed exactly equidistant to the stimulator electrodes.

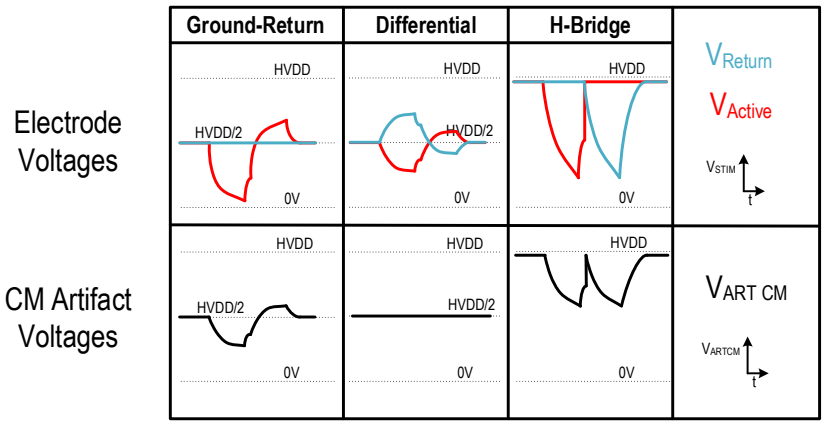

Figure 8: Common-Mode Artifact Superposition Examples

In addition to the common-mode artifact caused by all neural tissue moving with the stimulator electrode voltages, there is a differential-mode artifact induced by stimulator currents flowing through the neural tissue itself. Any tissue space between differential recording electrodes presents a mostly resistive impedance. As the stimulator current propagates between stimulator electrodes, it radiates throughout the neural tissue, which is a distributed impedance created mostly by the saline solution between living cells. Based on measured artifacts during experiments, the internal tissue impedance seen by the stimulation current once in the tissue is mostly real, with a distributed shunt capacitance to ground. This tissue impedance creates a small voltage difference between the differential recording terminals, presenting itself as a differential-mode artifact, as shown in Figure 9. These artifacts are orders of magnitude smaller than the common-mode artifacts described above (approximately 10mV compared to ~1V common-mode artifacts). Based on measurements, the equivalent Z_{TISSUE} for differential stimulation across one millimeter of neural tissue with recording electrodes in parallel two millimeters away is approximately 15Ω with negligible shunt capacitance to ground. The low-pass effects of the neural tissue itself are overpowered by the parasitic capacitances in the electrodes, wires, and interconnects.

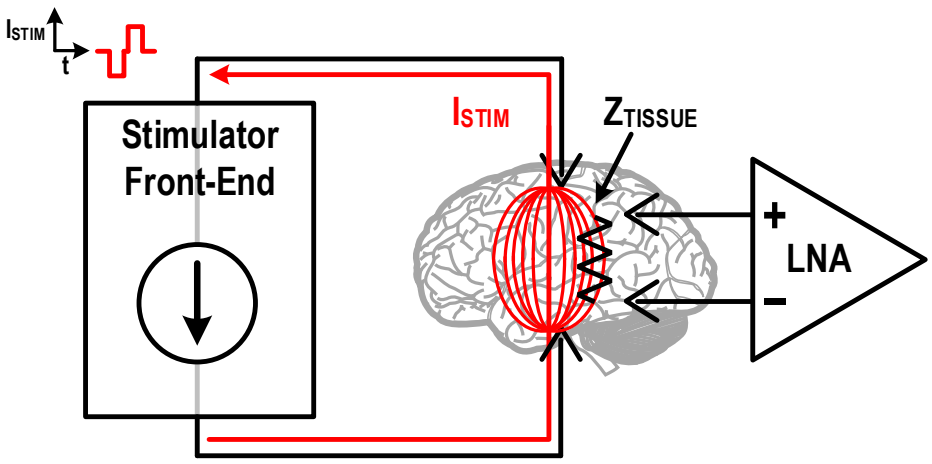

Figure 9: Differential-Mode Artifact Generation Method Example (Measured Equivalent Z_{TISSUE} ~15Ω)

3.1.5 *Experimental Validation*

An in-vivo experiment was performed in November of 2015 to determine the exact amplitudes and electrode-placement dependence of common-mode and differential-mode artifacts. Stimulation was performed with a fully differential stimulator in the motor cortex of a sedated non-human primate. All electrodes in this case are platinum-iridium "macrowires" in two square electrocorticography (ECoG) patches, one on the brain surface and another implanted 1mm directly below. Stimulation was performed between various electrodes, while the recording electrodes were held static.

The following figures depict voltages for 10 different electrode configurations, each stimulated with 70μA, 200μA, 500μA, and 700μA. During the experiment, stimulation electrode voltages reached peaks of 13V with 700μA of controlled current, as shown in Figure 10. Ideally, the voltage for each electrode location would be the same. However, physical damage, differences in tissue contact, and material impedance mismatches change each electrode's impedance and current-voltage transfer characteristic. This mismatch also manifests itself as a common-mode artifact. As discussed earlier, the differential stimulator should have a near-zero common-mode

artifact. However, common-mode voltages were observed for all electrode configurations, even those that were supposed to be symmetrical, as shown in Figure 11. Notice that the artifact voltages are ~100x smaller than the original stimulator voltages, which is consistent with the impedance mismatch theory.

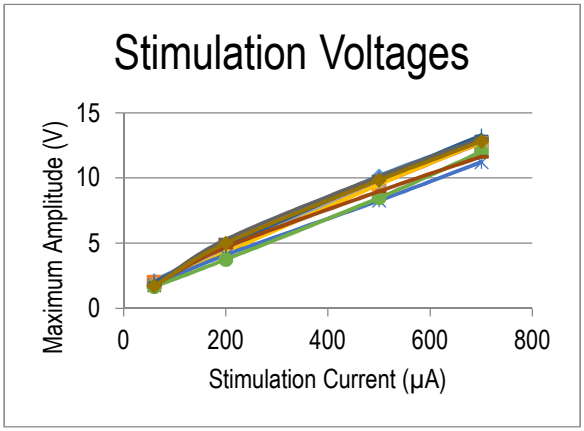

Figure 10: Stimulation Electrode Voltages at Varying Locations and Stimulation Currents

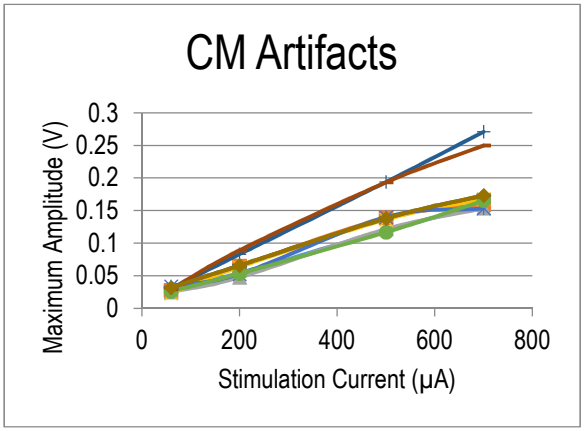

Figure 11: Recording Electrode Common-Mode Voltages for Varying Stimulation Locations and Currents

Differential artifacts were also observed for every stimulator electrode configuration, as shown in Figure 12. However, there were significantly larger differential artifacts when the differential stimulator electrodes are placed on either side of the differential recording electrodes. That is, the recording electrodes are directly in the current flow path from one stimulator electrode to the other. This is consistent with the tissue impedance model proposed in the earlier section. Additionally, the ~20mV peak artifact voltage observed in this experiment is the worst-case differential artifact observed in five in-vivo trails over the last 2 years. Therefore, 20mV has been used as the goal for the maximum cancellable artifact in this work. Another notable conclusion from this experiment was that different electrodes experience different artifacts. This means that the artifacts seen on adjacent recording channels, even if the electrodes are less than a millimeter apart, must have different stored artifact profiles for efficacious cancellation on multiple channels.

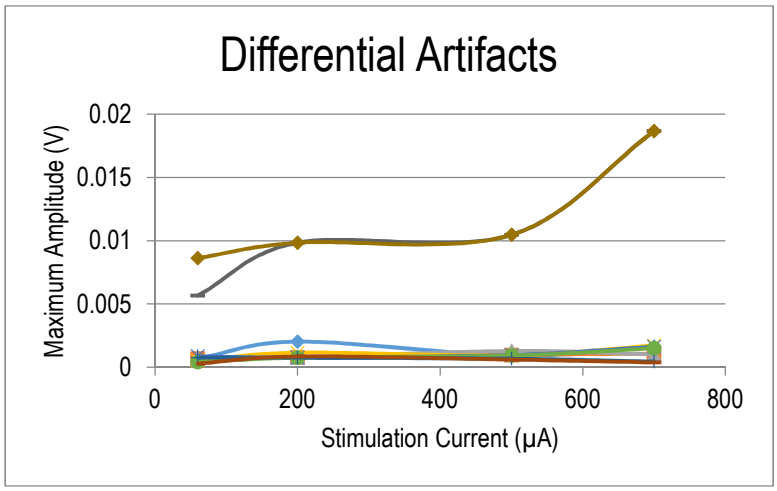

Figure 12: Recording Electrode Differential-Mode Voltages for Varying Stimulation Locations and Currents

3.2 The Recording Front-End: A Mixed Signal Interface

The ideal integrated neural recording front-end has a very well-defined set of design specifications. The recording bandwidth must extend from near-DC (~0.1Hz) to 2kHz for ECoG and 32kHz for full neural signals. The front-end amplifier should have 60dB of gain to capture microvolt-level neural potentials. The recording inputs must have a near-infinite input impedance; the capacitive nature of neural electrodes causes the input transfer function's high pass corner to raise with reductions in input impedance. Additionally, the recording chain as a whole must contribute ~200nV/√Hz of noise to prevent obfuscation of the small neural signals.

These specifications severely limit the topological choices available for an integrated artifact canceller. Namely, the input impedance and noise specifications disallow connection of any active devices to the input nodes. Injection of any noise current into the high-impedance inputs (typically MOSFET gates) will cause large noise voltages. To the best of the author's knowledge, there are no amplifier topologies capable of sub-femtoamp/√Hz noise currents. For this reason, no auxiliary-path amplifiers can be fed back into the input nodes for artifact cancellation. Additionally, current draw at the input node (via charging switched capacitances or resistive loads)

causes a decrease in the real input impedance of the front-end amplifier. Equivalent input impedances below 100MΩ raise the front-end high pass corner above the desired signal bandwidth of sub-1Hz. This is due to wide and long input MOSFET devices (sized to reduce flicker noise) that form large shunt capacitance to ground. As a result, only high-impedance switched-capacitor circuits can be used to interface with the recording input nodes. Because of the pre-existing input capacitance, additional capacitance at the input node will not contribute excessive kT/C noise or affect input impedance. A digitally controlled capacitive DAC (CDAC) is the safest method for adding or subtracting charge from the input terminals without adding noise or reducing the input impedance.

3.3 Adaptive Equalization Methods

Modern communications technologies use extensive adaptive equalization techniques to predict the characteristics of an unknown channel by observing the response to a known (or predictable), given input signal. In the earliest examples of adaptive channel equalization, the input signal consisted of a modulated sinusoid transmitted over a dispersive, non-linear medium. This was typically a radio-frequency transmission through air from one antenna to another. The channel in this case consists of the direct transmission path as well as reflections off of various surfaces. These reflections result in inter-symbol interference (ISI), where a transmitted symbol earlier in time affects the information currently being received. The adaptive equalizer in this case convolves the received signal with an approximation of the transmission channel's inverse transfer function to recover the originally transmitted signal. The inverse channel is approximated by minimization of various optimization functions, several of which will be explored in this chapter. The techniques developed for communication channel estimation can also be applied to predicting the transmission of stimulation signals through neural tissue. The same approach used to isolate the

original signal by attenuating its delayed components can be extended to attenuate all signals correlated to the stimulation pulse. Additionally, the characteristics of the input stimulation pulse are completely known, allowing for more guided approaches than those typically employed in blind channel estimation.

3.3.1 *Linear Transversal Equalizers*

Adaptive equalizers are built around a tapped delay line architecture known as the linear transversal equalizer (LTE). Samples of an input waveform are propagated down a delay line of arbitrary discretization and tapped samples are scaled and summed at the output node. Figure 13 shows the basic operation, where the input signal, $x(n)$, is delayed and tapped in discrete units. Each tap is scaled by a set of filter coefficients, $h_{0\ldots N}$, and summed to create the output, $y(n)$. The resulting transfer function is, also shown as a block diagram in Figure 13:

$$y(n) = \sum_{i=0}^{N} h_i x(n-i) \tag{3.1}$$

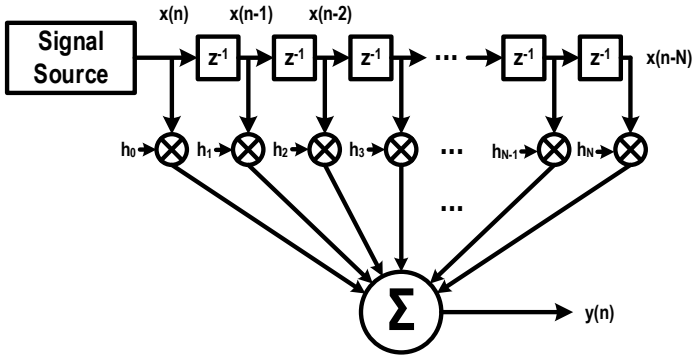

Figure 13: Linear Transversal Equalizer Block Diagram

This structure is used for both feedforward and feedback architectures. The infinite impulse response (IIR, feedback) or finite impulse response (FIR, feedforward) nature depends on the causality relationship between the input and output. The inclusion of feedback from output to input allows an impulse input to create an infinitely continuous output.

The LTE structure was invented by communications engineers in the early 1960's, before the advent of computers or digital circuits. The implementation in this work is entirely digital, but their filters were continuous-time and often physically tuned by hand. Figure 14 [23] illustrates the complex analog structures that were originally used to implement the FIR filter.

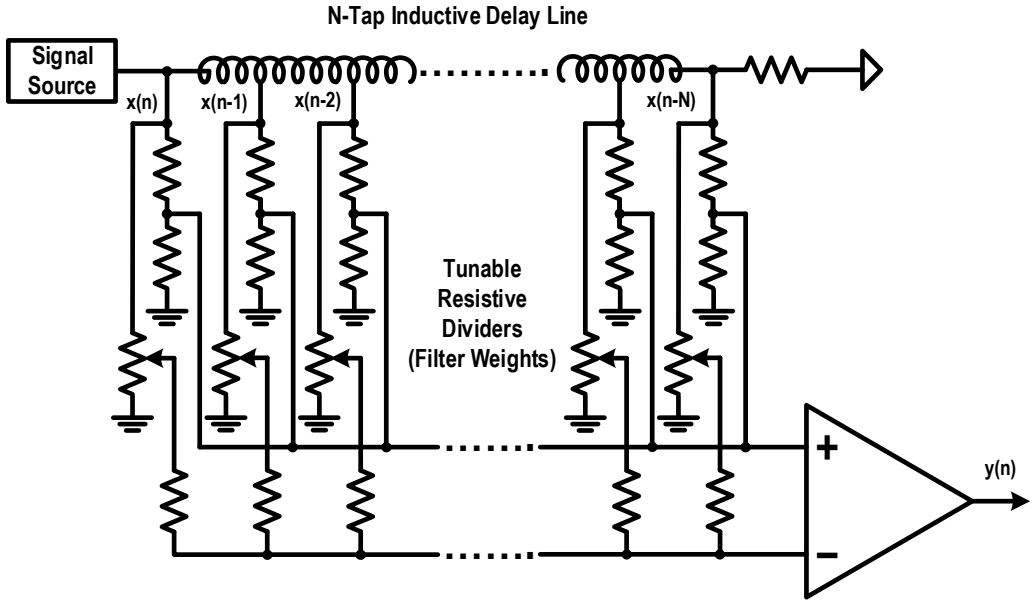

Figure 14: Discrete Analog Implementation of the Linear Transversal Equalizer

3.3.2 *Least Mean Squared Error Minimization*

The least mean squared (LMS) adaptation method was first developed in the early 1960's in Bell Telephone Labs by Floyd Becker, Erich Port, and Robert Lucky [23]. This method of filter coefficient estimation is widely used for its quick convergence, ease of implementation, and high noise tolerance. The following discussion explains the algorithm in its original context (preserving the original input signal after de-convolving the channel response). The theory will be expanded slightly to relate to the presented work.

Given a transmitted input signal $x(n)$ and a channel with impulse response $g(n)$, the received signal $y(n)$ represents a convolution of these two continuous-time functions, where

$y(n) = g(n) * x(n)$. In channel estimation equalizers, the impulse response $w(n)$ is built to be the inverse of the channel response, where the goal is that $w(n) * g(n) = \delta(n)$. $\delta(n)$ is the Dirac delta function, where $\delta(n) = \begin{Bmatrix} 1, n = 0 \\ 0, n \neq 0 \end{Bmatrix}$. Meeting this condition, the output of the equalizer, $h(n) = y(n) * w(n)$, is ideally equal to the original input signal, $x(n)$. For the sake of generalization, the filter has taps for time indices from $-N$ to N (this implies a combined feed-forward and feed-back architecture). This signal flow is shown in Fig. 15.

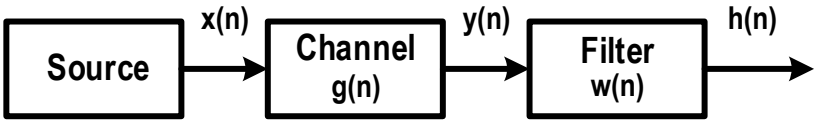

Figure 15: LMS Derivation Signal Flow Block Diagram

The original LMS algorithm uses mean squared error (MSE) to evaluate filter effectiveness. The MSE quantifies the amount of residual signal in $h(n)$ that is not equal to $x(n)$. Derivation of the MSE begins with the mean squared distortion (MSD) [23], [24], defined by:

$$MSD = \frac{1}{h^2(0)} \sum_{\substack{n=-\infty \\ n \neq 0}}^{\infty} h^2(n) \tag{3.2}$$

The MSD quantifies the power of residual errors left over after deconvolving the channel impulse response, normalized by the original signal power. The MSE is the non-normalized form of this quantity:

$$MSE = \epsilon = \sum_{\substack{n=-\infty \\ n \neq 0}}^{\infty} h^2(n) = \sum_{n=-\infty}^{\infty} h^2(n) - h^2(0) \tag{3.3}$$

Recall that $h(n)$ is defined as the convolution of the channel output and the filter response. If N defines the length of the discrete-time filter, it follows:

$$h(n) = y(n) * w(n) = \sum_{i=-N}^{N} w(i)y(n-i) \tag{3.4}$$

$$h(0) = \sum_{i=-N}^{N} w(i)y(-i) \tag{3.5}$$

The definitions in Equations 3.4 and 3.5 identify ϵ as a quadratic function of the filter coefficients $w(n)$, which means it has a global minimum with respect to the coefficients themselves. The minimum can be found with differentiation:

$$\frac{\partial \epsilon}{\partial w(i)} = \frac{\partial}{\partial w(i)}\left(\sum_{n=-\infty}^{\infty} h^2(n) - h^2(0)\right) \tag{3.6}$$

$$= \sum_{n=-\infty}^{\infty}\left(2h(n)\frac{\partial h(n)}{\partial w(i)}\right) - 2h(0)\frac{\partial h(0)}{\partial w(i)} = 0$$

Using Equations 3.4 and 3.5 to compute the two partial derivatives, Eq. 3.6 simplifies to:

$$\sum_{n=-\infty}^{\infty} h(n)y(n-i) = h(0)y(-i), \qquad i = -N \ldots N \tag{3.7}$$

The main tap of the filter output, $h(0)$, can be arbitrarily set to be 1. This simplifies calculations and can be compensated with scaling later on. This does not affect minimization of MSE for the ISI contributors (taps intended to cancel delayed components of the channel output). This simplification and substitution of Equation 3.4 gives:

$$\sum_{n=-\infty}^{\infty}\sum_{j=-N}^{N} w(j)y(n-j)y(n-i) = y(-i) \tag{3.8}$$

Bringing the filter taps out of the second summation:

$$\sum_{j=-N}^{N} w(j)\sum_{n=-\infty}^{\infty} y(n-j)y(n-i) = y(-i) \tag{3.9}$$

The correlation coefficient of the channel output can be written as:

$$a(i,j) = \sum_{n=-\infty}^{\infty} y(n-j)y(n-i) \qquad 3.10$$

Equation 3.9 then simplifies to:

$$\sum_{j=-N}^{N} w(j)a(i,j) = y(-i), \qquad i = -N \ldots N \qquad 3.11$$

This gives a linear equation for each filter tap $w(i)$, from $-N$ to N. The $(2N+1)$ resulting equations are known as the Wiener-Hopf equations [23]. These equations can be solved concurrently with full system visibility and extensive computational capability. However, in realistic applications, there are limits to memory and computational power. Systems do not typically store each $y(i)$ from $-N$ to N. Further simplifications allow computationally efficient estimation of the solutions to Equation 3.11.

The solution process is greatly simplified with a known input signal. In the case of wireless transmission, a predetermined training sequence is sent to accelerate the equalizer adaptation. The known training signal is sent for a predetermined period of time. After this time, the real data is transmitted. It is assumed that the channel characteristics remain constant after this initial training sequence. In real applications, the training sequence is sent periodically to recalibrate the equalizer to changing channel characteristics. In neural stimulator artifact cancellation, every input is known; the stimulator, recording system, and artifact canceller are controlled by the same circuitry and stimulation parameters are user-specified. Therefore, every stimulation pulse can be used to train the adaptive filter.

Knowing the input signal, $x(n)$, we can define an error signal, $e(n) = x(n) - \tilde{x}(n)$, which represents the sample-by-sample error of the equalizer, calculated using the equalizer output redefined as $\tilde{x}(n)$. Based on this error signal, the MSE becomes:

$$\epsilon = \sum_{n=-\infty}^{\infty} e^2(n) \tag{3.12}$$

Defining these quantities in terms of Equation 3.4:

$$e(n) = x(n) - \sum_{i=-N}^{N} w(i)y(n-i) \tag{3.13}$$

$$MSE = \epsilon = \sum_{n=-\infty}^{\infty} \left[x(n) - \sum_{i=-N}^{N} w(i)y(n-i) \right]^2 \tag{3.14}$$

Similar to the process before, the optimum filter response is given by minimizing the MSE with respect to the filter coefficients via partial derivative:

$$\frac{\partial \epsilon}{\partial w(i)} = -2 \sum_{n=-\infty}^{\infty} \left[\left(x(n) - \sum_{j=-N}^{N} w(j)y(n-j) \right) y(n-i) \right] = 0 \tag{3.15}$$

Rearranging this equation into a more easily recognizable form:

$$\sum_{n=-\infty}^{\infty} x(n)y(n-i) = \sum_{j=-N}^{N} w(j) \sum_{n=-\infty}^{\infty} y(n-j)y(n-i) \tag{3.16}$$

At this point, the minimization method must reconcile the limited nature of real systems. In Equation 3.14, error is minimized over an infinite interval of n, which is practically impossible. To overcome this, we will follow the autocorrelation method [25]. Data outside of the range $n \in [-N, \dots, N]$ is assumed to be zero. Then, recognize that Equation 3.16 contains the autocorrelation of the channel output

$$r(i,j) = r(|i-j|) = \sum_{n=-N}^{N} y(n-j)y(n-i) \tag{3.17}$$

and the cross-correlation of the predetermined input sequence and the received channel output:

$$\psi(-i) = \sum_{n=-N}^{N} x(n)y(n-i) \tag{3.18}$$

A new set of linear equations can now be defined, through substitution into Equation 3.16:

$$\sum_{j=-N}^{N} w(j)r(i,j) = \psi(-i), \qquad i = -N \ldots N \tag{3.19}$$

These equations are similar to those generated in Eq. 3.11, but they represented guided filter training. "Blind" training, based entirely on channel output characteristics, can lead to rapid accumulation of errors. Blind error minimization will mathematically optimize the filter, but guided training is far more likely to find the true channel characteristics through incorporation of a known training signal.

Equation 3.19 presents a significant improvement from the equations in Eq. 3.11, but there is still an issue of memory and computation. For each $(2N + 1)$ equalizer tap, there is a channel output sample to be stored and a linear equation to simultaneously solve. This restricts these accurate and thorough adaptation methods to post-processing, where there is typically unlimited computational power.

3.3.3 *Decision Feedback Equalizers*

To overcome the computation bottleneck, communications engineers developed sample-by-sample MSE minimization techniques in the late 1960's and early 1970's [26], [27]. These techniques are simplified steepest-gradient descent methods derived from the basic LMS algorithm. Rather than minimizing the squared error for the entire sample space (requiring

simultaneous solution of many linear equations), the expectation of the squared error is minimized for a given tap and corresponding sample in time. Statistical analysis is incorporated, taking advantage of the random, additive noise that is picked up in the transmission channel. Additionally, later derivations of decision-based LMS algorithms differentiate between feed-back and feed-forward filters with separate sets of taps, lending themselves to realistic filter designs [27]. A diagram of a typical decision-feedback architecture is shown in Figure 16.

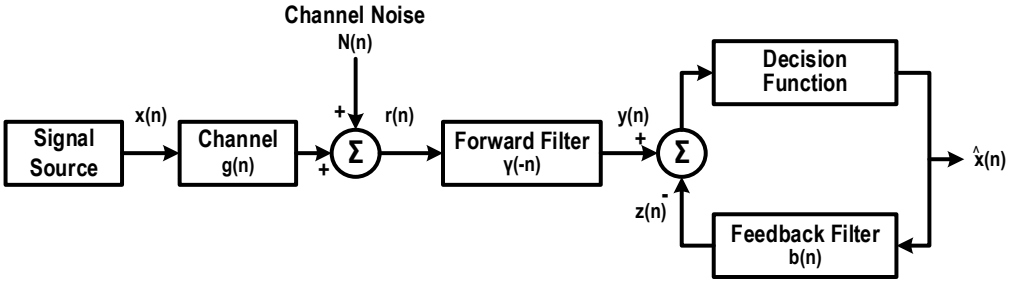

Figure 16: Generalized Decision Feedback Equalizer

In this diagram, as before, $x(n)$ represents the original input signal, which is reconstructed after filtering as $\hat{x}(n)$. $r(n)$ represents the channel output, including additive Gaussian noise. $y(n)$ is the forward filter output and $z(n)$ is the feedback network output. This system merely a specific implementation of the generic equalizer structure outlined in the beginning of this chapter. The negative tap numbers are merely designated as the forward filter, while positive tap numbers represent the feedback filter. For this derivation, we will assume that the input is a known training signal and that the goal is $\hat{x}(n) = x(n)$.

Redefining the error function in terms of feed-forward and feed-back components:

$$e(n) = y(n) - z(n) - x(n)$$
$$= \sum_{k=-\infty}^{0} \gamma(k) r(k+n) - \sum_{k=1}^{\infty} b(n) x(n-k) - x(n) \qquad 3.20$$

Generalizing this error function into a discrete Hilbert space, we can minimize error with an expectation function ($\langle a, b\rangle$ represents the inner vector product of a and b):

$$\min_{\gamma,b} E\left[\left(\langle \gamma, r(n)\rangle - \langle b, x(n)\rangle - x(n)\right)^2\right] \tag{3.21}$$

Solution of the minimum in Equation 3.21 is typically done in the frequency domain, generalized to continuous time systems. It is outside of the scope of this discussion. However, the minimum can be approximated by steepest gradient descent, where filter tap coefficients are modified by the error gradient with respect to that tap to achieve convergence. A convergence parameter, μ, is added to tune stability and locking rate. This solution can be expressed for each tap in terms of expected values (Note that tap values γ and b will now be represented with indices of time, not tap location):

$$\gamma(n+1) = \gamma(n) - \mu(E[r(n)^2]\gamma(n) - E[r(n)x(n)]) \tag{3.22}$$

$$b(n+1) = b(n) + \mu(E[x(n)^2]b(n) - E[x(n)^2]) \tag{3.23}$$

The statistics of the channel output and input signals are not known, due to the memory and computational constraints noted earlier. For this reason, the gradient descent algorithm must be further estimated using a limited number of samples, with the number of averaged samples L. This simplifies Equations 3.22 and 3.23 to:

$$\gamma(n+1) = \gamma(n) - \frac{\mu}{L}\sum_{i=nL-L+1}^{nL} r(i)\big(r(i)\gamma(n) - x(n)\big) \tag{3.24}$$

$$b(n+1) = b(n) + \frac{\mu}{L}\sum_{i=nL-L+1}^{nL} x(i)\big(x(i)b(n) - x(i)\big) \tag{3.25}$$

Notice that the final term inside of the sum represents the error function defined in Eq. 3.20, albeit in two different state spaces. In the context of neural processing, a feedforward filter

is unnecessary for the desired equalization accuracy. Ignoring the feedforward space, we can generalize a feedback decision metric:

$$b(n+1) = b(n) + \frac{\mu}{L} \sum_{i=nL-L+1}^{nL} x(i)e(i) \quad 3.26$$

In applications with little noise and small residual MSE, a single sample is sufficient for convergence ($L = 1$). This simplifies the update equation by removing averaging, resulting in the most well-known form of the LMS update algorithm:

$$b(n+1) = b(n) + \mu x(n)e(n) \quad 3.27$$

3.3.4 *Stability and Convergence of the Basic LMS Update Algorithm*

Ensuring that an adaptive equalizer will converge to the optimum, lowest MSE solution is a matter of tuning the step-size parameter, μ, to match the error signal power. In the sample-by-sample example that was just enumerated, it is difficult to analyze the statistics of convergence, as the coefficient update algorithm is a highly non-linear system. However, by abstracting the update to arbitrarily small step sizes over an infinite number of averaged samples, general properties emerge.

To categorically ensure convergence, the step-size parameter must be chosen so that it is the inverse of the product of the total normalized error signal power and number of filter taps. The error signal power in this case includes both the channel response correlated to the input signal and all additive noise. The normalized maximum captures the maximum transient error magnitude with respect to the average error power magnitude. The derivation in [26] assumes a continuously non-zero input function, so all N taps of the filter contribute to the output, and an increasing

number of taps engaged at once contributes to algorithm instability. This leads to the convergence criterion [24], where the integration period, T, defines the known set of error values:

$$\mu\, N \max \left| \frac{e(n)^2 \cdot T}{\int_{-\frac{T}{2}}^{\frac{T}{2}} e(n)^2} \right| \leq 1 \qquad 3.28$$

It follows that smaller update step sizes are required for increased noise or signal power. This will always slow convergence. In systems with limited computational accuracy, the divisions made necessary by very small update step sizes alters this ideal convergence criterion. Truncation of less-significant bits and digits can cause overshoot or undershoot in the weight update step, hindering convergence speed and stablity.

The update algorithm can be further simplified by limiting the amount of information known about the input signal and the error signal. The numerical accuracy of $x(n)$ and $e(n)$ can be limited to only contain information about signal polarity. This slows convergence speed but grants a degree of noise immunity and reduces computational complexity. These methods are called the sign-data or sign-error LMS algorithms. By thresholding the input signals to sign information, MSE and noise must take up the full range of the input quantizer to cause stability issues.

3.3.5 *Scalability and Efficiency*

Even in its most limited form, the LMS algorithm still involves several multiplications, additions, and a division, non-negligible computations in custom digital blocks. These operations must be performed multiple times for each weight update iteration, which may happen at frequencies approaching 1MHz. Additionally, the trained filter taps are equivalent to an averaged transient domain recording of the artifact voltage. Storing enough information to cancel millisecond-long artifacts for many, many channels becomes a considerable task of memory

management. This work develops an adaptation algorithm and filter implementation that can be scaled between CMOS processes, operating frequencies, and increasing channel counts.

The idealized versions of the LMS update filter given in Equations 3.4 and 3.27 gain significant complexity when implemented as bitwise hardware-defined digital systems. Typical neural recordings in integrated circuits consist of 8-bit analog-to-digital (ADC) samples, sometimes supplemented by additional modulated samples [12], [18], [28]. This defines the baseline resolution of $e(n)$ to 8 bits. In order to keep artifacts within the linear range of the amplifier and ADC, 20mV artifacts must be suppressed by 40dB, to approximately 200μV. A DAC with 8 bits of resolution gives ~48dB of dynamic range, a sufficient range for 40dB of cancellation. For this reason and convenience, the filter taps, $b_{0\ldots N}$, will each have 8 bits of resolution. If the algorithm is to respond to changing stimulation amplitudes, the filter input, $x(n)$, must also have a non-negligible bit depth. In the stimulator used for this work, the H-bridge sink-mode current DAC has 9 bits of resolution. Therefore, for baseline versions of this filter design, $x(n)$ was given a 9-bit depth. Finally, the update step size parameter, μ, must be sufficiently tunable to make the filter reliably converge at varying speeds and under varying noise and input conditions. μ is always less much less than one to ensure the stability criterion given in Equation 3.28. Bitwise division is exceptionally expensive operation, so μ is implemented as a combination of an arithmetic bitwise shift towards the least-significant bit (LSB) and multi-sample averaging.

Assigning these bit depths turns the weight update operation given in Equation 3.27 into the following series of operations: an 8-bit by 9-bit multiplication, scaling the 17-bit product by μ, adding the result to the 8-bit weight, and storing the sum for the next artifact sample. Each filter tap is then another 8-bit by 9-bit multiplication, resulting in N 17-bit products to sum for the output. A block diagram for this unrefined digital filter implementation is shown in Figure 17. All

throughout, the conversion between long 17-bit multiplication products and 8-bit inputs and outputs is rife with opportunities for overflow. In two's complement, overflow means rolling over from a large positive number to a large negative number, and visa-versa. Because the LMS update algorithm is essentially a partially stable negative feedback loop, large perturbations caused by computational overflow cause unsalvageable instabilities that necessitate filter reset. The problems outlined above are exacerbated when multiplied to many channels. The weight for each tap for each channel must be stored in memory, becoming a register file with $8 \cdot N \cdot N_{Channels}$ bits. This large memory must read and write fast enough to perform the necessary multiplications for each weight update. The multiple attempted solutions to these problems will be described in the implementation chapter.

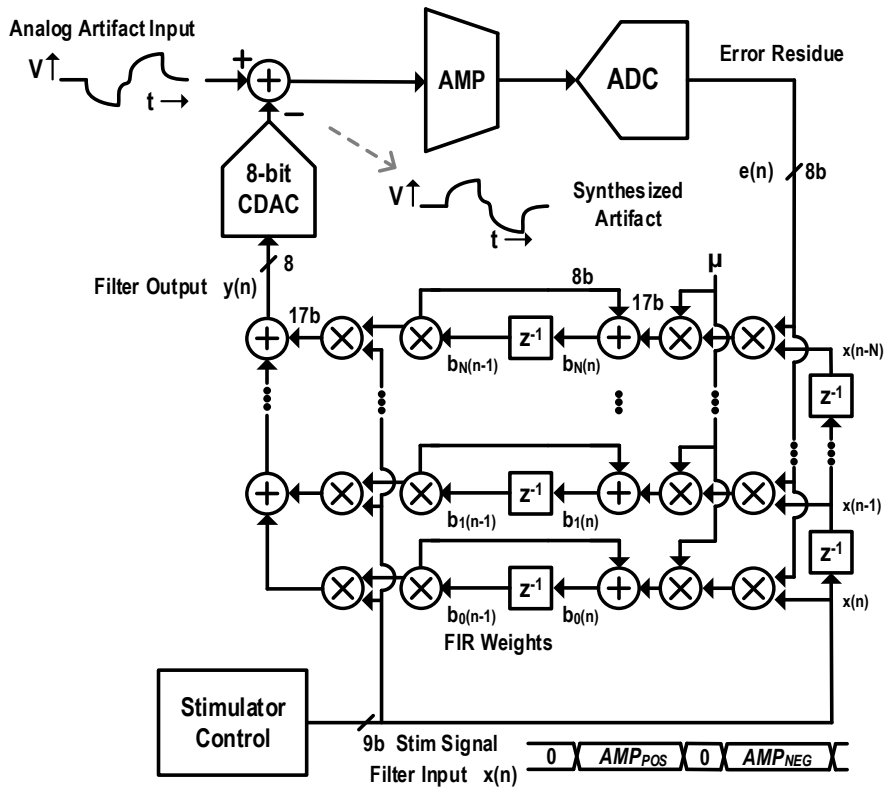

Figure 17: Block Diagram of a Full-Resolution Digital Realization of an Adaptive Artifact Cancellation Filter for a Single Channel

Chapter 4. IMPLEMENTATION

An adaptive LMS update artifact canceller was implemented in 65nm bulk CMOS alongside a ±12V-compliant H-bridge stimulator and a 64-channel, delta-encoded, time-domain multiplexed recording front end. The chip had fundamental design flaws with the artifact cancellation update method and multi-channel capability of the recording system. The attempt at a monolithic bidirectional interface was an important iteration of the design, verifying key components of the recording system and the adaptive canceller and proving out coexistence of high-voltage stimulation and sensitive recording on the same chip. The colleague responsible for recording later implemented a standalone version of the same recording system on its own CMOS die. The author assisted with testing and created a field-programmable gate array (FPGA)-based adaptive filter to interface with the new chip. This section will summarize the stimulator and recording systems that interface with the artifact canceller and outline the details of the system-on-chip (SoC) and FPGA implementations of the adaptive artifact canceller.

4.1 BIDIRECTIONAL NEURAL INTERFACE SOC

The fully bidirectional neural interface chip included an array of features that will be described in detail. A full block diagram is shown in Figure 18. The three main chip sections are the high-voltage H-bridge stimulator, designed by Eric Pepin, the delta-encoded, time-multiplexed recording front-end, designed by William Anthony Smith, and the digital adaptive artifact canceller, developed by the author. The recording front end also features a switched-capacitor common-mode artifact cancelling scheme. Each of the main integrated circuit blocks will be considered separately.

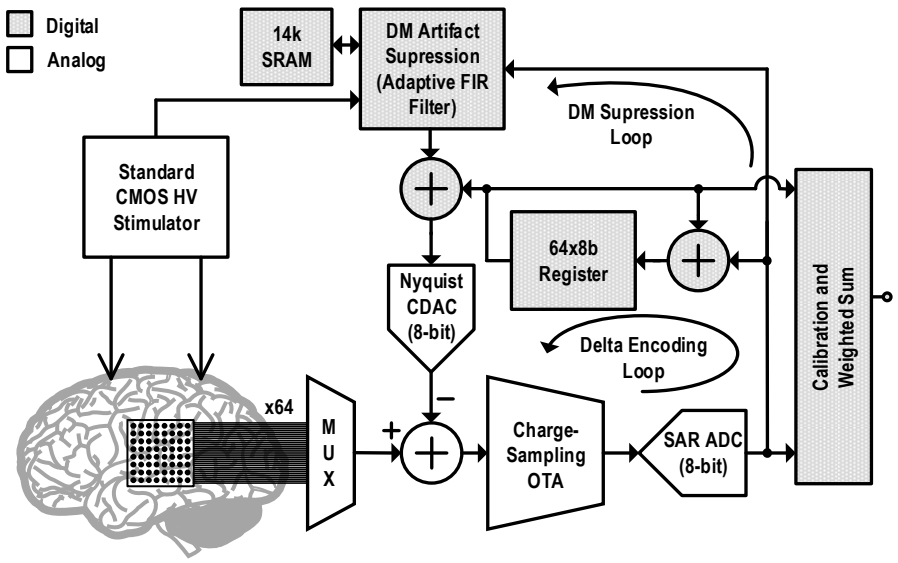

Figure 18: Bidirectional Neural Interface SoC Block Diagram

4.1.1 *High-Voltage-Compliant H-Bridge Stimulator*

The stimulator used to test and verify the proposed artifact cancellation system is an electrode-invariant, ±12V-compliant H-bridge topology designed in the TSMC 65nm GP CMOS process [22]. The stimulator delivers between 50µA and 2mA of biphasic, charge-balanced current pulses with active and passive discharge. A highly simplified block diagram of the stimulator is shown in Figure 19. The unique topology used in this design employs standard 1.2V and 2.5V CMOS devices in highly robust configurations to allow ±12 volts of swing at the electrode output terminals. High-voltage adapters (HVAs) act as current buffers to level shift the high electrode voltages to safe operating voltages for the sensitive, well-matched 1V current source. To save power during the stimulation process, dynamic voltage supplies (DVSs) track the voltage changes across the high-impedance electrode-tissue load, delivering only as much power as necessary to charge the tissue and maintaining the proper biasing to prevent current source saturation.

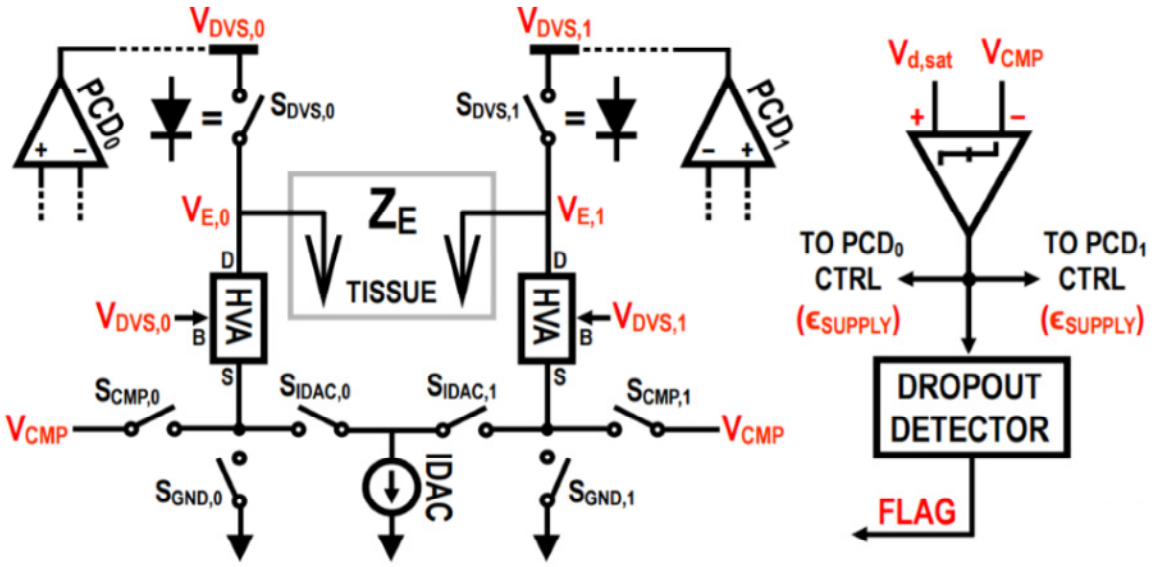

Figure 19: Simplified H-Bridge Stimulator Block Diagram

The electrode voltages are dynamically scaled with a negative feedback structure outlined in Figure 20. The positive current drivers (PCDs) are ideally represented as op-amps to clearly demonstrate the feedback architecture. As one side of the H-bridge sinks current through the electrode-tissue load (steps 2 and 5), a voltage drop across the load forces the sinking side voltage down and the supply side voltage up. To prevent the current source from saturating, a PCD is used in feedback to raise the supply side voltage to track ΔV across the load. The stimulator then follows a strict break-before-make scheme to avoid sudden transient currents. The previously sinking side is connected to ground while the previously sourcing side is connected to the current source (steps 3 and 4). The high-side switches are implemented with integrated diodes, as high-side CMOS switches face reliability and control complications. These diode switches are forced off by the same PCD tracking loop, forcing the supply voltages to track the electrode voltages exactly, forcing a voltage bias of zero across the diodes.

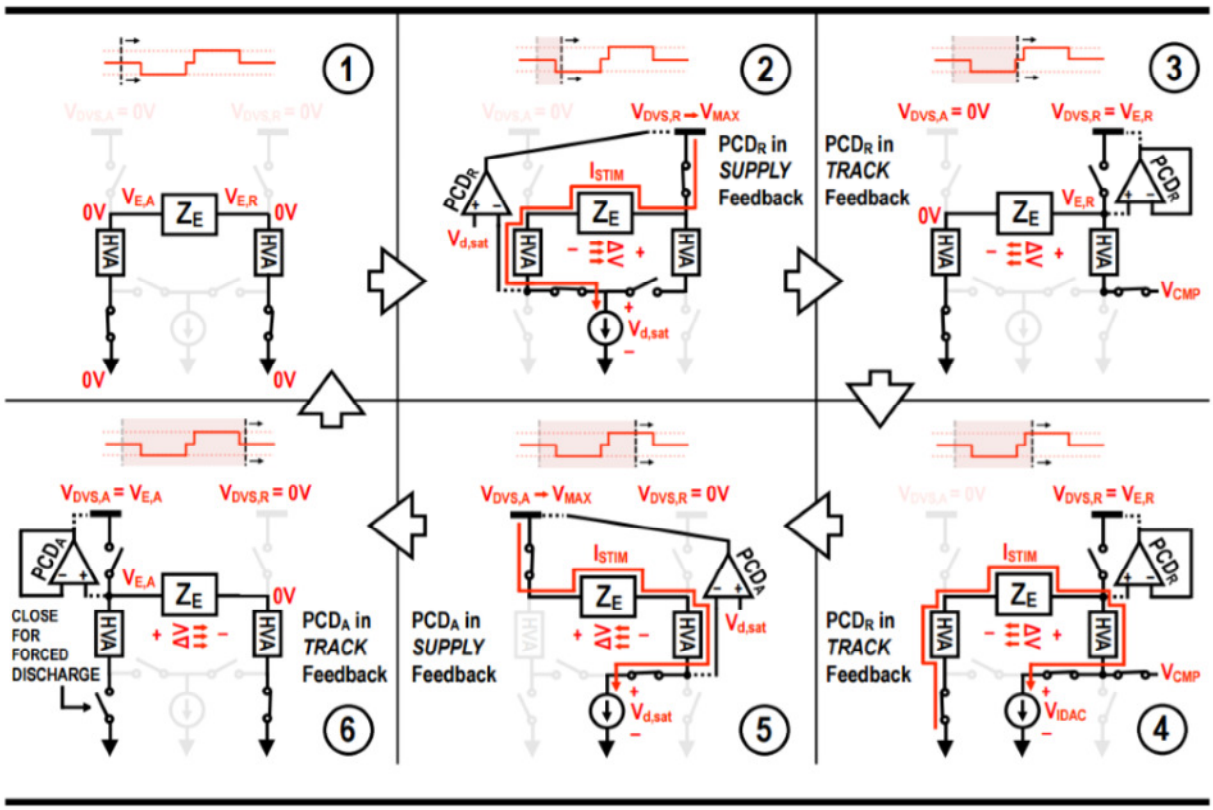

Figure 20: H-Bridge Stimulator State and Dynamic Voltage Supply Feedback Demonstration

The HVA circuits are implemented with stacks of 2.5V CMOS devices incrementally biased to accept a full 12V across seven stacked common-gate devices. Capacitive division of the PCD output voltage keeps the entire stack biased and protected during high-voltage stimulation events. The PCD circuits are comparator-based feedback control networks that feed clocks into the DVS. The bandwidth of the feedback architecture is sufficient to track rapid changes in the electrode voltages, such as the near-instantaneous $\frac{\Delta V}{\Delta t}$ presented by a purely resistive load. The DVS is a capacitive charge pump, whose capacitors dominate the chip area. To view the intricate circuit details of the stimulator, reference [22].

Use of a dynamic voltage supply solves one problem fundamental to traditional H-Bridge stimulators, the inherent bias of connected neural tissue to HVDD. In this case, the dynamic supply voltage tracks the necessary ΔV across the electrode-tissue impedance to supply the necessary

stimulation current. In contrast with the stimulation voltage profile shown in Figure 7, where the active electrode is driven down from HVDD while the sourcing electrode is held at HVDD, this H-Bridge stimulator follows a voltage profile similar to that shown in Figure 21. The resting bias to ground facilitates interaction between this stimulator and DC-coupled recording architectures.

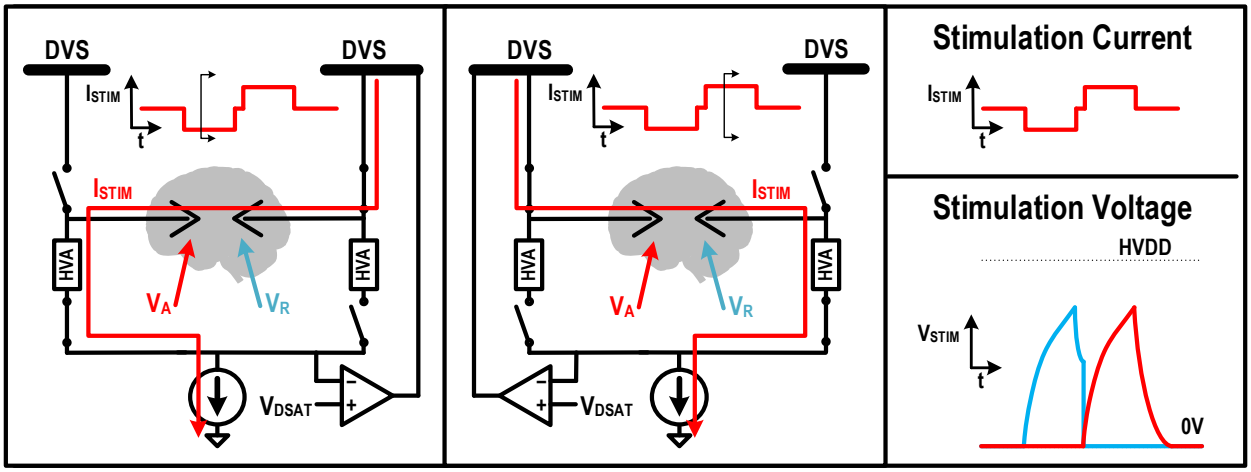

Figure 21: Dynamic Voltage Supply Effects on H-Bridge Stimulator Voltage Profile

This stimulator has been used extensively in benchtop and in-vivo tests. Both versions of the stimulator, the first a standalone SoC and the second an integration with the recording and artifact cancellation system, operate robustly and predictably. This is especially important when there is direct electrical coupling between the stimulator and recording electronics. Stimulators that are not well designed and verified have a chance of sourcing or sinking unwanted currents due to unreliable switching architectures. The resulting voltages are unsafe for any in-vivo subject and can also trip the electrostatic discharge (ESD) protection of the recording front end. The stimulator summarized above has been silicon verified to stimulate at high electrode voltages on the same

silicon substrate as sensitive recording electronics without reliability issues. A summary of one of the in-vivo verification experiments is shown in Figure 22.

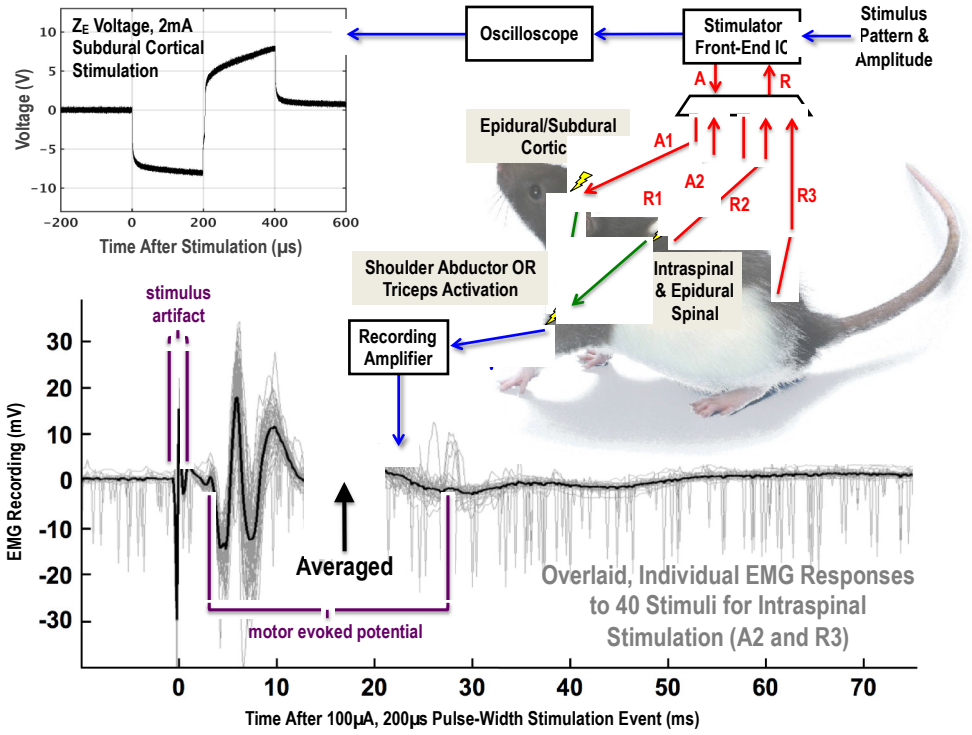

Figure 22: In-Vivo Demonstration of the H-Bridge Stimulator, Including Invoked Potentials

4.1.2 *Time-Domain Multiplexed, Delta-Encoded Digital Feedback Recording System*

The other critical component of a bidirectional neural interface SoC is the recording front end. This work leverages a highly scalable recording system that represents the state of the art in minimizing power and area consumption per recording channel. The recording front-end uses digital delta-encoded feedback to relax the noise and dynamic range constraints of the analog subsystems, taking advantage of the 1/frequency² spectral characteristics of neural signals. As the input signal slowly drifts due to large low-frequency signal content, a capacitive DAC adds or subtracts charge from the recording inputs to keep the low-resolution ADC in its dynamic range. This digital feedback scheme facilitates artifact cancellation by introducing a convenient injection

point for subtracting the synthesized artifact, at the point of digital-to-analog feedback, as shown in Figure 18. Furthermore, the time-domain multiplexed nature of the front-end can be extended to simultaneous artifact cancellation across many channels.

4.1.3 *Integrated Artifact Cancellation*

The first implementation of the proposed artifact cancellation system was integrated in 65nm bulk CMOS on the same die as the aforementioned stimulator and recording systems. The system consists of a full 32-tap adaptive FIR filter with 7 bits of FIR tap depth and time-interleaved adaptation and operation over 64 independent channels. In terms of dedicated hardware, the artifact canceller contains a dedicated processing block with multipliers and adders for each FIR tap update and calculation. Time interleaving is enabled by a custom 14.336-kilobit static random access memory (SRAM) used to store FIR taps. A complete block diagram is shown in Figure 32. This section outlines the detailed operation with description of each functional block.

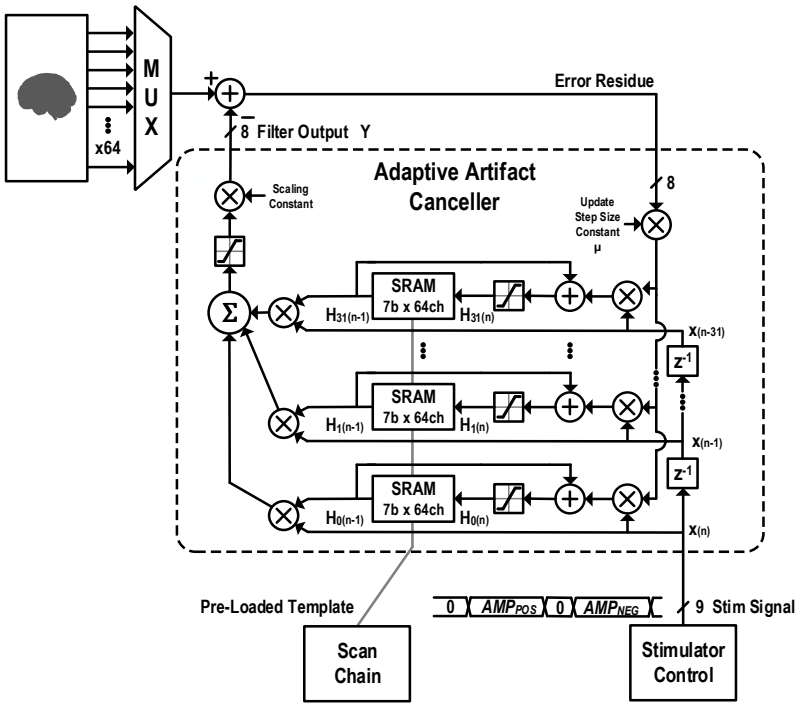

Figure 23: Integrated Adaptive Artifact Cancellation System Block Diagram

The heart of the adaptive canceller is the LMS weight update algorithm that estimates the channel response to a given stimulation pulse. In Figure 32, the update algorithm is represented by each row of in-line multipliers, adders, saturation blocks, and SRAM modules. In the integrated implementation, each FIR tap was designed with dedicated computation hardware, with 32 weight update modules in total. The block diagram representation above implements the following update function (derived from Methods Equation 3.27):

$$h_i(n) = \begin{cases} -63 & h_i(n-1) + \mu x(n-i)e(n) < -63 \\ h_i(n-1) + \mu x(n-i)e(n) & -63 \le h_i(n) \le 63 \\ 63 & h_i(n-1) + \mu x(n-i)e(n) > 63 \end{cases} \quad 4.1$$

This function saturates at weight codes of -63 and 63 due to the resolution limit of 7-bit two's complement numbers. Care was taken to not allow overflow of the adder and multiplier outputs. The resulting flip from a large positive number to a large negative number (or visa-versa) would cause unstable oscillations on every overflow, requiring frequent reset. The saturation operation was explicitly implemented with combinational logic synthesized from behavioral Verilog. It was later found that allowing the filter taps to saturate is not a desirable function. Saturation indicates that the voltage artifact is outside of the full-scale range of the filter output, the capacitive DAC. The range of feasible ADC operation on the margins of the CDAC dynamic range is narrow due to high amplifier gain, so operating exactly at the weights' full range will likely saturate the recording channel. Alternate methods to saturation were included in the FPGA implementation, which will be described shortly.

The weight update and FIR functions are time-interleaved between 64 channels, which is enabled by the dedicated SRAM. The on-chip memory is organized in sixty-four 224-bit blocks, corresponding to seven-bit weights for all thirty-two FIR taps. The eight-transistor SRAM was

custom-designed in schematic and layout with individual bit-cells optimized for power consumption, read/write stability, and compactness. A snapshot of the custom cell layout is shown in Figure 24. The read and write indices are driven by delayed versions of the current channel pointer generated in the recording logic. The read index follows the current recording channel and loads the weight inputs to the FIR filter. A version of this index is delayed by one time-interleaved period and used as the memory write pointer. By latching the ADC error signal and corresponding input data at the previous interleaving period, having a trailing write pointer allows the FIR update logic to compute for a full 1/128kHz=7.8125µs, more than sufficient time for the multiplier, adder, and memory write critical path. A layout editor snapshot of the full 14.336 kilobit memory is shown in Figure 25.

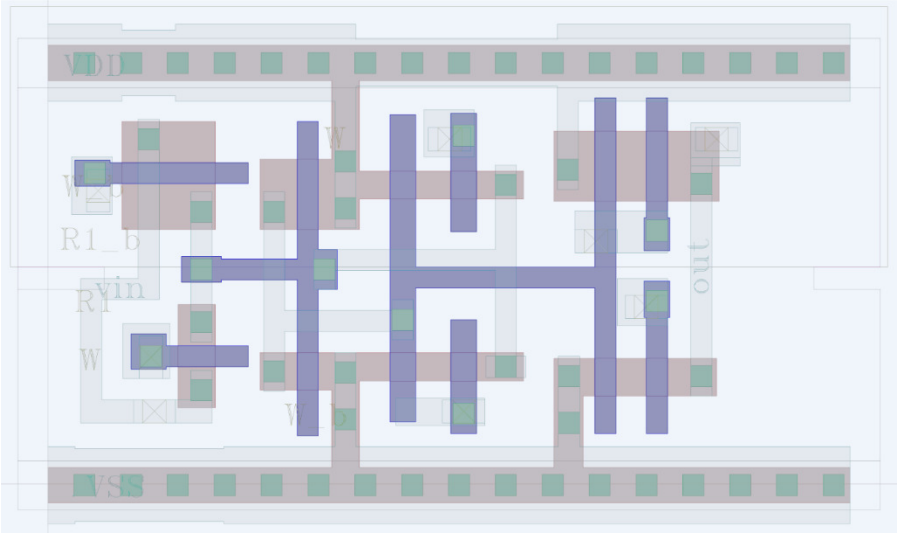

Figure 24: Custom Eight-Transistor SRAM Cell Layout

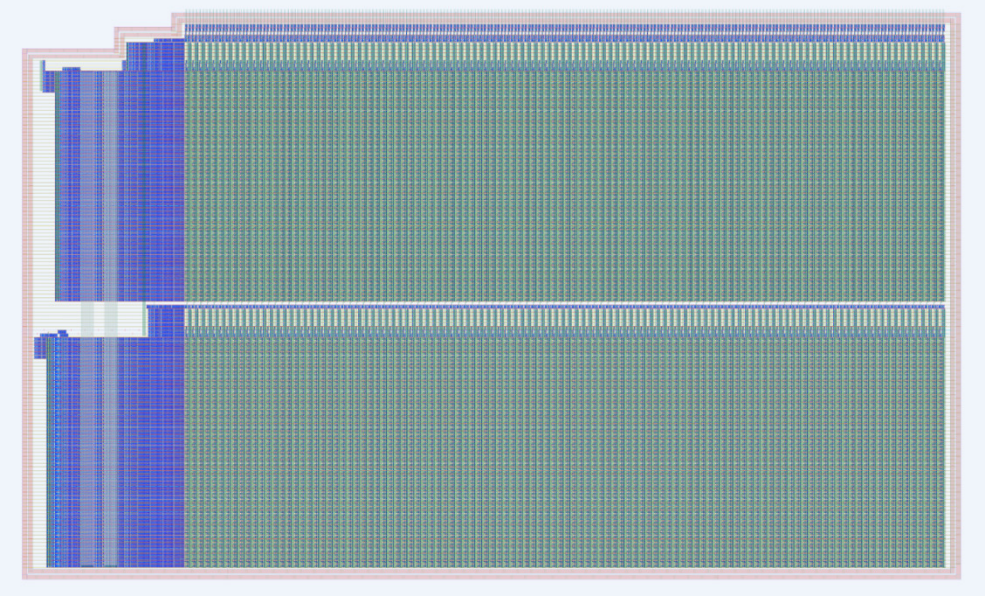

Figure 25: Full 14.336 Kilobit SRAM Layout

In the LMS weight update algorithm, the step size parameter, μ, must be set appropriately to reduce the destabilizing effects of transient noise. As identified in Chapter 3, this parameter must be the inverse of the total noise power in order to ensure categorical stability. In the recording system described above, the background neural "noise" is close to the ADC full range, taking up 5-6 bits of the 8-bit range. For this reason, μ must be a very small fraction. In digital computation, arbitrary division is a prohibitively expensive operation, requiring floating point arithmetic and wide bus widths. To overcome this bottleneck, the division is implemented with an arithmetic bitwise shift. This only allows division by powers of two while significantly decreasing logical complexity. The coarse step size parameter is sufficient to achieve the LMS stability criterion by scaling the weight update steps by orders of magnitude to match the overall noise power on the error signal.

There are four main interfaces between the artifact canceller and the other bidirectional interface blocks. The error signal coming from the ADC is an 8-bit, two's complement bus that is used exclusively as the error estimate for training the FIR filter taps. This signal is latched upon

receipt of a valid data signal from the ADC. It is then held for use in computation until the next valid set of ADC data. The output of the FIR filter goes to the capacitive DAC of the recording system after undergoing similar saturation protection to that in the weight update module. This 8-bit, two's complement bus is summed with the recording system's accumulator value and then asserted on the CDAC at the beginning of every recording cycle. The recording system has a tunable delay between the system clock and the logic-generated valid signal (~100ns). This delay allows time for weights read from memory to propagate through the multipliers and adder of the FIR filter for proper evaluation by the time of CDAC assertion. Finally, the input signal to the FIR filter and LMS weight update comes from the stimulator controller. In this SoC implementation, the stimulator was controlled by an off-chip state machine embedded in an FPGA. The state machine outputs were separately decoded into a simplified representation of the stimulator pulse. Two 1-bit signals, pos_stim and neg_stim, were sent in through dedicated digital pads to indicate the timing and duration of the positive and negative stimulation pulses. The positive pulse and negative pulse amplitudes were pre-loaded through the scan chain, and a decoder within the synthesized canceller logic parsed the timing signals and amplitudes into a 9-bit representation of the stimulator current pulse. This signal is sent through a digital delay line to the corresponding weight update blocks and FIR tap points.

The on-chip scan chain is used to load static operating parameters and initial conditions. A list of the static scan parameters and their functions is given in Table 1. The SRAM has a built-in capability to pre-load a set of FIR weights as a starting point for the FIR filter or a predetermined artifact template. This was extremely useful for debug purposes and enabled continued experimentation upon failure of the LMS update module.

Table 1: Integrated Canceller Scan Parameters

Parameter	Bit Width	Function
tap_num	5	Number of enabled taps (1-32)
weight_in	224	Scans in arbitrary FIR template
pos_amp	8	Amplitude for stimulator positive pulse decode
neg_amp	8	Amplitude for stimulator negative pulse decode
mu_shift	3	Bitwise shift parameter. Division from ½ to 1/128
out_shift	8	Output scaling parameter. Two's complement bitwise shift
scanmode	1	Enables scan in of FIR template
scan_channel	6	SRAM channel for template scan in
cm_mode	1	Enables CM canceller only during stim pulse (1) or always (0)
cm_enable	1	Enables CM canceller
cm_delay	7	Length of CM canceller enable after stim onset

Appendix 1.A. contains the full Verilog code of the synthesized LMS update and FIR filter blocks. All multipliers and adders were implemented with Synopsys DesignWare blocks, pre-optimized for area and power consumption. A die shot of the implemented SoC can be seen in Figure 26. The on-chip memory and digital logic take up approximately 610µm by 420µm of area. The large memory is entirely necessary, but the LMS logic (multipliers and adders) could be optimized via multiplexing and pipelining.

The SoC implementation of the LMS algorithm was unable to converge to a stable cancellation solution in noisy neurological experiment environments. On a controlled benchtop with ideal input signals, the input noise power was sufficiently low to allow filter convergence. However, relying on bitwise shift-based division did not allow a small enough step size parameter to mitigate the effects of large transient noise caused by 60Hz interference and physical manipulation of electrodes during an in-vivo experiment. The specific challenges and their

solutions are explained in the next section. The FPGA-based prototype addresses this issue with additional noise mitigation methods.

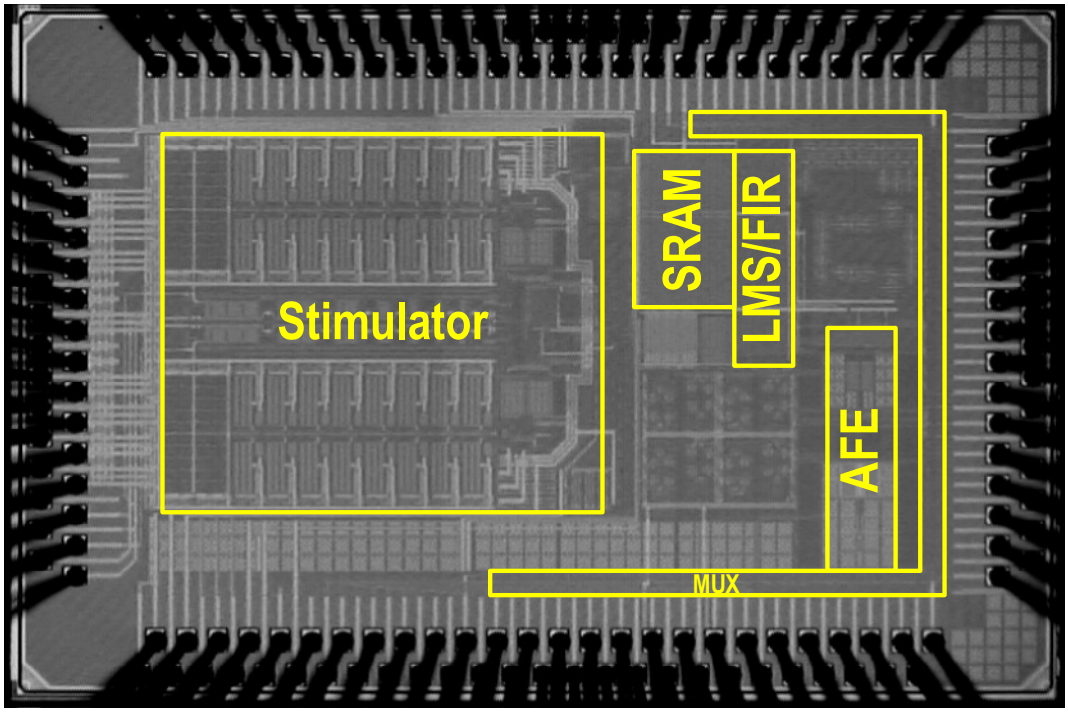

Figure 26: Bidirectional Neural Interface SoC with Integrated Artifact Cancellation Die Photo

The fully integrated bidirectional interface was taped-out in TSMC 65nm bulk CMOS on a two millimeter by three millimeter die. The chip has a total of 164 staggered bondpads, with the majority taken up by 32 fully differential recording inputs. It was bonded to a 17x17 pillar grid array (PGA) package to accommodate for the high pin count and ease part swapping during test. The author designed a prototype printed circuit board (PCBs) for chip testing and verification. The board, shown in Figure 27, contains 24 independent voltage regulators for supply isolation, optoisolation on all digital control lines (connected to an FPGA, not shown), simulated electrode-tissue interface loads, breakouts for arbitrary discrete loads, an on-board oscillator-based frequency reference, fully differential 16-bit DACs for generation of arbitrary input vectors, and a

DAC for generating an analog visualization of the recording system output. The board was fabricated with six layers, largely for power routing, and the majority of the board is taken up by connectors for the recording inputs and stimulator outputs.

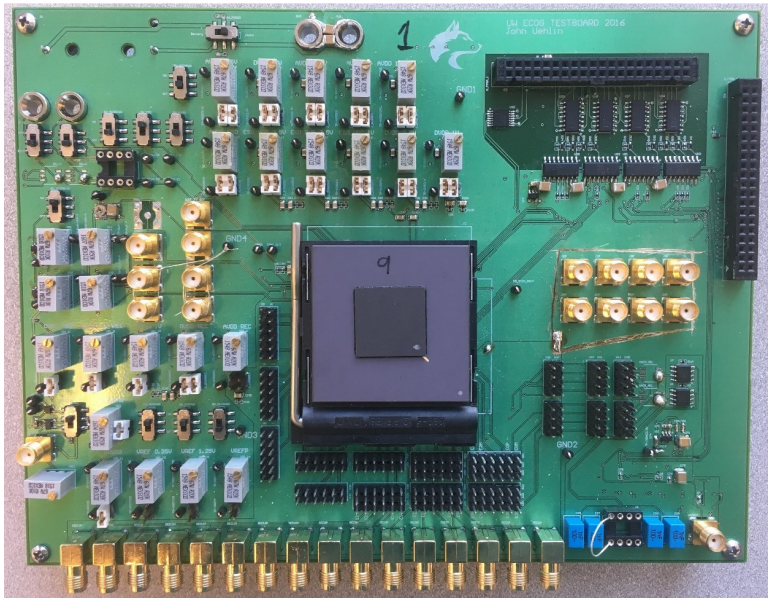

Figure 27: Bidirectional Neural Interface SoC Testboard

4.2 FPGA-Based Artifact Cancellation Testbed

The SoC implementations of the recording system and artifact canceller had fundamental design flaws that prevented recording on multiple channels and convergence of the LMS update algorithm. Specifically, stabilizing methods in the filter update algorithm were insufficient to compensate for the large transient noise present in neural recording environments. In order to move forward and develop robust artifact cancellation methods, a revision of the integrated recording system enumerated above was fabricated on its own CMOS die. This implementation includes a serializer interface for injection of an artifact cancellation signal into the CDAC data path. This allows rapid prototyping of an off-chip adaptive artifact canceller to test numerous adaptation methods with a real analog interface.

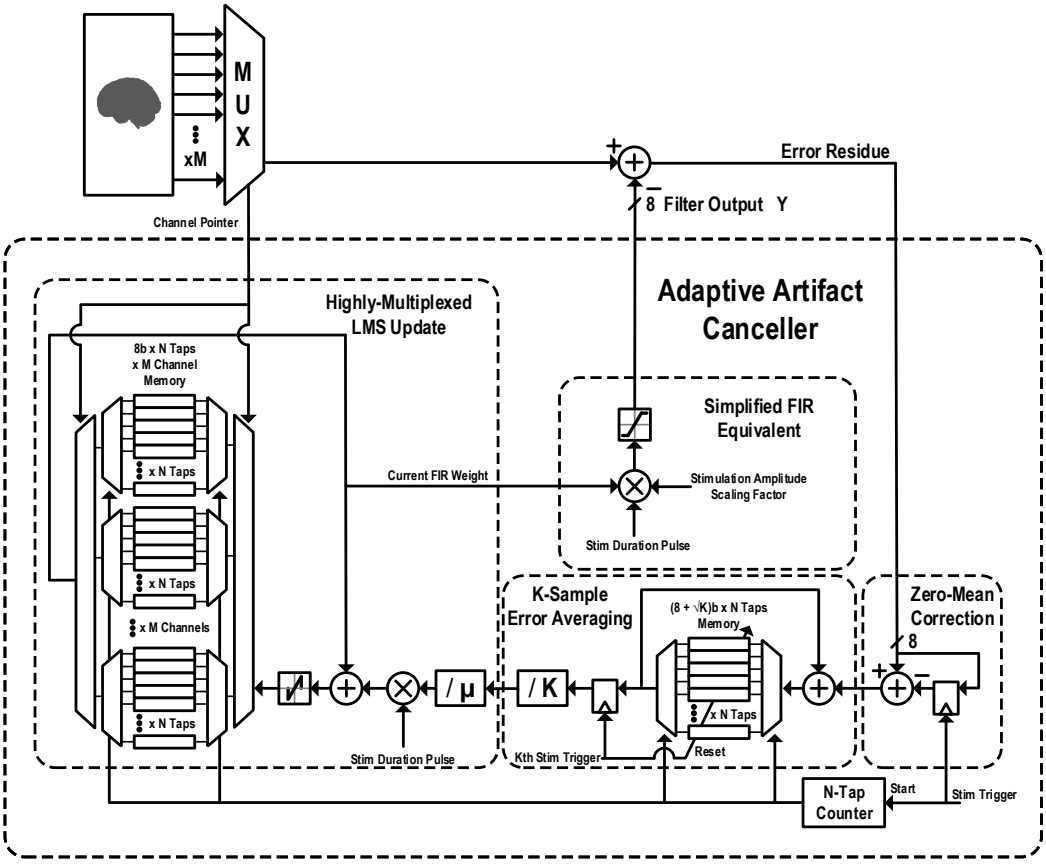

Figure 28: FPGA-Based Adaptive Artifact Canceller Block Diagram

A canceller has been integrated into an Altera Cyclone V FPGA, which was already being used to test the recording chip. This canceller includes significant improvements in algorithm modularity and several advanced noise mitigation techniques, reducing system complexity and achieving robust LMS algorithm convergence. A functional system diagram is shown in Figure 28, with details regarding system timing in Figure 29. The following discussion outlines and explains the features of the improved canceller. Assume for this discussion that the recording and stimulator systems are identical to those outlined in the SoC implementation section.

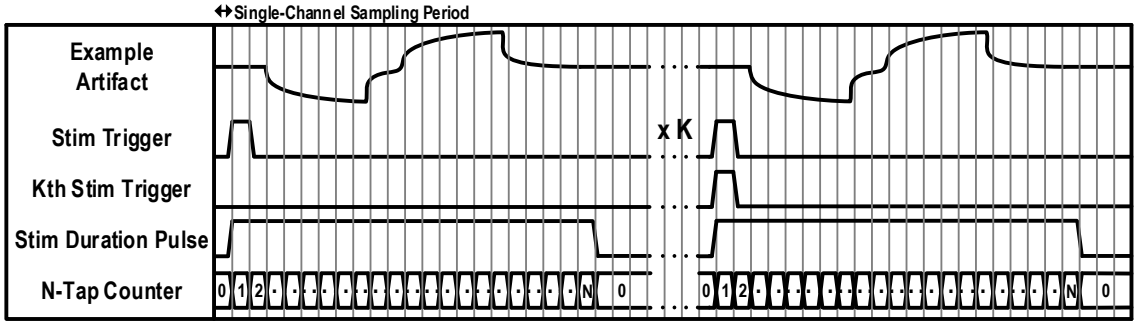

Figure 29: Timing Diagram for the FPGA-Based Artifact Canceller

4.2.1 *Highly-Multiplexed LMS Update and the Impulse Response*

System complexity is significantly reduced by migrating from calculation of the neural channel's response to the full stimulation pulse to calculation of the simple impulse response. Treating the stimulation pulse as an impulse, a single non-zero input sample per stimulation pulse, allows calculation of each FIR tap weight separately. In the previous implementation, use of a multiple-sample stimulator input signal necessitated parallel update of multiple weights at once and dedicated, parallel computation hardware. Calculating the impulse response can be done with a single multiplier and adder time-domain multiplexed amongst the multiple FIR taps, as the impulse response taps are independent of one another. Additionally, the stability criterion outlined earlier is relaxed dramatically; a noisy error signal only affects the output signal via one tap at a time. The multiplexing operation is performed at the inputs and outputs of the weight storage memory, inside of the LMS update feedback loop. First, the signals are multiplexed to dedicated memory banks for each channel, based on the recording system's channel pointer. Second, each channel's memory bank has its own multiplexer and de-multiplexer to select the current tap's memory element.

Using the impulse response further simplifies the FIR filter module, allowing equivalent operation with a single multiplier, as opposed to a multiplier for each tap and a large adder at the

output. Rather than a tapped delay line, a single signal, held high for the duration of the stimulation pulse, is multiplied with the de-multiplexed FIR tap weights. As the de-multiplexer cycles through the FIR taps based on a counter triggered at the onset of each stimulation pulse, the tap weights progress through the equivalent of a tapped FIR delay line, without multiple multipliers or a physical delay line. At the output of this single multiplier, a scaling factor is applied to compensate for changes in the stimulation current amplitude after training of the tap weights, with saturation protection at the final output of the multiplier to prevent instability-causing numerical overflow.

The saturation blocks previously inserted into the LMS update path were found to encourage system instability is certain boundary cases. In the SoC implementation, filter taps saturated at +63 or -63 fully exercised the CDAC dynamic range, which stuck at full range in cases of filter instability. In most cases, large scale erroneous codes obfuscated the recording system's digital feedback loop containing large-amplitude, low-frequency neural data. By taking up the full CDAC range with artifact cancellation data that, during training, was not necessarily correct, the artifact canceller prevented the CDAC from applying feedback from the digital loop. Additionally, exercising a full-scale CDAC code creates cross-talk between channels and tap samples without excessive auto-zero settling times. This cross-talk due to a saturated FIR tap weight influences adjacent error samples, which could cause unstable oscillations through positive feedback. To prevent these potential errors, the saturation block is replaced by a zero-upon-saturate block, which resets the weight value as soon as it reaches full scale. This is acceptable because a full-scale tap represents an artifact amplitude outside of the CDAC dynamic range, which is outside of the system's cancellation abilities. Zeroing the weight prevents crosstalk due to large CDAC codes and resets instabilities indicated by "stuck" taps.

4.2.2 *Zero-Mean Correction*

The LMS algorithm outlined in Chapter 3 makes several assumptions about the input signals to achieve a tractable update procedure. The most important assumption, fundamental to the stochastic descent nature of the algorithm, is that the additive channel noise is uncolored (has uniform spectral power) and has a zero mean. The noise present in a neural recording system, by nature, does not fit either of these qualifications. Neural signals, the "noise" in this case, has a red or brown spectral characteristic; that is, noise power rolls off by $1/freq^2$. Because the neural noise has high signal power at low frequencies, there are large, slow-moving offsets present at all times.

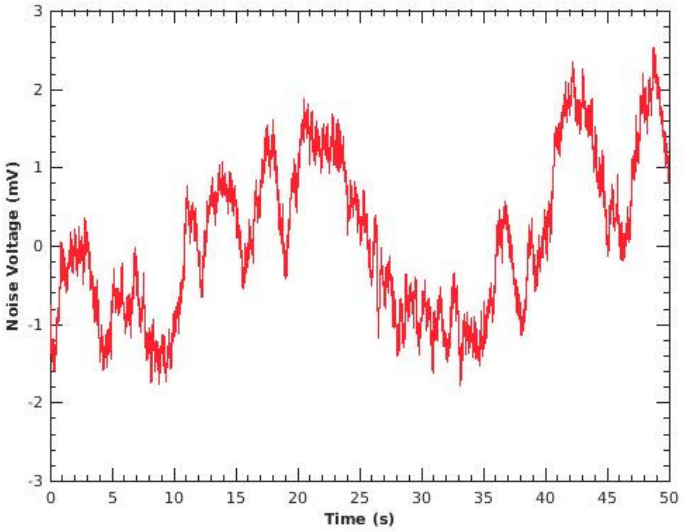

Figure 30: Example Red Noise (Zero Mean over a 50 Second Window)

The window during which the canceller samples the noise as an error signal (the duration of the stimulation pulse) is not infinite in length. In reality, the update window is at most several milliseconds. During this window, the large, slow offsets create a non-zero mean, although the signal mean may truly be zero for an infinite window. Figure 30 shows 50 seconds of zero-mean red noise sampled at 2kSamples/s. To demonstrate the offset, Figure 31 shows a 1-second window

of the red noise, exemplifying how the offset-causing effects of shaped noise worsen as the window is constricted.

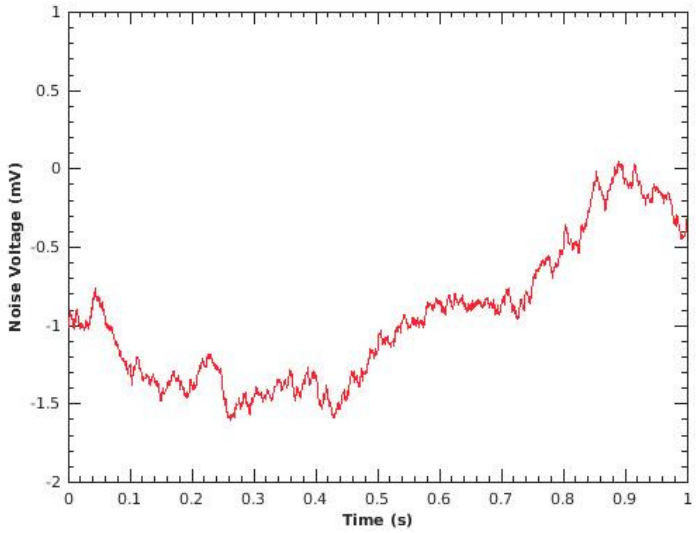

Figure 31: Example Red Noise (1 Second Window Exemplifying Offset)

Although the digital feedback accumulation of the used recording system compensates for some of this slow-moving offset, there is still a significant amount of accumulated low-frequency noise left over within the ADC dynamic range. Figure 32 and Figure 33 demonstrate the remaining offset in the presence of low-frequency digital feedback. Figure 32 shows a simulation of 20 seconds of zero-mean red noise with a standard deviation of two millivolts, scaled to the equivalent ADC codes. Notice significant drift over the course of twenty seconds. Figure 33 shows half a second of the same noise after application of digital feedback. As a reminder, the CDAC adds or subtract charge from the input of the recording chain upon the ADC code crossing a predetermined threshold. To generate this figure, the accumulator subtracts 100 ADC codes upon crossing of a +64 LSB threshold and adds 100 ADC codes upon crossing -64. These arbitrary values are sufficient to demonstrate the offset present in a slow-moving noise environment. Even after digital

feedback, small windows of the ADC input still see significant offsets while the noise moves slowly within the ±64 LSB thresholds.

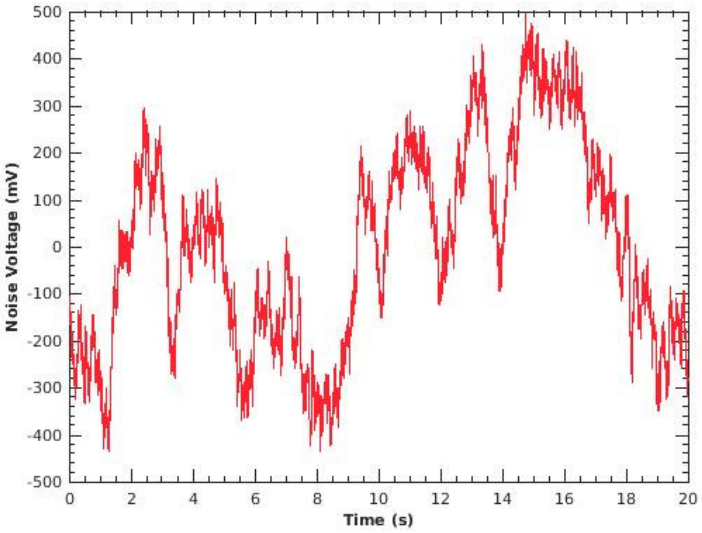

Figure 32: Example Red Noise Scaled into ADC Codes

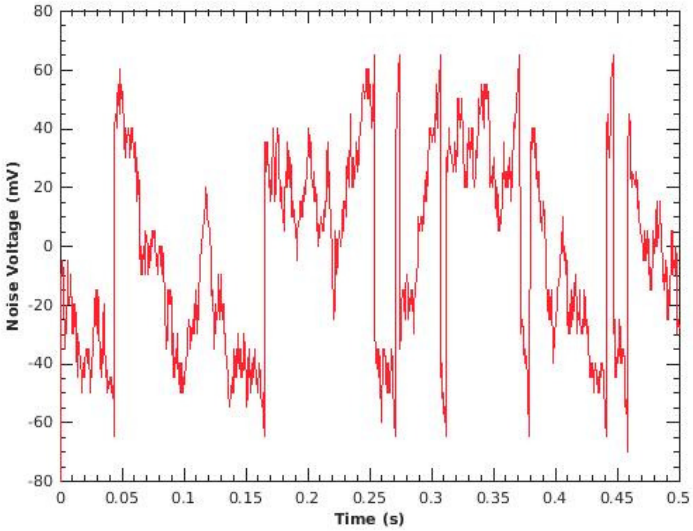

Figure 33: Red Noise at the ADC Output with Digital Delta-Encoded Feedback

To correct for the non-zero mean over small windows of colored noise, the improved artifact canceller conditions the window of noise to negate the effect of slow-moving offsets. By taking the ADC sample directly before the onset of stimulation and subtracting it from subsequent samples during the stimulation window, built-up low-frequency data not entirely captured by the

digital feedback accumulation is prevented from reaching the LMS algorithm and affecting adaptation. A simulated demonstration is shown in Figure 34. The error input comes directly from the simulated ADC; it includes sampled red noise and artifacts recorded in-vivo with a commercial benchtop recording system with a high dynamic range. As can be seen in the raw error input, there is an offset of approximately -25 ADC LSBs at the onset of each stimulation pulse. This offset is captured prior to each pulse and subtracted from the error input, resulting in the mean-corrected error in the second row. The resulting error signal used for LMS algorithm adaptation now has approximately zero mean for the duration of the stimulation window.

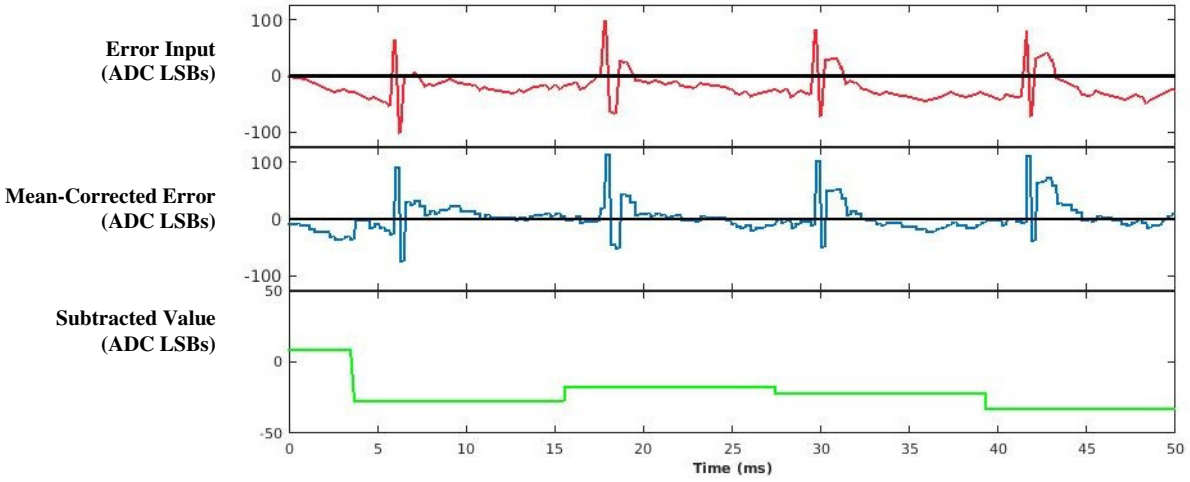

Figure 34: Zero-Mean Offset Correction Example

4.2.3 *Averaging*

As previously described, the neural environment is extremely noisy, especially when using an ADC in digital feedback. This makes it particularly difficult to achieve the stability criterion identified in Eq. 3.27. With increasing error magnitudes, the convergence step size must be decreased. With a custom digital integrated system, power and area limitations restrict computational complexity. As a result, dividing by a large number to achieve a small step size truncates a significant degree of information from the weight update algorithm. This slows

convergence and creates a lower bound to the step size parameter. The limitations of bitwise shift-based division make it unsuitable for the noisy neural environment.

There are physical considerations in realistic neural recording environments that contribute significant transient noise. The cramped, equipment-filled conditions of neurological research laboratories radiate significant amounts of 60Hz power line noise, with up to volts of signal accumulating in neural tissue. This signal is common to all tissue, so differential recording mitigates a majority of the power, however common-mode-to-differential-mode conversion still contributes several millivolts of 60Hz noise to differential recording. Furthermore, the electrodes and connectors used to interface with the brain are delicate and, in most research environments, ad hoc. Any real neurological experiment is done with a live subject that may be moving freely. As the subject moves, electrodes may brush against one another, the electrode-tissue interface may temporarily become an open, etcetera. These events cause sharp spikes in the neural recording. A short may completely ground one or more samples. An open may accumulate charge and saturate the recording amplifier for a sample. These spurs, which are even more common in mixed stimulation/recording applications, can throw off the LMS adaptation algorithm if they occur during the stimulation pulse window. An example is shown in Figure 35, which consists of actual *in-vivo* data collected in a study of stimulation voltage artifacts. It was collected with the mentioned recording system in a low-gain setting (voltage gain of 10 as opposed to the normal 2000 during neural recording). The full artifact voltage was captured in approximately 40 ADC codes. The periodic spikes with peaks of approximately 30 LSBs are the artifacts. Notice the near full-scale spikes in the center of the second-long recording. It was found latter that one electrode was barely connected and would become an open connection when our primate subject moved excessively. The primate was awake, playing a hand movement game. He would frequently kick against his

restraints violently, so it was no surprise that the electrode connections were disturbed. These large transient spikes only last for one or two ADC samples, but they were sufficient to destabilize the LMS adaptation algorithm and render the artifact cancellation experiment for that day a failure.

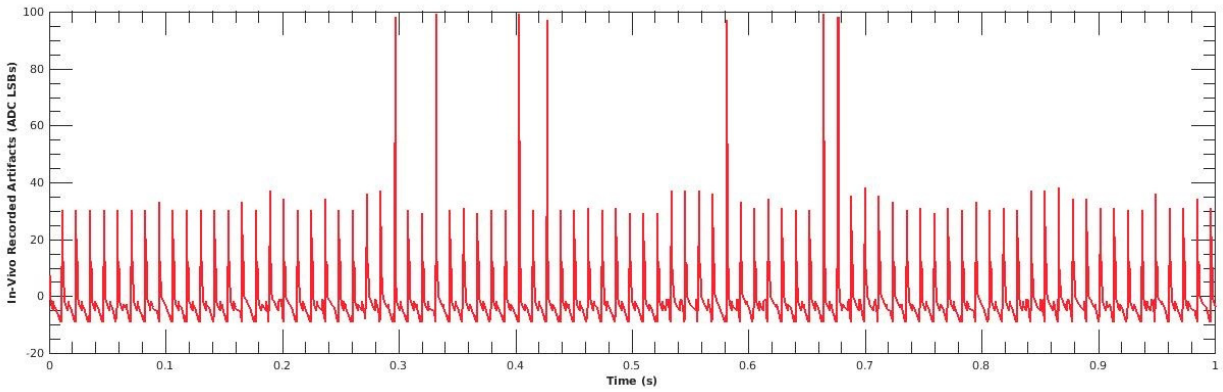

Figure 35: In-Vivo Recording of Voltage Artifacts, Including Transient Spikes due to Electrode Opens

In order to reduce the apparent noise power and mitigate the effects of transient spikes, the improved artifact canceller averages subsequent samples of a given time index of the voltage artifact to create a cleaner representation of the residual error signal. The averaging reduces the magnitude of any present zero-mean noise and smooths any aperiodic spikes that may occur in the recording. As shown in Figure 28, averaging is implemented with a multiplexed accumulation register. The register has a memory cell for each tap of the FIR filter of bitlength $8 + \sqrt{K}$ for an 8-bit error signal averaged over K samples. The accumulator adds K (where K is a power of 2) samples in the accumulator and then divides the signal by K via bitwise shift. The memory required for this operation is equivalent to adding a single extra channel to the large system memory, making it relatively inexpensive. However, the adaptation algorithm using averaging must be done one channel at a time to prevent it from becoming prohibitively expensive. To adapt using averaging for all channels simultaneously, the accumulator for each tap on each channel would effectively double the necessary memory space.

Sending the artifacts in Figure 35 through a simulated artifact cancellation system readily shows the benefits of averaging in the presence of large transient perturbations. The spikes in the example recording are aperiodic and occur in intervals as widely spaced as 0.5s. Without averaging, the filter coefficients are easily perturbed, especially when the step size parameter, μ, is set at a large value to achieve rapid convergence. The result is extreme instability, as seen in the error signal shown in Figure 36 at $t = 0$, when the step size parameter is changed from $1/64$ to 1/2.

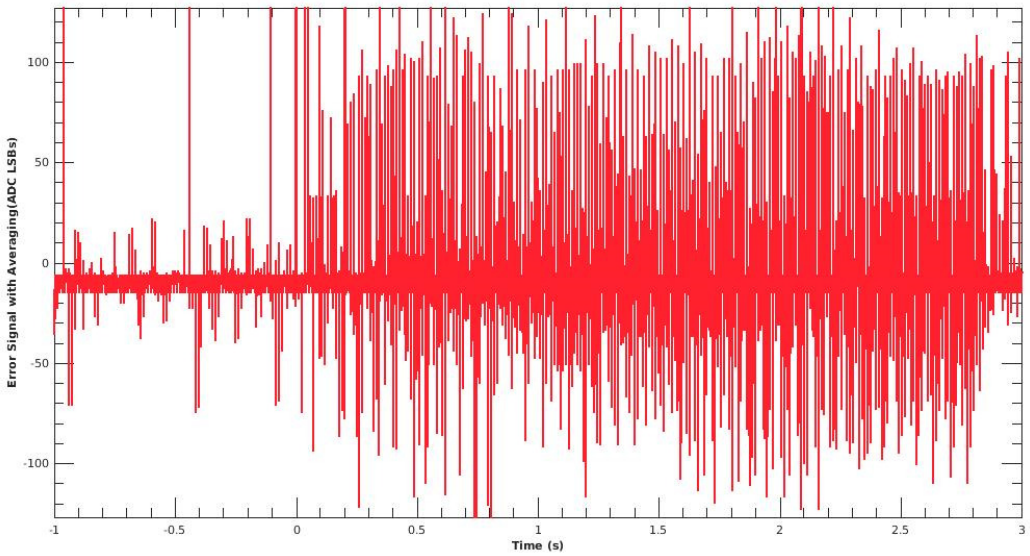

Figure 36: LMS Algorithm without Averaging Destabilized by Transient Spikes

Running the exact same transient data through the LMS algorithm with averaging over eight error samples gives the resulting error shown in Figure 37, where the algorithm has found a stable minimum error solution that is minimally perturbed by large transient spikes. Note that for the presented simulation data the filter was pre-trained to the same ideal set of weights, resulting the in the minimum square error. The convergence parameter, μ, is set to ½ at $t = 0$ in the above

plots. The transient spikes readily apparent in Figure 35 and Figure 36 destabilize the filter in Figure 36 due to violation of the convergence criterion identified in Equation 3.27:

$$\mu\, N \max \left| \frac{e(n)^2 \cdot T}{\int_{-\frac{T}{2}}^{\frac{T}{2}} e(n)^2} \right| \leq 1$$

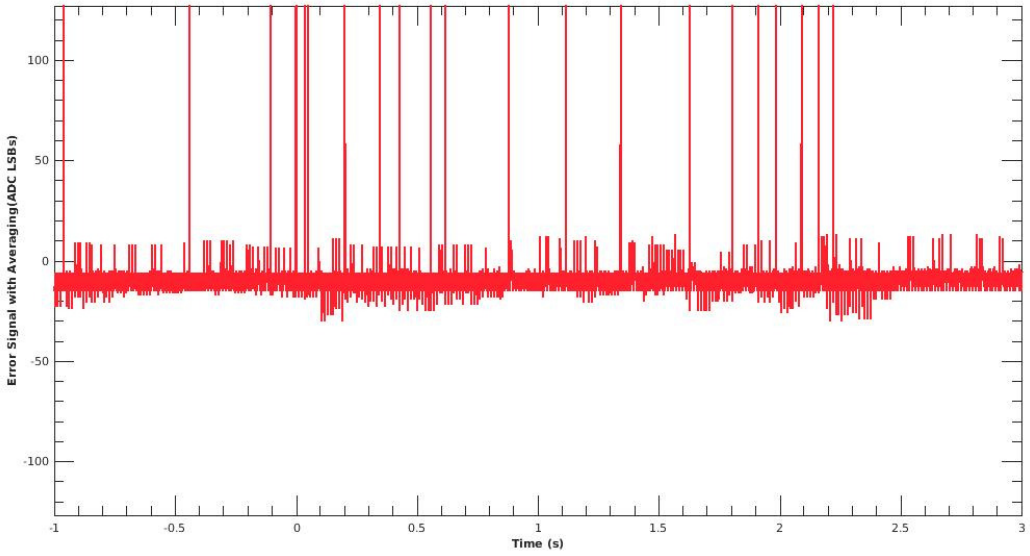

Figure 37: LMS Algorithm with Averaging Maintaining Stability in the Presence of Transient Spikes

How the two approaches fit into the convergence criterion will now be explained in detail: In the reduced-complexity LMS update algorithm, using the impulse response entails use of only one filter tap at any given instance in time. This equates to $N = 1$ in the above equation. Integrating the raw error signal prior to cancellation to calculate the average error results in $\frac{\int_{-\frac{T}{2}}^{\frac{T}{2}} e(n)^2}{T} \cong 144$ $(\text{ADC LSBs})^2$, equivalent to an average error of ±12 LSBs. This average is over ten seconds of stimulation at 100 pulses/second, with artifact amplitudes approaching 40 LSBs peak-to-peak and frequent aperiodic saturation events at 127 LSBs. The difference between the system with and without averaging is the apparent $\max\big(e(n)\big)$. The single-sample transient spikes of 127 LSBs is

averaged among eight samples. When the error signal reaches the update loop input after averaging, a single-sample event will be reduced in magnitude by a factor of eight. Inserting this information into the stability criterion (with $N = 1$ and $\max\big(e(n)\big) = \frac{127}{8}$):

$$\mu \max\left|\frac{\left(\frac{127}{8}\right)^2}{144}\right| \leq 1 \rightarrow \mu \leq 0.571 \tag{4.1}$$

A μ of 0.5, with averaging, satisfies this criterion. Averaging dramatically increases filter stability by acting as a low-pass filter on the additive noise while keeping the artifact error signal intact, as it more or less consistent between averaged samples. This low pass filter is most effective at suppressing spurious error events that occur over few samples, but the low-pass action also reduces the effect of additive zero-mean Gaussian or colored noise.

4.2.4 *FPGA Implementation Details*

As with its integrated version, the FPGA-based canceller is defined in Verilog code. The complete code, including interface and memory elements is included in Appendix 1.B. This prototype code includes the memory required for one channel of cancellation with a 128-tap FIR filter having 8 bits of weight resolution. The 128-tap filter enables experimentation with a variety of different stimulation pulse shapes and durations. Specifically, the ability to cancel artifacts that cover many ADC enables testbench experiments at higher sampling rates, with 128kS/s being the peak rate of the accompanying recording system. The artifact canceller interfaces with the recording system through an on-FPGA serializer. Each ADC sample has a corresponding synthesized artifact sample coming out from the canceller. The serializer/deserializer interface has an intrinsic latency of two samples; this delay is compensated by leading the read operation that feeds the FIR filter by two samples. The FIR filter taps and error averaging data are stored in a register file on the FPGA. For larger, multi-channel operation, macros will be used to compile

large RAM banks. Also integrated onto the FPGA is a stimulator controller used to synchronously operate a standalone version of the previously identified H-Bridge stimulator. Including the recording and stimulator interfaces on the same FPGA as the artifact canceller greatly reduces system complexity and aides visibility and access. The specifics of the combined two-chip/FPGA setup will be described in detail in the Results chapter.

Chapter 5. RESULTS

This chapter covers bench testing of the designed artifact canceller and canceller performance when tested with in-silico recording and stimulator systems in the analog domain. The specific methods used to collect data will be described in detail. Section one covers the specific testing setup and the corresponding considerations. Section two contains the results and performance measurements. Section three compares the presented artifact canceller with the state of the art and gives perspective on competing systems' performance numbers.

5.1 TESTBENCH SET-UP

Creating a robust system for testing artifact cancellation presented several difficulties. Surprisingly, the artifact canceller itself was the least of the many concerns involved with integrating recording and stimulation. Because the canceller was implemented entirely in Verilog on the Altera Cyclone V FPGA, it could be debugged and verified in its entirety in simulation. A few race conditions became apparent in the compiled and assembled implementation on the FPGA. These problems were traced back to incongruous state machine design and were eliminated by revising code to rely on a single synchronous clock.

Figure 38 shows a block diagram of the full testing setup. Separate chips and test boards were used for the H-Bridge stimulator and highly multiplexed recording front-end. An earlier version of the H-Bridge stimulator chip was used to allow external control of the stimulator control state machine's clock. The fully integrated bidirectional interface chip was not able to be tightly controlled in time due to dependence on an internal ring oscillator without locking. Configuration information for both the stimulator and recording chips is loaded with a serial interface, which is driven with LabJack U3 PC interfaces and custom Python scripts. These configuration streams could have been implemented through the FPGA, but lack of appropriate Altera Quartus licenses

made compiles prohibitively lengthy, preventing rapid configurability. The LabJack/Python interface could reconfigure the chips in less than 10 seconds, as opposed to a several-minute FPGA RAM recompile. The FPGA contains Verilog modules used for deserializing and decoding the recording system output, synchronously controlling the stimulator chip to create consistent artifacts, and implementing the artifact canceller. The canceller output is serialized and streamed back into the recording chip, where it is decoded and relayed to the capacitive DAC via a synthesized Verilog block.

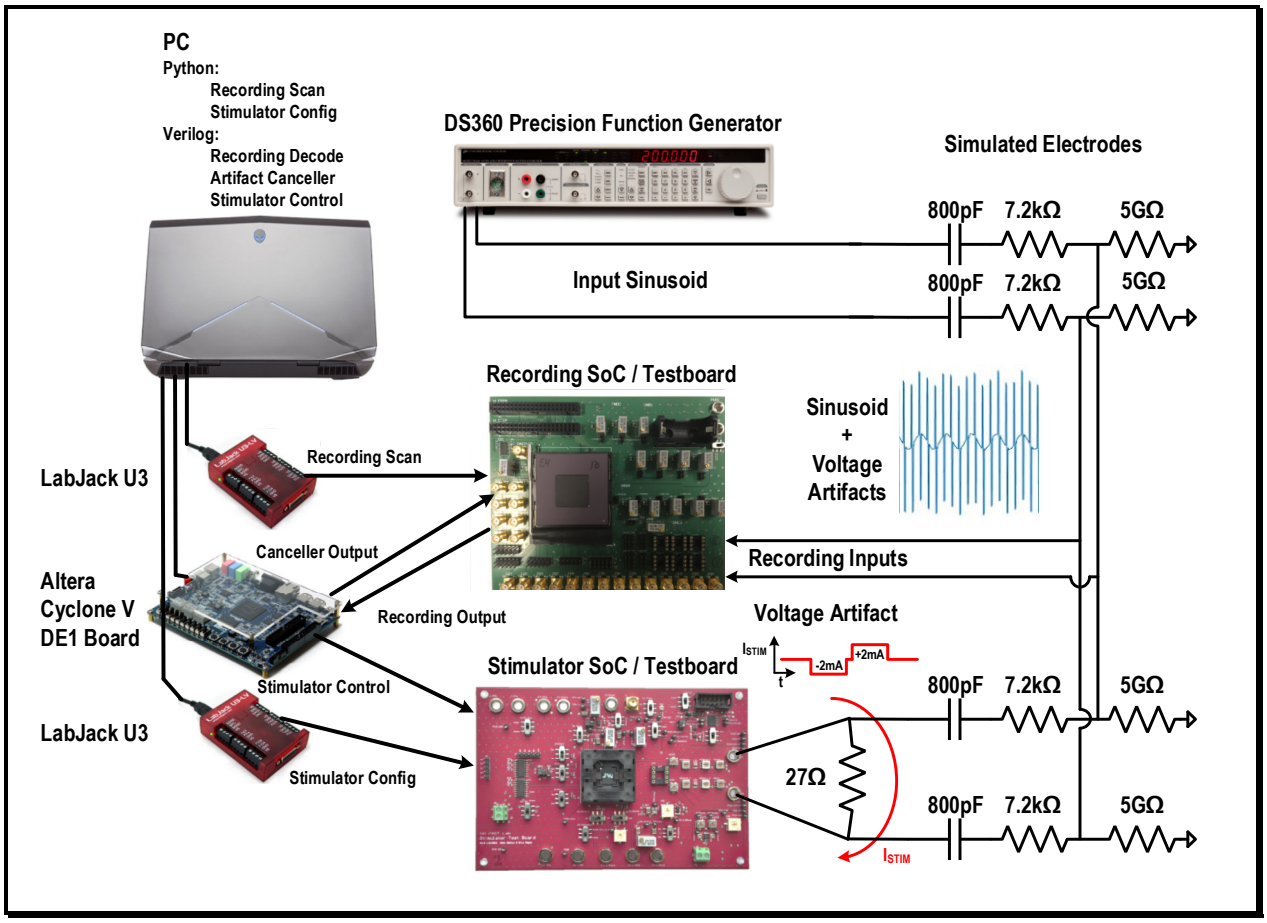

Figure 38: Combined Two-Chip/FPGA Testbench Setup Block Diagram

The stimulator is configured to drive a ±2mA balanced biphasic pulse through a 27Ω resistor. A high current/low resistance configuration was chosen to mitigate the large amount of

high-frequency noise generated by the stimulator PLL and control logic. This noise is typically low-pass filtered by the neural tissue and long connection leads. However, this setup directly injects the stimulation artifact into the recording inputs. The stimulator uses a combination of active and passive discharge after the pulse. This is not necessary with such a low load resistance, but it is important to include the effects of active discharge (zero or low source impedance at the stimulator outputs) on the recording inputs in the test results. The stimulator is synchronized by using a divided version of the recording system serializer enable as a clock. This enable signal goes high at the end of every channel's recording cycle. Dividing this clock by 64 gives a clock that increments once every 2kHz sample, making a consistent artifact when sampling at 2kHz.

In order to demonstrate maintained signal integrity while using the artifact canceller, the stimulation artifact was combined with an arbitrary sinusoidal input, resulting in the data given in the next chapter. This was done by shunting two simulated electrodes in parallel, allowing the stimulator and signal source to equally drive the differential recording inputs. The simulated electrodes are designed into the recording system test board and consist of series 800pF capacitors and 7.2kΩ resistors with a shunt 5GΩ resistor to ground at the recording inputs. These simulated electrodes appropriately bias the recording inputs to ground and roughly approximate the source impedance presented by the neural electrode-tissue interface. Realistically simulating this source impedance is crucial to predict the frequency transfer characteristics of the recording front-end in a true neural recording environment.

Figure 39 shows the real implementation of the aforementioned test setup. Interfacing two chip test boards with a single FPGA while maintaining signal integrity was challenging. However, the seemingly messy setup proved robust and consistent.

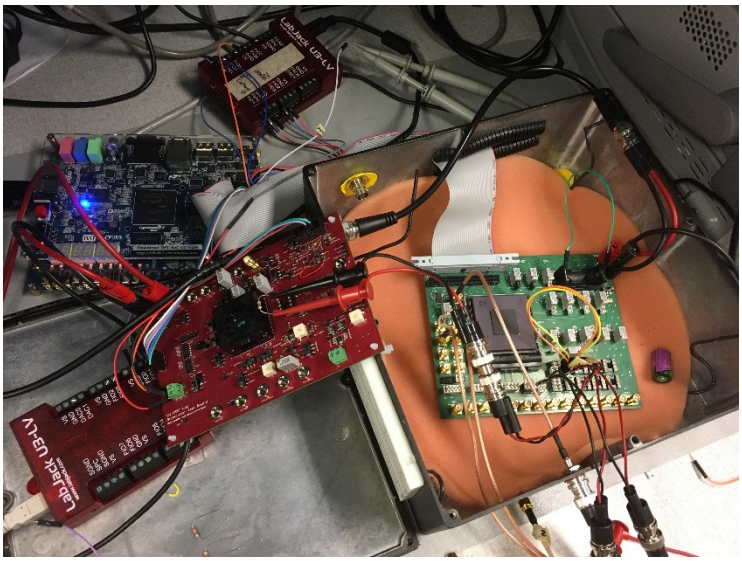

Figure 39: Physical Implementation of the Combined Two-Chip/FPGA Testbench Setup

5.2 Measurement Results

There are a few measurements that verify correct operation of the adaptive artifact canceller. The filter must converge when satisfying the LMS convergence criterion. Also, the artifact must be sufficiently suppressed to prevent obfuscation of the underlying signal. This section outlines the collection of these measurements and provides satisfactory proof of the canceller's operation. All measurements in this section are derived from transient data streams obtained through Quartus SignalTap II software, which reads data from internal FPGA registers to a PC in semi-real-time. Three signals were collected, the recording ADC output, the recording accumulator DAC code, and the artifact canceller DAC code.

5.2.1 *Convergence*

When an un-cancelled artifact hits the recording inputs, it saturates the amplifier and ADC chain. This slews the CDAC accumulator value, but does not fully capture the artifact amplitude. The artifact is too fast for the slow CDAC digital feedback to fully correct for the high voltages. Because of this saturation, looking at the ADC output only gives limited information on artifact

suppression and convergence. Figure 40 shows convergence of the adaptive canceller represented by only the ADC and accumulator output (summed to create a full-scale recording output). The artifact signal is averaged over 8 samples and a bitwise-shift division by 16 gives an equivalent μ of $1/128$. Stimulation pulses 2ms in width occur at 54Hz. As can be seen, saturation of the ADC prevents realization of the full convergence trend.

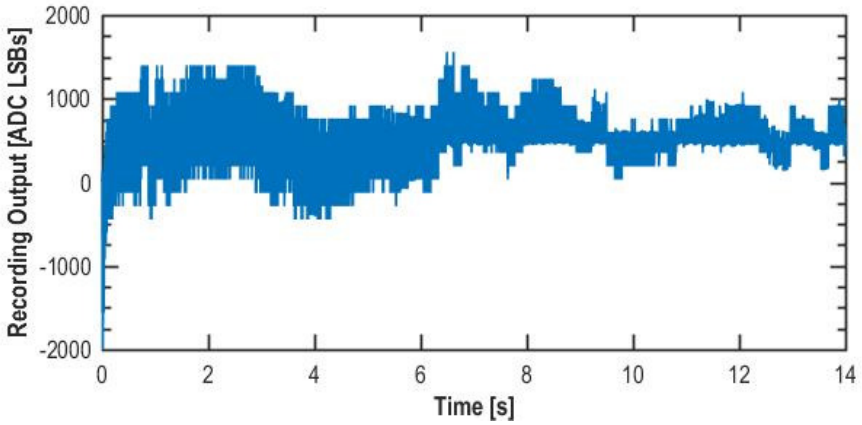

Figure 40: Artifact Cancellation Convergence with μ=1/128 and ±54mV Artifacts at 54Hz Via Recording Output

With a known artifact (after convergence the artifact is "recorded" by the artifact canceller FIR tap weights), the full scale artifact and its equivalent suppression can be superimposed on this convergence plot, showing the full extent of convergence and suppression, as shown in Figure 40.

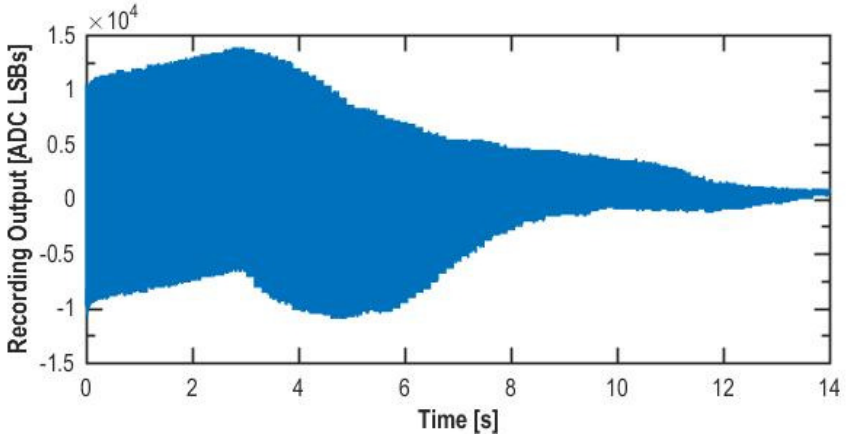

Figure 41: Artifact Cancellation Convergence with μ=1/64 and ±54mV Artifacts at 54Hz Via Full Signal

Figure 40 and 41 show convergence with minimal background noise. Figure 42 shows the same convergence trend in the presence of a 5mV, 17Hz sinusoid. The sinusoid is too large to be captured entirely by the ADC alone, and the DAC transitions to capture its full amplitude. Because the adaptive canceller only sees the ADC output, the resulting noise is chopped up in frequency to be more random than a 17Hz sine wave. The resulting "noise" covers approximately ±100ADC codes, which is sufficiently suppressed by the μ of 1/128.

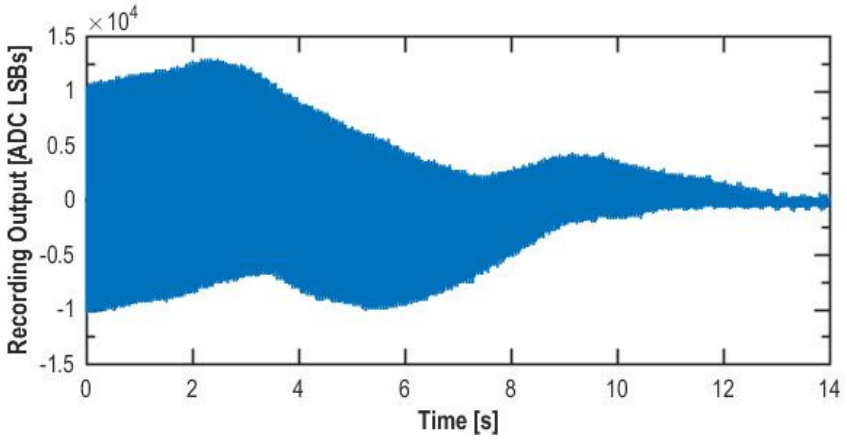

Figure 42: Artifact Cancellation Convergence with μ=1/64 and ±54mV Artifacts at 54Hz with 5mV, 17Hz Background Sinusoid Via Full Signal

5.2.2 *Voltage Suppression Magnitude*

This section explores several methods of measuring artifact suppression performance. The simplest method is raw analog voltage suppression capability. That is, the ability to suppress a given-magnitude voltage event to a lower voltage. Figure 43 shows a sample artifact before and after suppression. The full-scale artifact is interpolated using the DAC-to-ADC transfer function and the stored FIR tap weights after training. This particular artifact was cancelled with the same convergence parameters given in the previous convergence section, μ=1/64 with ±54mV Artifacts

at 54Hz and a 5mV, 17Hz background sinusoid. The canceller demonstrates 43.85dB of voltage suppression in Figure 43.

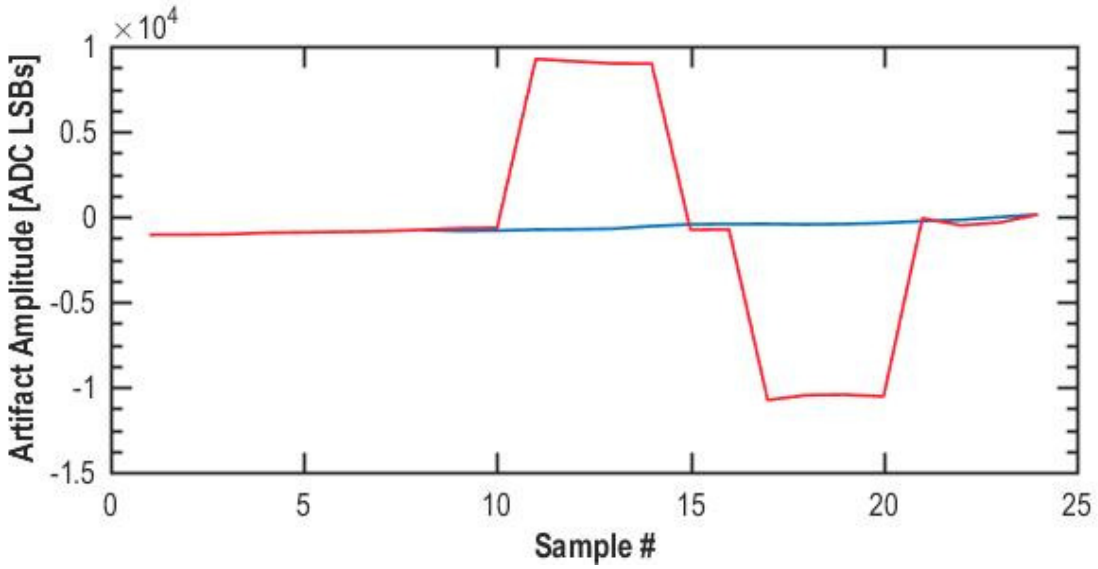

Figure 43: Artifact Suppression Demonstration; Before and After Showing 43.85dB of Voltage Suppression

5.2.3 *Signal-to-Noise Ratio Improvement*

A more comprehensive evaluation of canceller performance also includes information about preserving the underlying signal while suppressing artifacts. This is captured by measuring the signal-to-noise-and-distortion ratio before and after cancellation. This measurement encapsulates artifact power suppression as well as signal recovery by preventing recording chain saturation. The amount of signal power obscured by the artifact is highly dependent on the stimulation frequency, length of the stimulation pulse, and frequency content of the signal in question. The following measurements are performed with a large (5mV or 10mV) 17Hz tone as the signal. These tones are slow and large enough to exercise the digital CDAC feedback (an important function that must be maintained during artifact suppression). The canceller is adapted in each case from scratch (starting from zeroed filter taps for each plot). Each instance uses a averaging window of 16 samples and varying bitwise-shift division values. This varying mu is a

well-known process known as gear shifting. Gear shifting was done visually and by hand in this implementation, but future work will include automatic gear shifts.

5.2.3.1 5Hz Stimulation Frequency

The first round of tests were performed with infrequent stimulation pulses to increase visibility and demonstrate the effect of recording chain saturation. Figures 44, 45, and 46 show transient plots of a reconstructed version of the analog input, the recording output without cancellation, and the recording output with cancellation, respectively, in the presence of ±54mV stimulation artifacts at 5Hz.

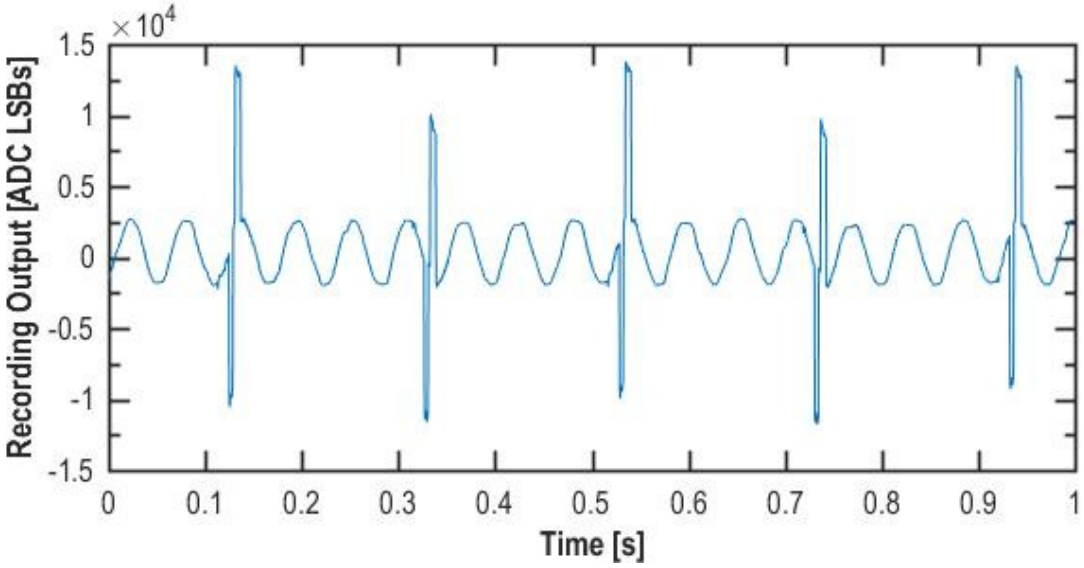

Figure 44: Reconstructed Analog Input, 10mV, 17Hz Sinusoid with ±54mV, 5Hz Stimulation

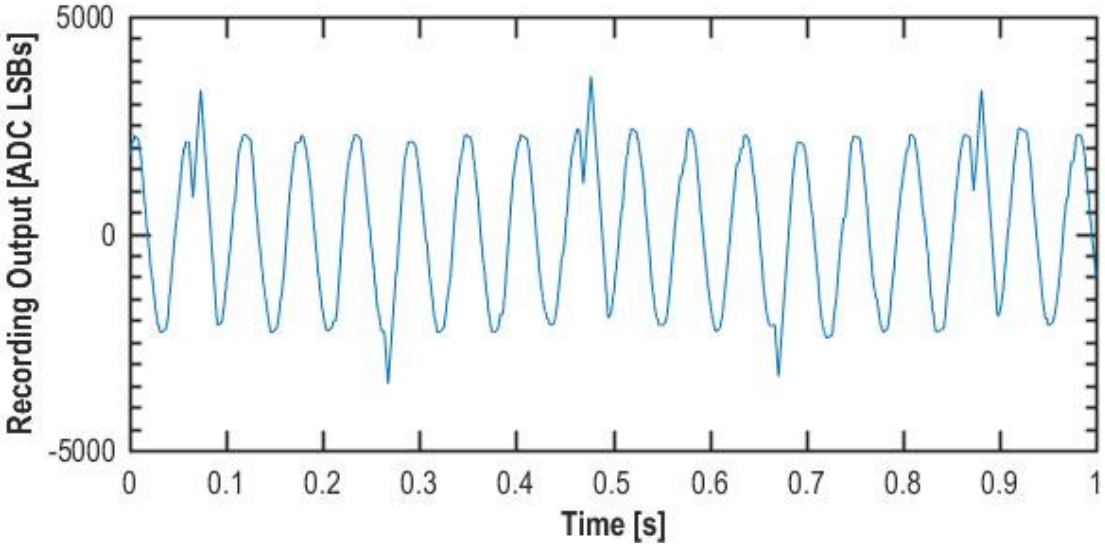

Figure 45: Recording Output Without Cancellation, 10mV, 17Hz Sinusoid with ±54mV, 5Hz Stimulation

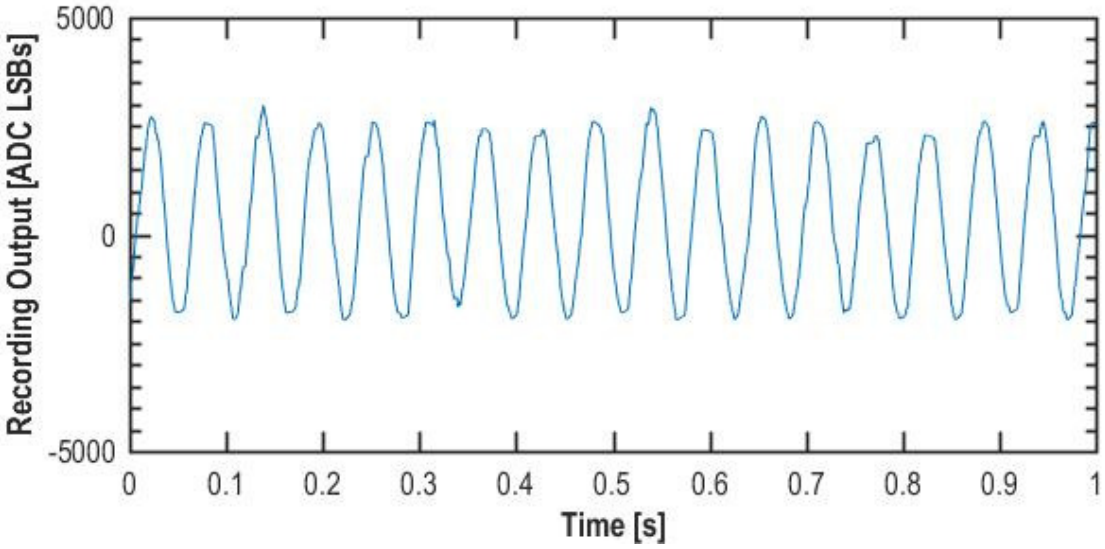

Figure 46: Recording Output With Cancellation, 10mV, 17Hz Sinusoid with ±54mV, 5Hz Stimulation

Figures 47, 48, and 49 show spectral density plots of 15 seconds of transient data, samples of which are shown in Figures 44-46 Again, the figures are ordered to show the reconstructed analog input first, the recording output without cancellation second, and the recording output with cancellation last.

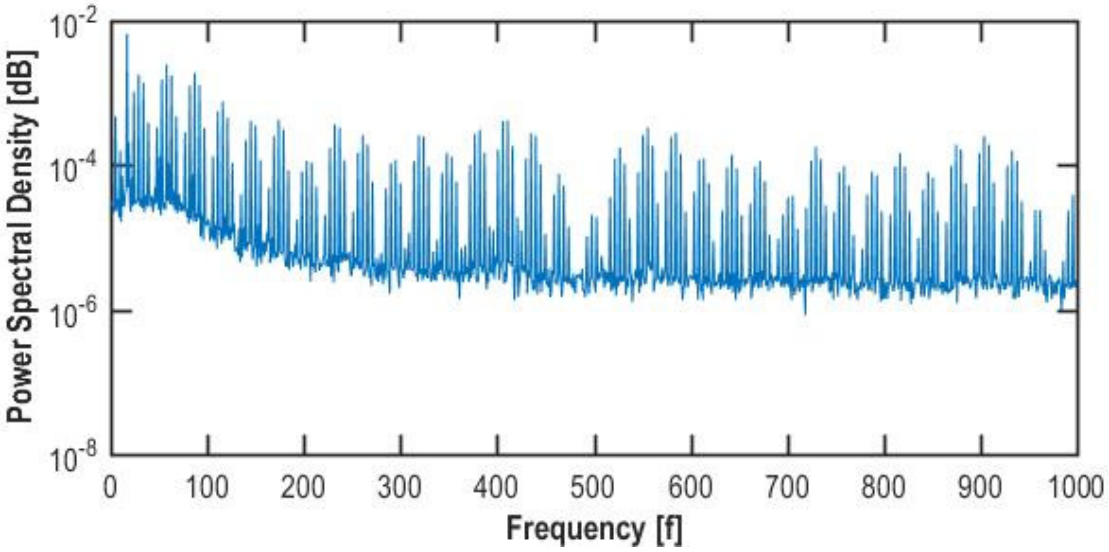

Figure 47: PSD of Reconstructed Analog Input, 10mV, 17Hz Sinusoid with ±54mV, 5Hz Stimulation (SINAD = -3.37)

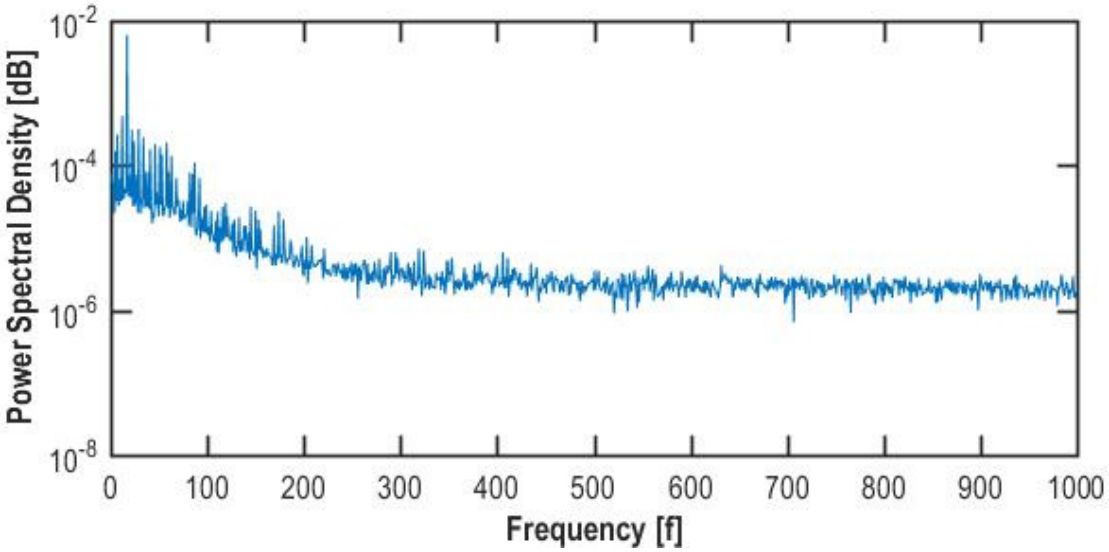

Figure 48: PSD of Recording Output Without Cancellation, 10mV, 17Hz Sinusoid with ±54mV, 5Hz Stimulation (SINAD = 8.84)

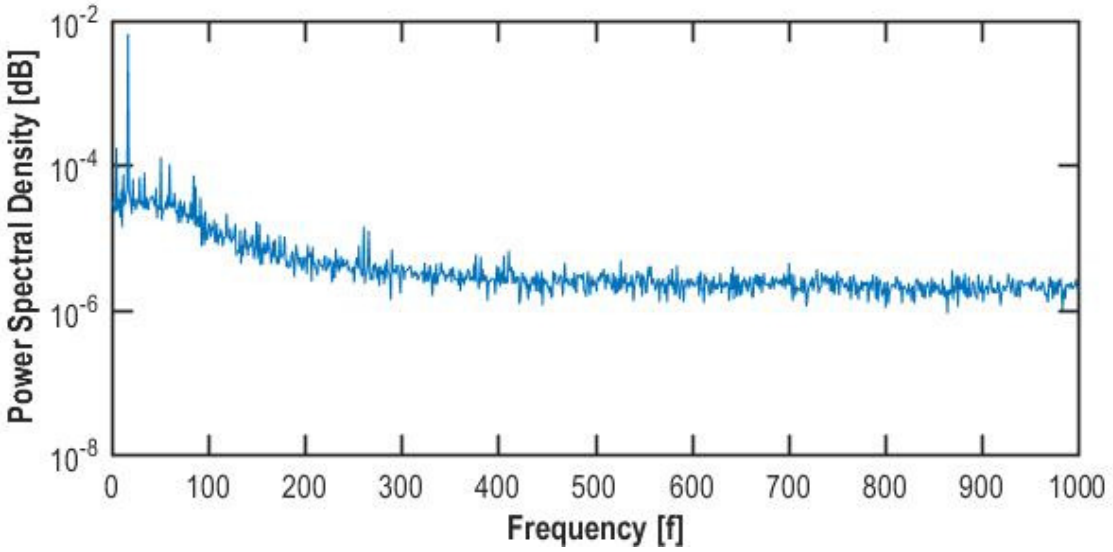

Figure 49: PSD of Recording Output With Cancellation, 10mV, 17Hz Sinusoid with ±54mV, 5Hz Stimulation (SINAD = 19.98)

The 5Hz stimulation frequency barely obscures the underlying sinusoid, so the signal degradation at the recording output is minimal. This can be seen by comparing the 17Hz tone amplitude between Figures 48 and 49. There is no appreciable attenuation. In this stimulator setup, the canceller suppresses artifact power by 23dB in the analog domain, and 11dB of this cancellation is captured by the saturated recording chain.

5.2.3.2 54Hz Stimulation Frequency

A more realistic stimulation frequency will obscure more signal information than the 5Hz period in the previous section. This is reflected as a considerable SINAD degradation in the saturated recording chain. Figures 50, 51, and 52 show the transient plots of a reconstructed version of the analog input, the recording output without cancellation, and the recording output with cancellation, respectively, in the presence of ±54mV stimulation artifacts at 80Hz.

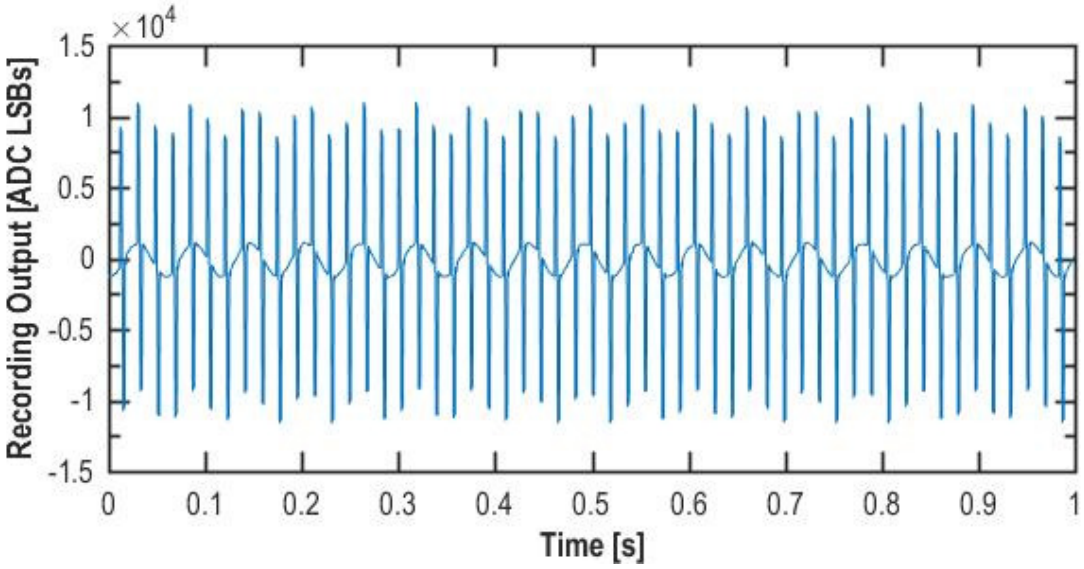

Figure 50: Reconstructed Analog Input, 10mV, 17Hz Sinusoid with ±54mV, 54Hz Stimulation

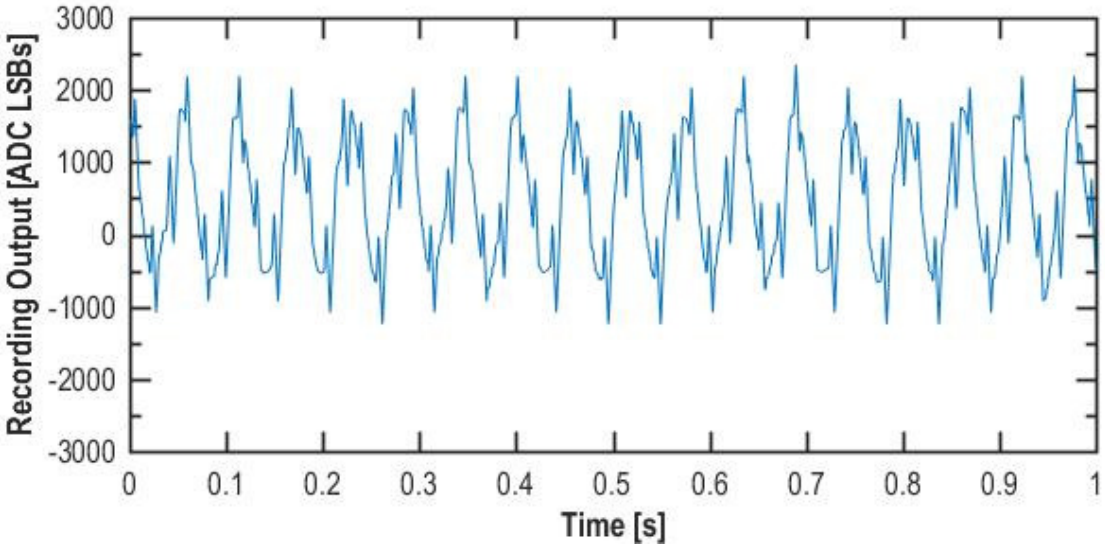

Figure 51: Recording Output Without Cancellation, 10mV, 17Hz Sinusoid with ±54mV, 54Hz Stimulation

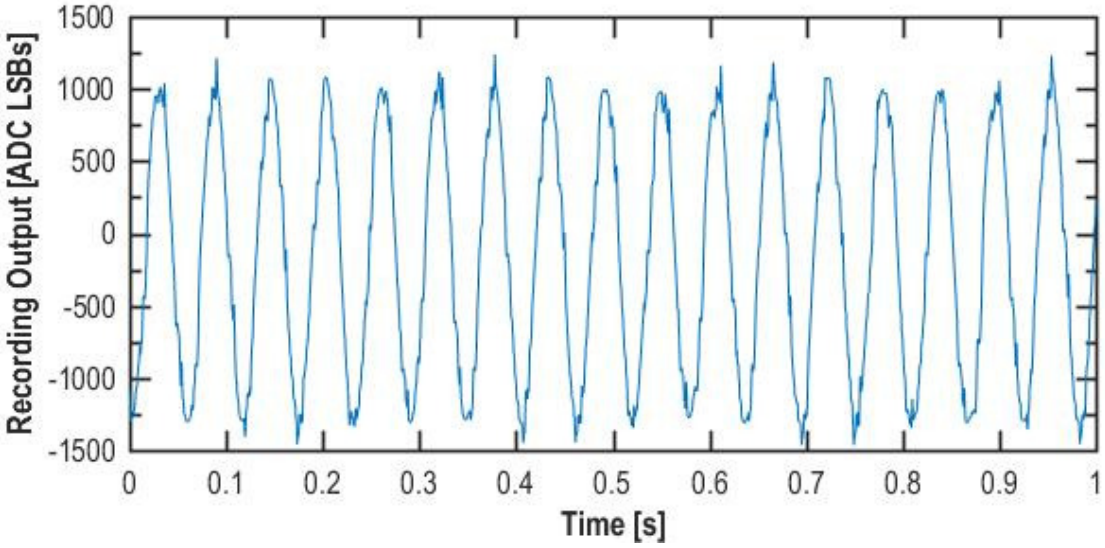

Figure 52: Recording Output With Cancellation, 10mV, 17Hz Sinusoid with ±54mV, 54Hz Stimulation

Figures 53, 54, and 55 show spectral density plots of 30 seconds of transient data, samples of which are shown in Figures 50-52. Again, the figures are ordered to show the reconstructed analog input first, the recording output without cancellation second, and the recording output with cancellation last.

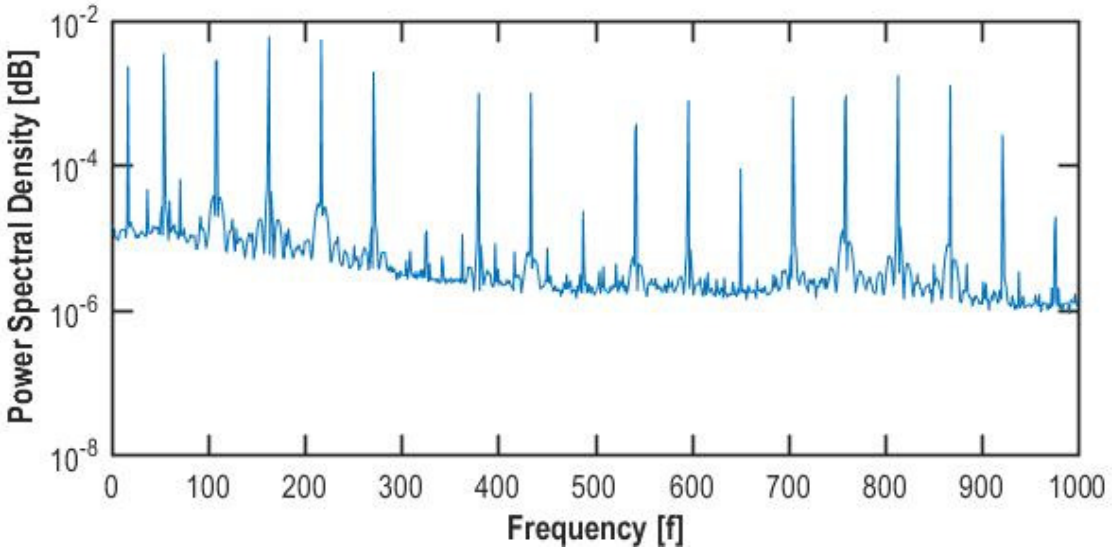

Figure 53: PSD of Reconstructed Analog Input, 10mV, 17Hz Sinusoid with ±54mV, 54Hz Stimulation (SINAD = -13.3)

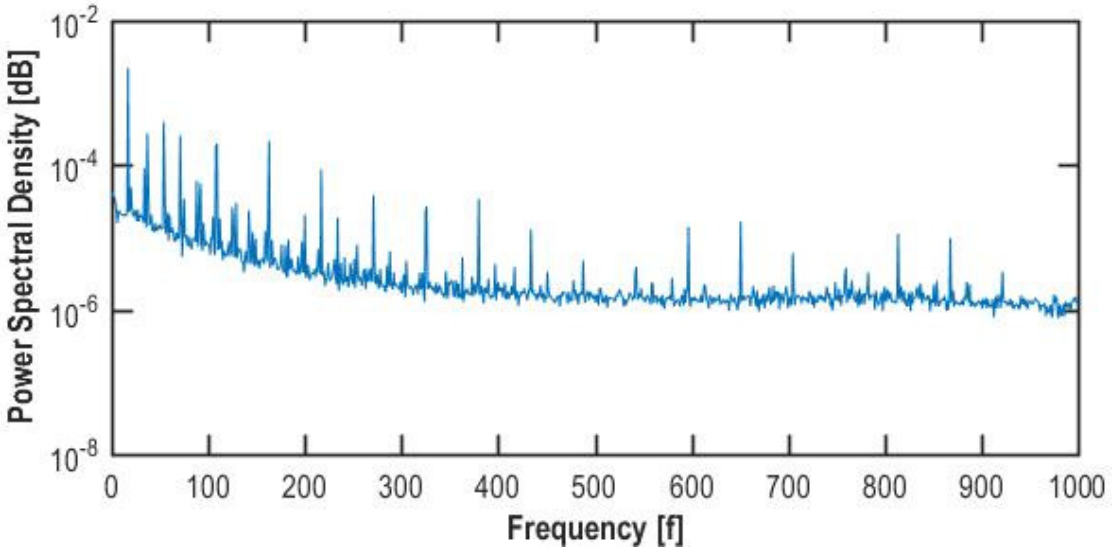

Figure 54: PSD of Recording Output Without Cancellation, 10mV, 17Hz Sinusoid with ±54mV, 54Hz Stimulation (SINAD = 7.31)

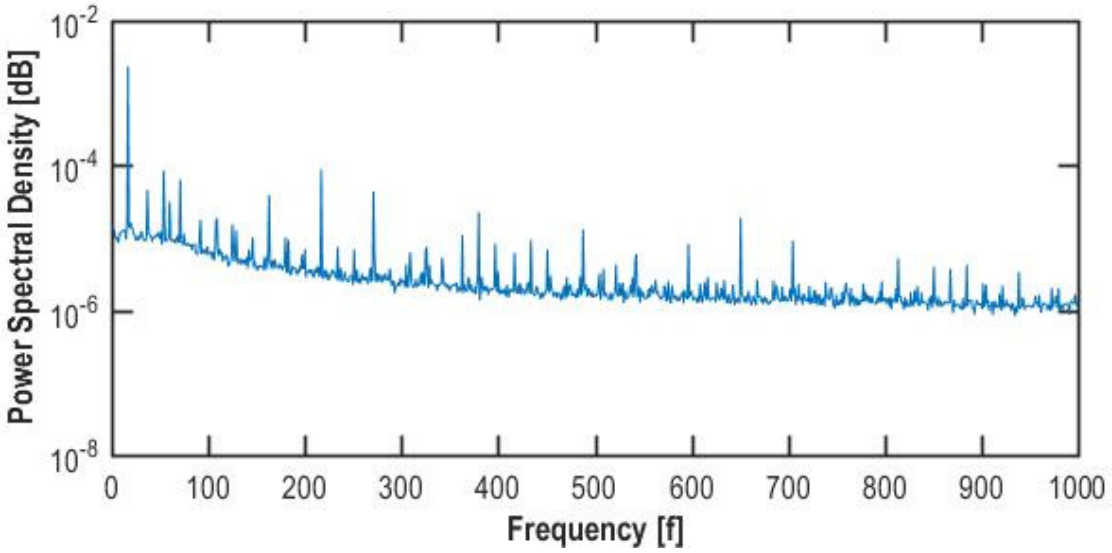

Figure 55: PSD of Recording Output With Cancellation, 10mV, 17Hz Sinusoid with ±54mV, 5Hz Stimulation (SINAD = 19.42)

As compared to the low-frequency stimulation case, the SINAD at the saturated recording output is degraded by 2dB, while the cancelled output is only degraded by 0.4dB by increasing the stimulation frequency. These plots show the efficacy of the artifact canceller in various stimulation environments, suppressing unwanted stimulation artifacts while preserving the underlying signal.

Chapter 6. CONCLUSION

6.1 SUMMARY

A scalable, technology-portable method for adaptive integrated neural stimulus artifact cancellation has been presented and demonstrated with integrated circuit interfaces and real-time measurements. The cancellation topology uses a combination of averaging and division by truncation to offer noise immunity, which is demonstrated in combination with a mixed-mode feedback recording architecture that by nature creates instability-causing transient noise. The digital adaptation algorithm is shown to be scalable to any number of simultaneous recording channels and arbitrary voltage artifact magnitudes and lengths.

The system has been implemented on a combined stimulate and record SoC in 65nm bulk CMOS and on a mixed-chip platform with a FPGA processing backbone. The FPGA prototype presented is highly reconfigurable, and its robustness will soon be demonstrated with in-vivo testing on a non-human primate subject. After full system verification with an in-vivo test, the adaptive artifact canceller will be re-integrated with improved versions of the stimulator and recording architectures. This final work will represent a significant breakthrough in channel count density and integration of recording and stimulator analog front-ends with unique enabling technologies, such as artifact cancellation, and a robust digital back-end.

The algorithm developed in this work is most likely in its final form. The theoretical derivation and practical considerations will be further investigated in preparation for publication to inform the research community as to the challenges of integrated artifact cancellation. Specific work includes side-by-side comparison of various methods of division in implementation of the step size parameter. The merits and shortcomings of the methods of averaging, truncation, and

floating-point division will be explored in detail. There will be specific focus on the considerations and optimizations made necessary by low-power integrated circuit design.

This work represents a significant foray into implementing robust bidirectional neural interfaces. The presented adaptive artifact canceller has been designed holistically considering the bidirectional interface. The design is synergistic with stimulator and recording systems and is closely tailored for efficiency and ease of use with an integrated circuit platform.

6.2 DIRECTIONS

The field of neural interfaces and artifact cancellation is moving towards improvements in scalability and efficiency. The neural recording front-end has been fully optimized in the analog domain. To achieve the power efficiency necessary to implement thousands of recording channels on a single chip, more and more functions must be integrated in digital CMOS circuits and shared between multiple channels. The neural signal is recorded at low data rates compared to the achievable speeds with modern bulk CMOS. Digital hardware for electrocorticography channels operating at two kilo-samples per second could be multiplexed to thousands of channels by performing computations in the megahertz, relatively slow compared to the speed limitations of fine-line digital CMOS. Similarly, artifact cancellation will continue to be modularized and scaled for use with many channels.

The amount of possible cancellation at the front-end is set by the dynamic range of the digital-to-analog interface. This is independent of the back-end operation, given that the artifact canceller has converged to an optimum cancellation solution. The path for improvement lies in optimizing the area and power consumption of the canceller back-end per recording channel. The artifact mitigation methods identified in Chapter 2 are all implemented on a channel-by-channel

basis. The next step is to multiplex a single integrated artifact canceller back-end between many channels, leveraging high speed CMOS to reduce power and area costs per recording channel.

Apart from optimizing existing systems, future work on the neural interface will mostly cover full system integration. The desired implantable SoC contains stimulation and recording front-ends for many channels, artifact cancellation, on-chip data processing for recording-dependent stimulation, wireless telemetry capabilities, and on-chip power conversion. This trend includes integration of a considerable digital back-end, possibly including a full microprocessor. The size and complexity of the full digital back-end and its corresponding memories may dwarf the canceller area and power requirements. Once this happens, the canceller update logic and memory will most likely be absorbed into the on-chip microprocessor. There will also most likely be digital post-processing on the recorded signal to further suppress the apparent artifact magnitude, enabled by on-chip DSP capabilities.

BIBLIOGRAPHY

[1] C. T. Moritz, S. I. Perlmutter, and E. E. Fetz, "Direct control of paralysed muscles by cortical neurons," *Nature*, vol. 456, no. 7222, pp. 639–642, Dec. 2008.

[2] R. B. Stein and V. Mushahwar, "Reanimating limbs after injury or disease," *Trends Neurosci.*, vol. 28, no. 10, pp. 518–524, Oct. 2005.

[3] M. F. Bear, B. W. Connors, and M. A. Paradiso, *Neuroscience: Exploring the Brain, 3rd Edition*, 3rd edition. Philadelphia, PA: Lippincott Williams and Wilkins, 2006.

[4] W. Franks, I. Schenker, P. Schmutz, and A. Hierlemann, "Impedance characterization and modeling of electrodes for biomedical applications," *IEEE Trans. Biomed. Eng.*, vol. 52, no. 7, pp. 1295–1302, Jul. 2005.

[5] W. M. Grill and J. T. Mortimer, "Stimulus waveforms for selective neural stimulation," *IEEE Eng. Med. Biol. Mag.*, vol. 14, no. 4, pp. 375–385, Jul. 1995.

[6] T. J. Foutz and C. C. McIntyre, "Evaluation of novel stimulus waveforms for deep brain stimulation," *J. Neural Eng.*, vol. 7, no. 6, p. 66008, Dec. 2010.

[7] M. Sahin and Y. Tie, "Non-rectangular waveforms for neural stimulation with practical electrodes," *J. Neural Eng.*, vol. 4, no. 3, pp. 227–233, Sep. 2007.

[8] A. Wongsarnpigoon and W. M. Grill, "Energy-efficient waveform shapes for neural stimulation revealed with genetic algorithm," *J. Neural Eng.*, vol. 7, no. 4, p. 46009, Aug. 2010.

[9] E. J. Peterson, D. A. Dinsmoor, D. J. Tyler, and T. J. Denison, "Stimulation artifact rejection in closed-loop, distributed neural interfaces," in *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, 2016, pp. 233–236.

[10] "The Development and Application of Optogenetics," *Annu. Rev. Neurosci.*, vol. 34, no. 1, pp. 389–412, 2011.

[11] T. P. Ladas, C.-C. Chiang, L. E. Gonzalez-Reyes, T. Nowak, and D. M. Durand, "Seizure reduction through interneuron-mediated entrainment using low frequency optical stimulation," *Exp. Neurol.*, vol. 269, pp. 120–132, Jul. 2015.

[12] W. Biederman *et al.*, "A 4.78 mm 2 Fully-Integrated Neuromodulation SoC Combining 64 Acquisition Channels With Digital Compression and Simultaneous Dual Stimulation," *IEEE J. Solid-State Circuits*, vol. 50, no. 4, pp. 1038–1047, Apr. 2015.

[13] H. Chandrakumar and D. Marković, “5.5 A 2 #x00B5;W 40mVpp linear-input-range chopper- stabilized bio-signal amplifier with boosted input impedance of 300M #x03A9; and electrode-offset filtering,” in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, 2016, pp. 96–97.

[14] W. Jiang, V. Hokhikyan, H. Chandrakumar, V. Karkare, and D. Markovic, “28.6 A ??50mV linear-input-range VCO-based neural-recording front-end with digital nonlinearity correction,” in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, 2016, pp. 484–485.

[15] D. A. Wagenaar and S. M. Potter, “Real-time multi-channel stimulus artifact suppression by local curve fitting,” *J. Neurosci. Methods*, vol. 120, no. 2, pp. 113–120, Oct. 2002.

[16] T. Al-ani, F. Cazettes, S. Palfi, and J.-P. Lefaucheur, “Automatic removal of high-amplitude stimulus artefact from neuronal signal recorded in the subthalamic nucleus,” *J. Neurosci. Methods*, vol. 198, no. 1, pp. 135–146, May 2011.

[17] J. W. Gnadt, S. D. Echols, A. Yildirim, H. Zhang, and K. Paul, “Spectral cancellation of microstimulation artifact for simultaneous neural recording in situ,” *IEEE Trans. Biomed. Eng.*, vol. 50, no. 10, pp. 1129–1135, Oct. 2003.

[18] A. E. Mendrela, J. Cho, J. A. Fredenburg, C. A. Chestek, M. P. Flynn, and E. Yoon, “Enabling closed-loop neural interface: A bi-directional interface circuit with stimulation artifact cancellation and cross-channel CM noise suppression,” in *2015 Symposium on VLSI Circuits (VLSI Circuits)*, 2015, pp. C108–C109.

[19] J. Lee, H. G. Rhew, D. R. Kipke, and M. P. Flynn, “A 64 Channel Programmable Closed-Loop Neurostimulator With 8 Channel Neural Amplifier and Logarithmic ADC,” *IEEE J. Solid-State Circuits*, vol. 45, no. 9, pp. 1935–1945, Sep. 2010.

[20] B. K. Thurgood, D. J. Warren, N. M. Ledbetter, G. A. Clark, and R. R. Harrison, “A Wireless Integrated Circuit for 100-Channel Charge-Balanced Neural Stimulation,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 3, no. 6, pp. 405–414, Dec. 2009.

[21] E. Pepin, D. Micheletti, S. Perlmutter, and J. C. Rudell, “High-voltage compliant, capacitive-load invariant neural stimulation electronics compatible with standard bulk-CMOS integration,” in *2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings*, 2014, pp. 260–263.

[22] E. Pepin, J. Uehlin, D. Micheletti, S. I. Perlmutter, and J. C. Rudell, “A high-voltage compliant, electrode-invariant neural stimulator front-end in 65nm bulk-CMOS,” in *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, 2016, pp. 229–232.

[23] R. W. Lucky, “Automatic equalization for digital communication,” *Bell Syst. Tech. J.*, vol. 44, no. 4, pp. 547–588, Apr. 1965.

[24] S. U. H. Qureshi, “Adaptive equalization,” *Proc. IEEE*, vol. 73, no. 9, pp. 1349–1387, Sep. 1985.

[25] A. Gray and J. Markel, “A spectral-flatness measure for studying the autocorrelation method of linear prediction of speech analysis,” *IEEE Trans. Acoust. Speech Signal Process.*, vol. 22, no. 3, pp. 207–217, Jun. 1974.

[26] P. Monsen, “Feedback equalization for fading dispersive channels,” *IEEE Trans. Inf. Theory*, vol. 17, no. 1, pp. 56–64, Jan. 1971.

[27] M. E. (Martin E. Austin, “Decision-feedback equalization for digital communication over dispersive channels.,” 1967.

[28] R. Muller *et al.*, “A Minimally Invasive 64-Channel Wireless #x03BC;ECoG Implant,” *IEEE J. Solid-State Circuits*, vol. 50, no. 1, pp. 344–359, Jan. 2015.

APPENDIX 1.A: SOC VERILOG SOURCE CODE

```
module lms_fir(clk, valid, reset, update, capture, tap_num, weight_in, pos_amp, neg_amp, pos_stim,
neg_stim, error_in, mu_shift, out_shift, ch_read, ch_write, scanmode, scan_channel, cm_mode, cm_enable,
reg_clk, reg_enable, data_out, reg_read_channel, reg_write_channel, cm_enable_out, reg_weight_in,
reg_weight_out, scan_weight_out, cm_delay);
//By John Uehlin
//ECOG3 Tapeout 2016, Artifact Top-Level Module

        input clk; //System clock from recording
        input valid; //Valid signal indicating valid ADC sample
        input reset; //Global system reset (non-synchronous)
        input update; //Update signal indicating valid scan in data
        input capture; //Capture signal that latches scan out data
        input [4:0] tap_num; //Number of enabled taps --SCAN
        input [223:0] weight_in; //Arbitrary filter weights from scan input --SCAN
        input [7:0] pos_amp; //Positive stimulation DAC code --SCAN
        input [7:0] neg_amp; //Negative stimulation DAC code --SCAN
        input pos_stim; //Positive stimulation timing pulse --PAD
        input neg_stim; //Negative stimulation timing pulse --PAD
        input signed [7:0] error_in; //Error signal from ADC
        input [4:0] mu_shift; //Training mu parameter (bitwise division only) --SCAN
        input signed [7:0] out_shift; //Output truncation shift (2's compliment, signed) --SCAN
        input [5:0] ch_read; //Current FIR channel, from recording
         input [5:0] ch_write; //Current weight update channel, from recording
        input scanmode; //Enables scan in of weights --SCAN
        input [5:0] scan_channel; //Indicates channel for weight scan in --SCAN
        input cm_mode; //Enables selective common-mode enabling (1: CM enable only during STIM, 0: always
enabled) --SCAN
        input cm_enable; //Enables common-mode canceller
        input [223:0] reg_weight_out; //Weights from register file
        input [6:0] cm_delay; //Length of CM enable after stim pulse --SCAN

        output reg_clk; //Clock to register
        output reg_enable; //Write enable to register file
        output signed [7:0] data_out; //FIR data out to CDAC
        output [5:0] reg_read_channel; //Read channel to register file
        output [5:0] reg_write_channel; //Write channel to register file
        output cm_enable_out; //CM canceller enable signal to recording
        output [223:0] reg_weight_in; //Weight update output to register file
        output [223:0] scan_weight_out; //Weight update output to scan chain

        wire signed [8:0] data_in;
        wire signed [15:0] data_out_int;
        wire [7:0] data_out_dummy;
        wire [31:0] tap_dec; //Thermometer encoded individual tap enable
        wire [15:0] dataout0, dataout1, dataout2, dataout3, dataout4, dataout5, dataout6, dataout7, dataout8,
dataout9; //FIR tap outputs
        wire [15:0] dataout10, dataout11, dataout12, dataout13, dataout14, dataout15, dataout16, dataout17,
dataout18, dataout19;
        wire [15:0] dataout20, dataout21, dataout22, dataout23, dataout24, dataout25, dataout26, dataout27,
dataout28, dataout29, dataout30, dataout31;
        wire [223:0] mac_weight_out; //Weight update output
        wire cm_enable_delay; //CM enable during stim
        wire [7:0] error_pos; //Positive error for mu shift
        wire signed [7:0] data_out_presat;

        //Input delay line
        reg signed [8:0] data_in_del2, data_in_del3, data_in_del4, data_in_del5, data_in_del6, data_in_del7,
data_in_del8, data_in_del9, data_in_del10, data_in_del11;
```

```
	reg signed [8:0] data_in_del12, data_in_del13, data_in_del14, data_in_del15, data_in_del16, data_in_del17,
data_in_del18, data_in_del19, data_in_del20, data_in_del21;
	reg signed [8:0] data_in_del22, data_in_del23, data_in_del24, data_in_del25, data_in_del26, data_in_del27,
data_in_del28, data_in_del29, data_in_del30, data_in_del31, data_in_del32, data_in_del33;
	reg signed [7:0] error_stored; //Latched ADC input
	reg pos_stim_hold, neg_stim_hold;

	//Enable scan in to register file
	assign reg_clk = (scanmode) ? update : ~valid;
	assign reg_enable = (scanmode) ? capture : ~valid;

	//Stim input decode/mu division
	assign data_in = (pos_stim_hold) ? {1'b0,pos_amp} : ((neg_stim_hold) ? (~{1'b0,neg_amp}+1'b1) : 9'b0);

	//Weight scan config
	assign reg_read_channel = (scanmode) ? scan_channel : ch_read;
	 assign reg_write_channel = (scanmode) ? scan_channel : ch_write;

	//CM enable while artifact is present
	assign cm_enable_out = (cm_enable) ? ((cm_mode) ? cm_enable_delay : 1'b1) : 1'b0;

	//Scan in/out weight functionality
	assign reg_weight_in = (scanmode) ? weight_in : mac_weight_out;

	//Pass reg weight out to left side for weight scan out
	assign scan_weight_out = reg_weight_out;

	//Convert error to magnitude
	assign error_pos = (error_in[7]) ? (~(error_in)+1'b1) : error_in;

	//Saturation protection on output arithmetic shift
	assign data_out = (data_out_int[15]) ? ((data_out_presat[7]) ? data_out_presat : 8'sb10000001) :
((data_out_presat[7]) ? 8'sb01111111 : data_out_presat);

	//Store error for computation/mu division
	always @(posedge valid) begin
		if (reset) begin
			error_stored <= 8'b0;
		end else begin
			error_stored <= (error_in[7]) ? (~(error_pos>>mu_shift[4:2])+1'b1) :
(error_pos>>mu_shift[4:2]);
		end
	end

	//Delay chain for input data
	always @(negedge clk) begin
		if(reset) begin
			pos_stim_hold <= 0;
			neg_stim_hold <= 0;
			data_in_del2 <= 9'b0;
			data_in_del3 <= 9'b0;
			data_in_del4 <= 9'b0;
			data_in_del5 <= 9'b0;
			data_in_del6 <= 9'b0;
			data_in_del7 <= 9'b0;
			data_in_del8 <= 9'b0;
			data_in_del9 <= 9'b0;
			data_in_del10 <= 9'b0;
			data_in_del11 <= 9'b0;
			data_in_del12 <= 9'b0;
			data_in_del13 <= 9'b0;
			data_in_del14 <= 9'b0;
```

```
                data_in_del15 <= 9'b0;
                data_in_del16 <= 9'b0;
                data_in_del17 <= 9'b0;
                data_in_del18 <= 9'b0;
                data_in_del19 <= 9'b0;
                data_in_del20 <= 9'b0;
                data_in_del21 <= 9'b0;
                data_in_del22 <= 9'b0;
                data_in_del23 <= 9'b0;
                data_in_del24 <= 9'b0;
                data_in_del25 <= 9'b0;
                data_in_del26 <= 9'b0;
                data_in_del27 <= 9'b0;
                data_in_del28 <= 9'b0;
                data_in_del29 <= 9'b0;
                data_in_del30 <= 9'b0;
                data_in_del31 <= 9'b0;
                data_in_del32 <= 9'b0;
                data_in_del33 <= 9'b0;
        end else begin
                pos_stim_hold <= pos_stim;
                neg_stim_hold <= neg_stim;
                data_in_del2 <= data_in;
                data_in_del3 <= data_in_del2;
                data_in_del4 <= data_in_del3;
                data_in_del5 <= data_in_del4;
                data_in_del6 <= data_in_del5;
                data_in_del7 <= data_in_del6;
                data_in_del8 <= data_in_del7;
                data_in_del9 <= data_in_del8;
                data_in_del10 <= data_in_del9;
                data_in_del11 <= data_in_del10;
                data_in_del12 <= data_in_del11;
                data_in_del13 <= data_in_del12;
                data_in_del14 <= data_in_del13;
                data_in_del15 <= data_in_del14;
                data_in_del16 <= data_in_del15;
                data_in_del17 <= data_in_del16;
                data_in_del18 <= data_in_del17;
                data_in_del19 <= data_in_del18;
                data_in_del20 <= data_in_del19;
                data_in_del21 <= data_in_del20;
                data_in_del22 <= data_in_del21;
                data_in_del23 <= data_in_del22;
                data_in_del24 <= data_in_del23;
                data_in_del25 <= data_in_del24;
                data_in_del26 <= data_in_del25;
                data_in_del27 <= data_in_del26;
                data_in_del28 <= data_in_del27;
                data_in_del29 <= data_in_del28;
                data_in_del30 <= data_in_del29;
                data_in_del31 <= data_in_del30;
                data_in_del32 <= data_in_del31;
                data_in_del33 <= data_in_del32;
        end
end

//CM enable counter
cm_counter C0 (
        .run_flag(cm_enable_delay),
        .clk(valid),
        .reset(reset),
```

```
            .start_flag(pos_stim_hold||neg_stim_hold),
            .stop(cm_delay)
);

//Weight update modules
weight_mac W0 (
            .clk(valid),
            .reset(reset),
            .data_in((data_in_del2[8]) ? data_in_del2[8:2]+1'b1 : (data_in_del2[8:2])),
            .error_in(error_stored),
            .weight_in(reg_weight_out[6:0]),
            .mu_shift(mu_shift[1:0]),
            .weight_out(mac_weight_out[6:0])
);

weight_mac W1 (
            .clk(valid),
            .reset(reset),
            .data_in((data_in_del3[8]) ? data_in_del3[8:2]+1'b1 : (data_in_del3[8:2])),
            .error_in(error_stored),
            .weight_in(reg_weight_out[13:7]),
            .mu_shift(mu_shift[1:0]),
            .weight_out(mac_weight_out[13:7])
);

weight_mac W2 (
            .clk(valid),
            .reset(reset),
            .data_in((data_in_del4[8]) ? data_in_del4[8:2]+1'b1 : (data_in_del4[8:2])),
            .error_in(error_stored),
            .weight_in(reg_weight_out[20:14]),
            .mu_shift(mu_shift[1:0]),
            .weight_out(mac_weight_out[20:14])
);

weight_mac W3 (
            .clk(valid),
            .reset(reset),
            .data_in((data_in_del5[8]) ? data_in_del5[8:2]+1'b1 : (data_in_del5[8:2])),
            .error_in(error_stored),
            .weight_in(reg_weight_out[27:21]),
            .mu_shift(mu_shift[1:0]),
            .weight_out(mac_weight_out[27:21])
);

weight_mac W4 (
            .clk(valid),
            .reset(reset),
            .data_in((data_in_del6[8]) ? data_in_del6[8:2]+1'b1 : (data_in_del6[8:2])),
            .error_in(error_stored),
            .weight_in(reg_weight_out[34:28]),
            .mu_shift(mu_shift[1:0]),
            .weight_out(mac_weight_out[34:28])
);

weight_mac W5 (
            .clk(valid),
            .reset(reset),
            .data_in((data_in_del7[8]) ? data_in_del7[8:2]+1'b1 : (data_in_del7[8:2])),
            .error_in(error_stored),
            .weight_in(reg_weight_out[41:35]),
            .mu_shift(mu_shift[1:0]),
```

```
        .weight_out(mac_weight_out[41:35])
);

weight_mac W6 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del8[8]) ? data_in_del8[8:2]+1'b1 : (data_in_del8[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[48:42]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[48:42])
);

weight_mac W7 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del9[8]) ? data_in_del9[8:2]+1'b1 : (data_in_del9[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[55:49]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[55:49])
);

weight_mac W8 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del10[8]) ? data_in_del10[8:2]+1'b1 : (data_in_del10[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[62:56]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[62:56])
);

weight_mac W9 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del11[8]) ? data_in_del11[8:2]+1'b1 : (data_in_del11[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[69:63]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[69:63])
);

weight_mac W10 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del12[8]) ? data_in_del12[8:2]+1'b1 : (data_in_del12[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[76:70]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[76:70])
);

weight_mac W11 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del13[8]) ? data_in_del13[8:2]+1'b1 : (data_in_del13[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[83:77]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[83:77])
);
```

```
weight_mac W12 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del14[8]) ? data_in_del14[8:2]+1'b1 : (data_in_del14[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[90:84]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[90:84])
);

weight_mac W13 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del15[8]) ? data_in_del15[8:2]+1'b1 : (data_in_del15[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[97:91]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[97:91])
);

weight_mac W14 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del16[8]) ? data_in_del16[8:2]+1'b1 : (data_in_del16[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[104:98]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[104:98])
);

weight_mac W15 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del17[8]) ? data_in_del17[8:2]+1'b1 : (data_in_del17[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[111:105]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[111:105])
);

weight_mac W16 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del18[8]) ? data_in_del18[8:2]+1'b1 : (data_in_del18[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[118:112]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[118:112])
);

weight_mac W17 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del19[8]) ? data_in_del19[8:2]+1'b1 : (data_in_del19[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[125:119]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[125:119])
);

weight_mac W18 (
```

```
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del20[8]) ? data_in_del20[8:2]+1'b1 : (data_in_del20[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[132:126]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[132:126])
);

weight_mac W19 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del21[8]) ? data_in_del21[8:2]+1'b1 : (data_in_del21[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[139:133]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[139:133])
);

weight_mac W20 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del22[8]) ? data_in_del22[8:2]+1'b1 : (data_in_del22[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[146:140]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[146:140])
);

weight_mac W21 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del23[8]) ? data_in_del23[8:2]+1'b1 : (data_in_del23[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[153:147]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[153:147])
);

weight_mac W22 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del24[8]) ? data_in_del24[8:2]+1'b1 : (data_in_del24[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[160:154]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[160:154])
);

weight_mac W23 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del25[8]) ? data_in_del25[8:2]+1'b1 : (data_in_del25[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[167:161]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[167:161])
);

weight_mac W24 (
        .clk(valid),
        .reset(reset),
```

```
        .data_in((data_in_del26[8]) ? data_in_del26[8:2]+1'b1 : (data_in_del26[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[174:168]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[174:168])
);

weight_mac W25 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del27[8]) ? data_in_del27[8:2]+1'b1 : (data_in_del27[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[181:175]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[181:175])
);

weight_mac W26 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del28[8]) ? data_in_del28[8:2]+1'b1 : (data_in_del28[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[188:182]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[188:182])
);

weight_mac W27 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del29[8]) ? data_in_del29[8:2]+1'b1 : (data_in_del29[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[195:189]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[195:189])
);

weight_mac W28 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del30[8]) ? data_in_del30[8:2]+1'b1 : (data_in_del30[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[202:196]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[202:196])
);

weight_mac W29 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del31[8]) ? data_in_del31[8:2]+1'b1 : (data_in_del31[8:2])),
        .error_in(error_stored),
        .weight_in(reg_weight_out[209:203]),
        .mu_shift(mu_shift[1:0]),
        .weight_out(mac_weight_out[209:203])
);

weight_mac W30 (
        .clk(valid),
        .reset(reset),
        .data_in((data_in_del32[8]) ? data_in_del32[8:2]+1'b1 : (data_in_del32[8:2])),
        .error_in(error_stored),
```

```
                .weight_in(reg_weight_out[216:210]),
                .mu_shift(mu_shift[1:0]),
                .weight_out(mac_weight_out[216:210])
        );

        weight_mac W31 (
                .clk(valid),
                .reset(reset),
                .data_in((data_in_del33[8]) ? data_in_del33[8:2]+1'b1 : (data_in_del33[8:2])),
                .error_in(error_stored),
                .weight_in(reg_weight_out[223:217]),
                .mu_shift(mu_shift[1:0]),
                .weight_out(mac_weight_out[223:217])
        );

        //Tap enable decoder
        DW_thermdec_inst DEC0 (
                .inst_en(1'b1),
                .inst_a(tap_num),
                .inst_b(tap_dec)
        );

        //Filter instants
           fir_tap TAP0 (
                        .data_in(data_in),
                        .weight_in(reg_weight_out[6:0]&{7{tap_dec[0]}}),
                        .data_out(dataout0));

           fir_tap TAP1 (
                        .data_in(data_in_del2),
                        .weight_in(reg_weight_out[13:7]&{7{tap_dec[1]}}),
                        .data_out(dataout1));

fir_tap TAP2 (
           .data_in(data_in_del3),
           .weight_in(reg_weight_out[20:14]&{7{tap_dec[2]}}),
           .data_out(dataout2));

fir_tap TAP3 (
           .data_in(data_in_del4),
           .weight_in(reg_weight_out[27:21]&{7{tap_dec[3]}}),
           .data_out(dataout3));

fir_tap TAP4 (
           .data_in(data_in_del5),
           .weight_in(reg_weight_out[34:28]&{7{tap_dec[4]}}),
           .data_out(dataout4));

fir_tap TAP5 (
           .data_in(data_in_del6),
           .weight_in(reg_weight_out[41:35]&{7{tap_dec[5]}}),
           .data_out(dataout5));

fir_tap TAP6 (
           .data_in(data_in_del7),
           .weight_in(reg_weight_out[48:42]&{7{tap_dec[6]}}),
           .data_out(dataout6));

fir_tap TAP7 (
           .data_in(data_in_del8),
           .weight_in(reg_weight_out[55:49]&{7{tap_dec[7]}}),
           .data_out(dataout7));
```

```
fir_tap TAP8 (
        .data_in(data_in_del9),
        .weight_in(reg_weight_out[62:56]&{7{tap_dec[8]}}),
        .data_out(dataout8));

fir_tap TAP9 (
        .data_in(data_in_del10),
        .weight_in(reg_weight_out[69:63]&{7{tap_dec[9]}}),
        .data_out(dataout9));

fir_tap TAP10 (
        .data_in(data_in_del11),
        .weight_in(reg_weight_out[76:70]&{7{tap_dec[10]}}),
        .data_out(dataout10));

fir_tap TAP11 (
        .data_in(data_in_del12),
        .weight_in(reg_weight_out[83:77]&{7{tap_dec[11]}}),
        .data_out(dataout11));

fir_tap TAP12 (
        .data_in(data_in_del13),
        .weight_in(reg_weight_out[90:84]&{7{tap_dec[12]}}),
        .data_out(dataout12));

fir_tap TAP13 (
        .data_in(data_in_del14),
        .weight_in(reg_weight_out[97:91]&{7{tap_dec[13]}}),
        .data_out(dataout13));

fir_tap TAP14 (
        .data_in(data_in_del15),
        .weight_in(reg_weight_out[104:98]&{7{tap_dec[14]}}),
        .data_out(dataout14));

fir_tap TAP15 (
        .data_in(data_in_del16),
        .weight_in(reg_weight_out[111:105]&{7{tap_dec[15]}}),
        .data_out(dataout15));

fir_tap TAP16 (
        .data_in(data_in_del17),
        .weight_in(reg_weight_out[118:112]&{7{tap_dec[16]}}),
        .data_out(dataout16));

fir_tap TAP17 (
        .data_in(data_in_del18),
        .weight_in(reg_weight_out[125:119]&{7{tap_dec[17]}}),
        .data_out(dataout17));

fir_tap TAP18 (
        .data_in(data_in_del19),
        .weight_in(reg_weight_out[132:126]&{7{tap_dec[18]}}),
        .data_out(dataout18));

fir_tap TAP19 (
        .data_in(data_in_del20),
        .weight_in(reg_weight_out[139:133]&{7{tap_dec[19]}}),
        .data_out(dataout19));

fir_tap TAP20 (
```

```
            .data_in(data_in_del21),
            .weight_in(reg_weight_out[146:140]&{7{tap_dec[20]}}),
            .data_out(dataout20));

fir_tap TAP21 (
            .data_in(data_in_del22),
            .weight_in(reg_weight_out[153:147]&{7{tap_dec[21]}}),
            .data_out(dataout21));

fir_tap TAP22 (
            .data_in(data_in_del23),
            .weight_in(reg_weight_out[160:154]&{7{tap_dec[22]}}),
            .data_out(dataout22));

fir_tap TAP23 (
            .data_in(data_in_del24),
            .weight_in(reg_weight_out[167:161]&{7{tap_dec[23]}}),
            .data_out(dataout23));

fir_tap TAP24 (
            .data_in(data_in_del25),
            .weight_in(reg_weight_out[174:168]&{7{tap_dec[24]}}),
            .data_out(dataout24));

fir_tap TAP25 (
            .data_in(data_in_del26),
            .weight_in(reg_weight_out[181:175]&{7{tap_dec[25]}}),
            .data_out(dataout25));

fir_tap TAP26 (
            .data_in(data_in_del27),
            .weight_in(reg_weight_out[188:182]&{7{tap_dec[26]}}),
            .data_out(dataout26));

fir_tap TAP27 (
            .data_in(data_in_del28),
            .weight_in(reg_weight_out[195:189]&{7{tap_dec[27]}}),
            .data_out(dataout27));

fir_tap TAP28 (
            .data_in(data_in_del29),
            .weight_in(reg_weight_out[202:196]&{7{tap_dec[28]}}),
            .data_out(dataout28));

fir_tap TAP29 (
            .data_in(data_in_del30),
            .weight_in(reg_weight_out[209:203]&{7{tap_dec[29]}}),
            .data_out(dataout29));

fir_tap TAP30 (
            .data_in(data_in_del31),
            .weight_in(reg_weight_out[216:210]&{7{tap_dec[30]}}),
            .data_out(dataout30));

fir_tap TAP31 (
            .data_in(data_in_del32),
            .weight_in(reg_weight_out[223:217]&{7{tap_dec[31]}}),
            .data_out(dataout31));

        //Output summing
        DW02_sum_inst SUM0 (
```

```
                .inst_INPUT({dataout0, dataout1, dataout2, dataout3, dataout4, dataout5, dataout6, dataout7,
dataout8, dataout9, dataout10, dataout11, dataout12, dataout13, dataout14, dataout15, dataout16, dataout17,
dataout18, dataout19, dataout20, dataout21, dataout22, dataout23, dataout24, dataout25, dataout26, dataout27,
dataout28, dataout29, dataout30, dataout31}),
                .inst_SUM(data_out_int)
        );

        //Output bitwise shift/truncation
        DW01_ash_inst ASH0 (
                .inst_A(data_out_int),
                .inst_DATA_TC(1'b1),
                .inst_SH({out_shift}),
                .inst_SH_TC(1'b1),
                .inst_B({data_out_presat,data_out_dummy})
        );

endmodule
```

module weight_mac(clk, reset, data_in, error_in, weight_in, mu_shift, weight_out);
//By John Uehlin
//ECOG3 Tapeout 2016, Artifact Canceller Weight Update Module

```
        input clk; //Latches in weight and data
        input reset; //Resets registers
        input signed [6:0] data_in; //Update X
        input signed [7:0] error_in; //Update E
        input signed [6:0] weight_in; //Update W(n-1)
        input [1:0] mu_shift; //Second-tier mu shift
        output reg signed [6:0] weight_out; //Updated weight

        reg signed [6:0] data_stored;
        reg signed [6:0] weight_stored;
        reg signed [6:0] weight_stored_2;
        reg signed [6:0] mult_prod;
        wire signed [7:0] overflow;
        reg overflow_flag_mult;
        reg sign_flag_mult;
        wire signed [14:0] mult_pretrunc;

    //mult_pretrunc = data_stored*error_in
    DW02_mult_inst_weight MWEIGHT (
        .inst_A(data_stored),
        .inst_B(error_in),
        .inst_TC(1'b1),
        .inst_PRODUCT(mult_pretrunc)
    );

    //Saturation and overflow protection
    assign overflow = mult_prod + weight_stored_2;
    always @* begin
        case ({data_stored[6],error_in[7]})
            2'b00: sign_flag_mult <= 0;
            2'b11: sign_flag_mult <= 0;
            2'b10: sign_flag_mult <= 1;
            2'b01: sign_flag_mult <= 1;
        endcase
        case (mult_pretrunc[14:6])
            9'b111111111: overflow_flag_mult <= 0;
            9'b000000000: overflow_flag_mult <= 0;
            default: overflow_flag_mult <= 1;
        endcase
```

```
      case ({overflow_flag_mult,sign_flag_mult})
         2'b00: mult_prod <= mult_pretrunc[6:0];
         2'b01: mult_prod <= mult_pretrunc[6:0];
         2'b10: mult_prod <= 7'b0111111;
         2'b11: mult_prod <= 7'b1000001;
      endcase
      case ({mult_prod[6],weight_stored[6]})
         2'b00: weight_out <= (overflow[7]) ? 7'b0111111 : overflow[6:0];
         2'b01: weight_out <= overflow[6:0];
         2'b10: weight_out <= overflow[6:0];
         2'b11: weight_out <= (overflow[7] != overflow[6]) ? 7'b100001 : overflow[6:0];
      endcase
   end

   //Latch weight input
   always @(negedge clk) begin
                     if (reset) begin
         weight_stored <= 7'b0;
                     end else begin
         weight_stored <= weight_in;
      end
   end

   //Delay weight input and latch X input
   always @(posedge clk) begin
          if (reset) begin
                     data_stored <= 7'b0;
         weight_stored_2 <= 7'b0;
                     end else begin
         weight_stored_2 <= weight_stored;
                  if (data_in[6]) begin
                           data_stored <= ~((~data_in+1'b1)>>>mu_shift)+1'b1;
                   end else begin
                           data_stored <= data_in>>>mu_shift;
                  end
          end
    end
endmodule

module fir_tap(data_in, weight_in, data_out);
//By John Uehlin
//ECOG3 Tapeout 2016, Artifact Canceller FIR Tap Module
          input signed [8:0] data_in;
          input signed [6:0] weight_in;
          output signed [15:0] data_out;

          //data_out = data_in*weight_in
   DW02_mult_inst_tap MTAP (
      .inst_A(data_in),
      .inst_B(weight_in),
      .inst_TC(1'b1),
      .inst_PRODUCT(data_out)
   );
Endmodule
```

APPENDIX 1.B: FPGA VERILOG SOURCE CODE

```
module lms_fir_final(serial_en, serial_clk, ch, ch_trip, reset, adapt, stim0, error_in, mu, data_out,
cm_enable_out);
//By John Uehlin
//ECOG4 FPGA-based Artifact Canceller 2017
//Artifact Top-Level Module
    input serial_en; //Input from recording chip serializer (High while serializer is transmitting data)
    input serial_clk; //Input clock from recording chip serializer (Typically 13.56MHz)
    input [5:0] ch; //Recording channel for current deserialized ADC sample
    input [5:0] ch_trip; //Channel at which the FIR and update clock increments
    input reset; //Global system reset (non-synchronous)
    input adapt; //LMS Update does not receive input when low, turns LMS adaptation on and off
    reg [7:0] tap_num = 8'd20; //Number of enabled taps
    reg [7:0] amp0 = 8'd1; //First amplitude stimulation DAC code
    reg [7:0] amp1 = 8'd1; //Second amplitude stimulation DAC code
    input stim0; //First stimulation timing pulse
    reg stim1 = 1'b0; //Second stimulation timing pulse --Not used in this implementation
    input signed [7:0] error_in; //Error signal from ADC
    reg [7:0] mu_mult = 8'd1; //Training mu parameter (multiplication factor) --Not used in this implementation
    input [5:0] mu; //Training mu parameter (bitwise division only): mu is actually 1>>>mu
    reg cm_mode = 1'b1; //1: CM enable only happens in presence of stim artifact, 0: CM always enabled (on
cm_enable)
    reg cm_enable = 1'b0; //Enables cm_canceller
    reg [6:0] cm_delay = 7'd31; //Length of CM enable after stimulation onset on cm_mode = 1
    output reg signed [7:0] data_out; //FIR data out to serializer, to recording CDAC
    output cm_enable_out; //CM canceller enable out to serializer, to recording front-end

    //Register input/output wires
    wire [7:0] reg_weight_out;
    wire [7:0] reg_weight_out_delay;
    wire reg_clk;
    wire reg_enable;
    wire [7:0] reg_weight_in;
    wire [11:0] reg_avg_in;
    wire [11:0] reg_avg_out;
    wire [7:0] mac_weight_out;

    //Latched error signals
    reg signed [7:0] error_stored;
    reg signed [8:0] error_stored_presat;

    //CM canceller wires
    wire cm_enable_delay;

    //FIR tap counter registers
    reg [7:0] input_counter;
    reg [7:0] input_counter_f;
    reg [7:0] input_counter_next;
    reg [7:0] input_counter_f_next;

    //FIR tap and update input registers
    reg [7:0] input_amp;
    reg input_flag;
    reg [7:0] input_amp_next;
    reg input_flag_next;
    reg clk, valid;

    //Subtracted value for mean zeroing
    reg signed [7:0] mean_zero;
```

```
//Translates serializer signals from chip into usable clocks (divides by # of channels)
lms_timing UTIMING(
    .serial_en(serial_en),
    .serial_clk(serial_clk),
    .ch(ch[5:0]),
    .ch_trip(ch_trip[5:0]),
    .lms_clk(clk),
    .lms_valid(valid)
);

//Clock inversion for register file
assign reg_clk = ~clk;
//Enable register write only during stimulation pulse and during adapation
assign reg_enable = (input_counter >= 1) && adapt;

//CM enable while artifact is present
assign cm_enable_out = (cm_enable) ? ((cm_mode) ? cm_enable_delay : 1'b1) : 1'b0;

//Name change for weight update/register interface
assign reg_weight_in = mac_weight_out;

//Latches mean-zeroing value at onset of stimulation
always @(posedge input_flag or posedge reset) begin
    if (reset) begin
        mean_zero <= 8'd0;
    end else begin
        mean_zero <= error_in;
    end
end

//Store error to be passed to weight update/averaging module
always @(posedge clk) begin
    if (reset) begin
        error_stored_presat <= 9'b0;
    end else begin
        error_stored_presat <= {error_in[7],error_in} - {mean_zero[7],mean_zero};
    end
end

//Overflow protection in the mean-zeroing subtraction
always @* begin
    if (error_stored_presat >= 9'sd127) begin
        error_stored <= 8'sd127;
    end else if (error_stored_presat <= -9'sd127) begin
        error_stored <= -8'sd127;
    end else begin
        error_stored <= error_stored_presat[7:0];
    end
end

//Latches for FIR tap counter state machine
always @(posedge clk) begin
    if(reset) begin
        input_counter <= 8'd0;
        input_counter_f <= 8'd0;
        input_amp <= 8'd0;
    end else begin
        input_counter <= input_counter_next;
        input_counter_f <= input_counter_f_next;
        input_amp <= input_amp_next;
    end
```

```
    end

    //Latches for stim duration flag state machine
    always @(negedge clk) begin
        if(reset) begin
            input_flag <= 0;
        end else begin
            input_flag <= input_flag_next;
        end
    end

    always @* begin //Amplitude select and decode (if both timing pulses come, 0 by default)
        case ({stim1,stim0,input_flag,input_counter})
            {3'b000,8'd0}: input_amp_next <= 8'd0;
            {3'b011,8'd0}: input_amp_next <= amp0;
            {3'b101,8'd0}: input_amp_next <= amp1;
            {3'b111,8'd0}: input_amp_next <= amp0;
            default: input_amp_next <= input_amp;
        endcase
        //FIR counter module  operation
        if (input_counter >= tap_num) begin
            input_counter_next <= 8'd0;
        end else if (input_flag) begin
            input_counter_next <= input_counter + 8'd1;
        end else begin
            input_counter_next <= 8'd0;
        end

        if (input_counter_f >= tap_num) begin
            input_counter_f_next <= 8'd0;
        //Delay to compensate for serializer/deserializer latency
        end else if ((input_flag && (input_counter_next != 8'd1) && (input_counter_next != 8'd2)) || (input_counter_f >
8'd0)) begin
            input_counter_f_next <= input_counter_f + 8'd1;
        end else begin
            input_counter_f_next <= 8'd0;
        end

        //Double-trigger prevention
        if (input_counter[7:0] != 8'd0) begin
            input_flag_next = 1;
        end else if (stim0||stim1) begin
            input_flag_next = 1;
        end else begin
            input_flag_next = 0;
        end
    end

    //FIR tap weight and averaging register file
    register128_8b REG0(
        .RESET(reset),
        .CLK(reg_clk),
        .INDEX_R0(input_counter),
        .INDEX_R1(input_counter_f),
        .INDEX_W(input_counter_f),
        .WRITE_ENABLE(reg_enable),
        .D(reg_weight_in),
        .D_AVG(reg_avg_in),
        .OUT0(reg_weight_out),
        .OUT1(reg_weight_out_delay),
        .OUT_AVG(reg_avg_out)
        );
```

```
    //CM enable counter
    cm_counter C0 (
        .run_flag(cm_enable_delay),
        .clk(valid),
        .reset(reset),
        .start_flag(input_flag),
        .stop(cm_delay)
    );

    //Weight update and averaging module
    weight_update W0 (
        .clk(valid),
        .reset(reset),
        .tap_num(tap_num),
        .tap_current(input_counter_f),
        .data_in(adapt?{1'b0,input_amp}:9'd0),
        .error_in(error_stored),
        .error_avg_in(reg_avg_out),
        .weight_in(reg_weight_out_delay[7:0]),
        .mu_mult(mu_mult),
        .mu(mu),
        .error_avg_out(reg_avg_in),
        .weight_out(mac_weight_out[7:0])
    );

    //Filter multiplier instant
    fir_tap TAP0 (
        .data_in(input_amp),
        .weight_in(reg_weight_out[7:0]),
        .data_out(data_out[7:0])
    );
endmodule

module lms_timing(
//By John Uehlin
//ECOG4 FPGA-based Artifact Canceller 2017
//Timing Signal Conditioning Module
    input serial_en, //Input from recording chip serializer (High while serializer is transmitting data)
    input serial_clk, //Input clock from recording chip serializer (Typically 13.56MHz)
    input [5:0] ch, //Recording channel for current deserialized ADC sample
    input [5:0] ch_trip, //Channel at which the FIR and update clock increments
    output reg lms_clk, //Inversion of lms_valid
    output reg lms_valid //Rising edge indicates recording deserializer data is valid
    );
    always @(negedge serial_clk) begin
        if (serial_en && (ch == ch_trip)) begin
            lms_valid <= 1'b1;
        end else begin
            lms_valid <= 1'b0;
        end
    end
    assign lms_clk = ~lms_valid;
endmodule

module register128_8b(
//By John Uehlin
//ECOG4 FPGA-based Artifact Canceller 2017
//Weight and Error Average Storage Register File
    input RESET, //Universal reset
    input CLK, //Clock in new write channel on data valid signal
    input [7:0] INDEX_R0, //Read weight index 0 -- FIR Read
```

```
    input [7:0] INDEX_R1, //Read weight index 1 -- Weight Update and Averaging
    input [7:0] INDEX_W, //Current write weight index -- Weight Update and Averaging
    input WRITE_ENABLE, //Enables write on posedge CLK
    input [7:0] D, //Weight Data
    input [11:0] D_AVG, //Averaging Data
    output reg [7:0] OUT0, //Out to FIR
    output reg [7:0] OUT1, //Out to Weight Update
    output reg [11:0] OUT_AVG //Out to Averaging
    );

    reg [7:0]   MEMORY[126:0];      //127 entries * 8 bit register file
    reg [11:0]    MEMORY_AVG[126:0];    //127 entries * 12 bit register file
    integer i;

    //Read
    //No use of input index 0 to prevent DC offset from canceller (idle state is 0)
    always @(*) begin
        if (INDEX_R0 != 8'd0) begin
            OUT0 = MEMORY[INDEX_R0-1];
        end
        if (INDEX_R1 != 8'd0) begin
            OUT1 = MEMORY[INDEX_R1-1];
            OUT_AVG = MEMORY_AVG[INDEX_R1-1];
        end
    end

    //Write
    always @(posedge CLK or posedge RESET) begin
        //Reset
        if (RESET) begin
            for (i = 0; i<127; i = i+1) begin
                MEMORY[i] <= 8'd0;
                MEMORY_AVG[i] <= 12'd0;
            end
        //Write on posedge CLK if WRITE_ENABLE
        //No use of input index 0 to prevent DC offset from canceller (idle state is 0)
        end else if(WRITE_ENABLE && (INDEX_W != 8'd0)) begin
            MEMORY[INDEX_W-1] <= D;
            MEMORY_AVG[INDEX_W-1] <= D_AVG;
         end
    end
endmodule

module cm_counter(run_flag, clk, reset, start_flag, stop);
//By John Uehlin
//ECOG4 FPGA-based Artifact Canceller 2017
//Common-Mode Canceller Enable Counter

//Takes a stop counter value. Counts clock posedges after input of start_flag.
//Run flag outputs high while counter is less than stop

//------------Output Ports--------------
    output reg run_flag;
//------------Input Ports--------------
    input clk, reset, start_flag;
    input [6:0] stop;
    reg [6:0] count;

    always @(posedge clk or posedge reset ) begin
        if (reset) begin
            count <= 7'b0;
            run_flag <= 1'b0;
```

```
        end else begin
            if(start_flag) begin
                run_flag <= 1'b1;
            end
            if(count >= stop) begin
                count <= 7'b0;
                run_flag <= 1'b0;
            end else if (run_flag == 1) begin
                count <= count + 1;
            end else begin
                count <= 7'b0;
            end
        end
    end
endmodule

module weight_update(clk, reset, tap_num, tap_current, data_in, error_in, error_avg_in, weight_in, mu_mult,
mu, error_avg_out, weight_out);
//By John Uehlin
//ECOG4 FPGA-based Artifact Canceller 2017
//LMS Weight Update and Error Averaging Module
    input clk; // Latches in ADC data, increments averaging counter
    input reset; // Resets internal registers
    input [7:0] tap_num; // Number of enabled taps
    input [7:0] tap_current; // Current tap used for weight update
    input signed [8:0] data_in; // Stimulator timing signal
    input signed [7:0] error_in; // ADC output
    input signed [11:0] error_avg_in; // Averaging accumulator input from register file
    input signed [7:0] weight_in; // Previous weight value from register file
    input [7:0] mu_mult; // Multiplicative mu factor
    input [5:0] mu; // Bitwise division mu divisor; mu_actual = mu_mult>>>mu
    output reg signed [11:0] error_avg_out; // Averaging sum output signal to register file
    output reg signed [7:0] weight_out; // Update operation weight out to register file

    reg [5:0] avg_counter; // Current number of averaged sampled
    reg [5:0] avg_counter_next; // Averaging counter state register
    reg signed [7:0] error_update; // Final, averaged error value
    reg signed [11:0] error_avg_next; // Averaging accumulator state register
    wire signed [11:0] error_avg_final; // Wire for bitwise shift and division in averaging

    // Final accumulator value
    assign error_avg_final = error_avg_in[11:0] + {{4{error_in[7]}},error_in[7:0]};

    // Increment the averaging counter and accumulator state machines
    always @(negedge clk or posedge reset) begin
        if (reset) begin
            avg_counter <= 6'd0;
            error_avg_out <= 12'd0;
        end else begin
            avg_counter <= avg_counter_next;
            error_avg_out <= error_avg_next;
        end
    end

    always @(*) begin
        // Average counter increments at the end of each stimulator pulse
        if ((tap_current >= tap_num)) begin
            if (avg_counter >= 7) begin
                avg_counter_next <= 12'd0;
            end else begin
                avg_counter_next <= avg_counter + 1;
            end
```

```
        end else begin
            avg_counter_next <= avg_counter;
        end

        // Error averaging accumulator
        if (avg_counter == 6'd0) begin
            error_avg_next <= {{4{error_in[7]}},error_in};
        end else if (avg_counter == 6'd7) begin
            error_avg_next <= 12'd0;
        end else begin
            error_avg_next <= error_avg_in + {{4{error_in[7]}},error_in};
        end

       // Passes final accumulator value to weight update after averaging
        if (avg_counter == 6'd7) begin
            error_update <= error_avg_final[10:3];
        end else begin
            error_update <= 8'd0;
        end
    end

    //Registers and wires for mu multiplication/shifting and zero-upon-saturate
    reg signed [6:0] mult_prod;
    wire signed [17:0] overflow;
    reg overflow_flag_mult;
    reg sign_flag_mult;
    wire signed [16:0] mult_pretrunc;
    wire [16:0] mult_pos;
    wire [23:0] mult_mu;
    wire [23:0] mult_mu_shift;

    //mult_pretrunc = data_in * error_update
    DW02_mult_inst_weight MWEIGHT (
        .inst_A(data_in),
        .inst_B(error_update),
        .inst_TC(1'b1),
        .inst_PRODUCT(mult_pretrunc)
    );

    //Converts negative two's complement numbers to positive for mu shifting
    assign mult_pos = (mult_pretrunc[16]) ? (~(mult_pretrunc[16:0])+1'b1) : mult_pretrunc[16:0];

    // mult_mu = mult_pos * mu_mult
    DW02_mult_inst_mu MMU (
        .inst_A(mult_pos[15:0]),
        .inst_B(mu_mult[7:0]),
        .inst_TC(1'b0),
        .inst_PRODUCT(mult_mu[23:0])
    );
    //Division via bitwise shift and conversion back to two's complement
    assign mult_mu_shift = (mult_pretrunc[16]) ? (~(mult_mu>>mu)+1'b1) : (mult_mu>>mu);
    //Weight update sum of existing weight and update
    assign overflow = {mult_mu_shift[16],mult_mu_shift[16:0]} + {{10{weight_in[7]}},weight_in};
    //Zero weight on overflow of +-127
    always @* begin
        case (overflow[17:7])
            11'b0000_0000_000: weight_out <= overflow[7:0];
            11'b1111_1111_111: weight_out <= overflow[7:0];
            default: weight_out <= 8'd0;
        endcase
    end
endmodule
```

```
module fir_tap(data_in, weight_in, data_out);
//By John Uehlin
//ECOG4 FPGA-based Artifact Canceller 2017
//FIR Tap Multiplier Module
    input signed [7:0] data_in;
    input signed [7:0] weight_in;
    output signed [7:0] data_out;
    wire dummy;

    //data_out = data_in*weight_in;
    DW02_mult_inst_tap MTAP (
        .inst_A({1'b0,data_in}),
        .inst_B(weight_in),
        .inst_TC(1'b1),
        .inst_PRODUCT({dummy,data_out})
    );
endmodule
```