Project Number: ECE-EXC-0501

Data Display, Acquisition and Feedback System for Biomedical Experiments

A Major Qualifying Project Report
submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
by

Patrick J. Bonneau

Anthony Fortunato

Jesse King

Date: February 15, 2006

Approved:

Professor Edward A. Clancy, Major Advisor

Abstract

Biomedical signals have various research applications in prosthetic limb development and other control applications. Consequently, a workstation that can be used to conduct biomedical experiments using EMG and other similar signals can be beneficial to the continuation of research in this growing field. We have investigated the possibility of creating a PC-based workstation to conduct these experiments using National Instrument’s LabVIEW. Our work suggests that such a system can not be used with experiments that require hard real-time control.

Acknowledgements

The authors would like to thank the advisor of this project, Ted Clancy, for his guidance, continual support and own personal work towards the advancement and completion of this project. The authors would also like to thank Garret Earnshaw, for his help in explaining the functionality of the PCI-6229, and Jason Thurlow, for his prompt response to all of our questions related to the PCI-DAS6402/16.

Authorship

Section	Section Title	Primary Author(s)
	Abstract	Pat
	Acknowledgements	Pat
	Authorship	Pat
	Executive Summary	Pat
1	Introduction	Pat
2	Background	Pat
2.1	Introduction to EMG	Pat
2.1.1	Sources and Characteristics of EMG	Tony
2.1.2	Methods of EMG Amplitude Estimation	Pat
2.1.2.1	Noise Rejection	Pat
2.1.2.2	Whitening	Pat
2.1.2.3	Multiple Channel Combination and Gain Normalization	Pat
2.1.2.4	Detection, Smoothing and Relinearization	Pat
2.2	Uses of EMG	Pat
2.2.1	Prosthetic Limb Development	Pat
2.2.2	Gait Rehabilitation	Pat
2.2.3	Other Control Systems	Pat
2.2.4	Discussion	Pat
2.3	Biomedical Signal Acquisition System	Pat
2.3.1	Embedded System	Pat
2.3.2	PC-based System	Pat
2.3.2.1	Data Acquisition Board Fundamentals	Pat
2.3.2.2	Benefits of a PC-based System	Pat
2.4	PC Operating System	Pat
2.4.1	Operating System Functions	Pat
2.4.1.1	Extended Machine	Pat
2.4.1.2	Resource Manager	Pat
2.4.2	Types of Operating Systems	Pat
2.4.3	Scheduling	Pat
2.4.3.1	General Scheduling Approaches	Pat
2.4.3.2	Scheduling in Real-Time Systems	Pat
2.4.3.3	Scheduling in Windows XP	Pat
2.5	Existing System	Pat
2.5.1	Hardware Overview	Pat
2.5.2	Software Overview	Pat
2.5.3	Discussion	Pat
2.6	Programming in LabVIEW	Pat/Tony
2.6.1	Basics	Pat/Tony

2.6.2	Calling Code from Other Languages	Pat/Tony
2.6.3	Linking MATLAB with LabVIEW	Pat/Tony
2.6.4	Linking C-code with LabVIEW	Pat/Tony
2.6.5	LabVIEW Analysis Methods	Pat/Tony
2.7	Summary	Pat
3.	Problem Statement	Pat
3.1	Problem Statement and Project Goals	Pat
3.2	Hard Real-Time Requirements	Pat
3.3	Objectives and Tasks	Pat
3.4	Summary	Pat
4	DAQ Board Performance Analysis	Pat
4.1	Measurement Computing PCI-DAS6402/16	Pat
4.1.1	Hardware Overview	Pat
4.1.2	LabVIEW Universal Library Functions	Pat
4.2	NI PCI-6229	Pat
4.2.1	Hardware Overview	Pat
4.2.2	NI-DAQmX LabVIEW Function	Pat
4.3	DAQ Board Comparisons	Pat
4.3.1	Data Transfer	Pat
4.3.2	Simultaneous Input and Output	Pat
4.4	Discussion	Pat
4.5	Chapter Summary	Pat
5.	LabVIEW Analysis Method Comparison	Pat
5.1	EMG Amplitude Estimation VIs	Jesse
5.2	Latency Test Procedure	Tony
5.2.1	Timed-Loop Vi	Pat
5.2.2	Analysis Method Comparison Procedure	Tony
5.3	Timed-Loop Test Results	Tony/Pat
5.4	Discussion	Tony/Pat
5.5	Chapter Summary	Tony/Pat
6.	Hard Real-Time LabVIEW Performance	Pat
6.1	Final Testing Procedure – MTBF	Tony
6.2	Results	Tony/Pat
6.3	Discussion of Results	Tony/Pat
6.4	Limitation of Windows Operating System	Tony
6.5	Chapter Summary	Pat
7	Using LabVIEW in a Data-Logging Application	Tony
8	Future Recommendations	Jesse
8.1	Other LabVIEW Design Options	Jesse
8.2	Other Alternatives	Jesse
8.3	Chapter Summary	Jesse
9	Conclusion	Pat
Appendix A	Compiling CIN Source Code	Tony
Appendix B	C Code for Choosing an Array of Filter Coefficients	Tony

Appendix C	**MATLAB Code to Simulate Adaptive Whitening Filter Parameters**	**Jesse**
Appendix D	**MATLAB Code that Creates Adaptive Whitening Filter Coefficients**	
Appendix E	**MATLAB Script Node Latency Test**	**Pat**
Appendix F	**Latency Test Results for EMG Algorithm**	**Pat**
Appendix G	**Testing to Evaluate LabVIEW Real-Time Performance**	**Pat**
Appendix H	**Detailed Description of EMG Amplitude VIs**	**Jesse**

- **Final Corrections were performed by Pat and Jesse**

Table of Contents

Table of Figures

Table of Tables

Executive Summary

The electrical activity of human muscle, as well as the various mechanical parameters related to human motion, can be used in a wide array of research studies. For instance, electromyogram (EMG) signals are currently being utilized in the development of artificial limbs and in the gait rehabilitation of stroke and brain injury patients [4], [5], [16], [23]. Hence, a workstation that can be used to acquire, analyze and display EMG and other similar signals can be beneficial to the continuation of research related to biomedical signal analysis.

A workstation that will be used to acquire and analyze the electrical activity of human muscle may need to operate in hard real-time for certain control applications. Operation in hard real-time refers to the notion that the system must not only perform logically correct computations but also operate under critical time-constraints as well [14]. The reason that a hard real-time system is required is that most of the previously described applications for this system involve feedback in a timely manner. For instance, signals analyzed by the workstation might be used to control the movement of a prosthesis. In order to accurately work like a human limb, the prosthesis must move within a certain window of time when it receives the appropriate stimulation and, consequently, not react after a long delay.

There are at least two fundamental ways that a system that can be used to conduct various biomedical experiments can be designed. The first option is to design an embedded system that can only perform the acquisition task. While this is a feasible approach, there are several drawbacks to such an approach. A standalone embedded

system is often difficult to program and maintain over long periods of time. Moreover, an embedded system might require the development of customized hardware to implement various algorithms. The drawback to making customized hardware is the notion that duplicating the hardware can be difficult and the hardware might need to be frequently updated when new analysis algorithms are implemented.

A second approach is to design a PC-based system. This type of system utilizes a standard desktop computer, with a data acquisition (DAQ) card and commercially available components to amplify and isolate signals. The PC would then have a software package that can be executed to perform desired biomedical signal acquisition experiments. A PC-based workstation is cost-effective, as it bypasses the need for custom instrumentation by using a currently available PC and inexpensive acquisition equipment.

However, the components chosen to operate with the PC must be selected carefully in order to optimize system performance. In particular, DAQ boards do not follow a universal convention of operation. This notion can be seen by comparing two commercial DAQ boards: National Instrument's PCI-6229 and Measurement Computing's PCI-DAS6402/16. While both boards have a FIFO buffer and transfer data between the PC and computer via DMA, data transfer is handled differently with each board. The PCI-DAS6402 must collect a minimum of 1024 data sample before the board can generate an interrupt and send data to the PC. To the contrary, the PCI-6229 does not have this minimum collection requirement and can even transfer a single datum point per sample by using a clock that is present on the device and bypassing its FIFO. Moreover, in order to conduct a control application, the DAQ board must be able to simultaneously input and output data. While both of the previously described boards contain analog input and output channels, only the PCI-6229 has the architecture to support simultaneous input and output.

Consequently, we have created a PC-based workstation, which will be utilized to conduct these various biomedical experiments, with the PCI-6229 DAQ board. Moreover, the software that we utilized is National Instruments’ LabVIEW. LabVIEW is a graphical programming language, for the Windows operating system, that is used for various signal processing applications. Since LabVIEW is a graphical programming language, it is fairly easy to learn and operate. Similarly, since the workstation will be run on a Windows PC, it can readily be used and adopted by a wide audience. Furthermore, drivers for data acquisition boards are readily available for use with LabVIEW. As a result, by using LabVIEW on a Windows PC, the need for writing customized hardware drivers was avoided.

While we used LabVIEW for a hard real-time application, LabVIEW was originally intended for soft real-time applications. Moreover, while Windows XP allows for the availability of data acquisition board drivers, it is not a real-time operating system. Therefore, the objective of this project was to determine the extent to which our created system can be used for a hard real-time application. In particular, our goal was to determine if biomedical experiments can be conducted with LabVIEW with latencies no greater then five milliseconds. This five millisecond worst case latency was chosen to ensure that a computer monitor used to display graphical data will have a real-time screen refresh rate.

In order to make this determination, a comparison of the various LabVIEW analysis methods was conducted. LabVIEW has two standard analysis methods: array-based and point-by-point. With point-by-point analysis, a datum point is operated on as soon as it becomes available. On the contrary, with array-based analysis, all computations are array oriented. As a result, when array-based analysis is utilized, a buffer is first defined. Data are only analyzed when a buffer unit is available.

Choosing which analysis method to utilize in a given application simply involves choosing which function library to utilize in LabVIEW. If point-by-point analysis is desired, one must merely choose functions listed in the point-by-point subsection of the LabVIEW function palette. In order to use array-based analysis, the user has two options. First, the standard functions available use array-based analysis. Second, the user can use Express VIs. Express VIs provide numerous different function capabilities in a single module and can be figured with a simple graphical user interface.

In order to determine which analysis method induced the smallest latency, a module was created to record the time it takes for a LabVIEW routine to operate. Three EMG amplitude estimation routines were created, using each of the previously identified LabVIEW function routine, and were then timed with this module. Our results indicated that, on average, our point-by-point and standard array-based algorithm (using a buffer size of one) garnered average latencies that were below our maximum latency standards. Thus, initial results indicated that it seemed possible to create a hard real-time control experiment using LabVIEW.

However, while the average latencies induced by these EMG algorithms were below our maximum acceptable latency, our statistics showed occasional outliers. Hence, a second test was performed to determine if a simultaneous input and output task, executing the previously tested point-by-point EMG amplitude estimation technique, could keep up without losing datum points any datum points do to these occasional outliers. Our tests showed that for sampling frequencies above approximately 500 Hz, datum points were lost on the order of every fraction of a second without any computation being performed. When a sampling frequency of 250 Hz was utilized, it was viewed that over seventeen hours passed without any datum point being lost without any computation being conducted. However, once a simple FIR point-by-point filter with 500 filter coefficients was introduced as computation, a datum point was lost on the order of once every fourteen seconds.

This final test indicated that our PC-based system failed at conducting hard real-time applications due to the frequency of datum points being lost with only minimum computation and not even a full EMG amplitude estimation routine. However, while hard-real time experiment can not be conducted with this workstation, it is possible to create a data logging application. That is, data can be streamed to disk and then be operated on offline.

1. Introduction

The electrical activity of human muscle, as well as the various mechanical parameters related to human motion, can be used in a wide array of research studies. For instance, electromyogram (EMG) signals are currently being utilized in the development of artificial limbs and in the gait rehabilitation of stroke and brain injury patients [4], [5], [16], [23]. Hence, a workstation that can be used to acquire, analyze and display EMG and other similar signals can be beneficial to the continuation of research related to biomedical signal analysis.

A workstation that will be used to acquire and analyze the electrical activity of human muscle may need to operate in hard real-time for certain control applications. Operation in hard real-time refers to the notion that the system must not only perform logically correct computations but also operate under critical time-constraints as well [14]. The reason that a hard real-time system is required is that most of the previously described applications for this system involve feedback in a timely manner. For instance, signals analyzed by the workstation might be used to control the movement of a prosthesis. In order to accurately work like a human limb, the prosthesis must move within a certain window of time when it receives the appropriate stimulation and, consequently, not react after a long delay.

There are at least two fundamental ways that a system that can be used to conduct various biomedical experiments can be designed. The first option is to design an embedded system that can only perform the acquisition task. While this is a feasible approach, there are several drawbacks to such an approach. A standalone embedded system is often difficult to program and maintain over long periods of time. Moreover, an embedded system might require the development of customized hardware to implement various algorithms. The drawback to making customized hardware is the notion that duplicating the hardware can be difficult and the hardware might need to be frequently updated when new analysis algorithms are implemented.

The second approach is to design a PC-based system. This type of system utilizes a standard desktop computer, with a data acquisition card and commercially available components to amplify and isolate signals. The PC would then have a software package that can be executed to perform desired biomedical signal acquisition experiments. A PC-based workstation is cost-effective, as it bypasses the need for custom instrumentation by using a currently available PC and inexpensive acquisition equipment.

Consequently, we have created a PC-based workstation that will be utilized to conduct these various biomedical experiments. Moreover, the software that we utilized is National Instruments' LabVIEW. LabVIEW is a graphical programming language, for the Windows operating system, that is used for various signal processing applications. Since it is a graphical programming language, it is fairly easy to learn and operate. Similarly, since the workstation will be run on a Windows PC, it can readily be used and adopted by a wide audience. Furthermore, drivers for data acquisition boards are readily available for use with LabVIEW. As a result, by using LabVIEW on a Windows PC, the need for writing customized hardware drivers was avoided.

While we used LabVIEW for a hard real-time application, LabVIEW was originally intended for soft real-time applications. Moreover, while Windows XP allows for the availability of data acquisition board drivers, it is not a real-time operating system. Therefore, the purpose of this project was to determine the extent to which LabVIEW can be used for a hard real-time application. In particular, our goal was to determine if biomedical experiments can be conducted with LabVIEW with latencies no greater then five milliseconds. This five millisecond worst case latency was chosen to ensure that a computer monitor used to display graphical data will have a real-time screen refresh rate.

Although tests conducted indicated that the average latencies generated by LabVIEW VIs could be on the order of fractions of milliseconds, outliers that well exceeded our latency requirements were frequently observed. These large outliers are a result of the limitations of the Windows XP operating system. Hence, our results indicate that a PC-based hard real-time system with our given latency requirements can not be implemented using standard LabVIEW running on the Windows XP operating system. Consequently, future recommendations are provided that offer alternative methods to implementing a workstation to conduct biomedical experiments.

2. Background

This chapter presents information on the background of the data display, acquisition and feedback system for biomedical experiments that we have created. First, an introduction to EMG signals is presented along with the various applications that can be conducted by utilizing these signals. Second, the general structure of biomedical signal acquisition systems is discussed. Third, the properties of operating systems that are used on PC-based acquisition systems are presented. Fourth, an existing acquisition system is described and its deficiencies are explored. Last, a common software package that is used in signal processing applications, LabVIEW, is introduced.

2.1 Introduction to EMG

There are various biomedical signals that will be analyzed with the workstation that we have created. In this chapter an introduction is given to one of these many biomedical signals: the electromyogram (EMG) signal. After giving basic information about the characteristic of EMG signals, examples depicting the type of applications that EMG signals are used in are presented.

2.1.1 Sources and Characteristics of EMG

The nervous system controls the voluntary movement of various body parts in humans by contracting and relaxing various skeletal muscles. To instantiate a contraction, a neuron generates a small electrical potential on the surface of the muscle fiber. This electrical potential causes depolarization of the muscle fiber tissue and a following depolarization waveform. This waveform travels the length of the muscle fiber and is known as the *action potential (AP)*. Figure 2.1 depicts the generation of electric fields in muscle fibers.

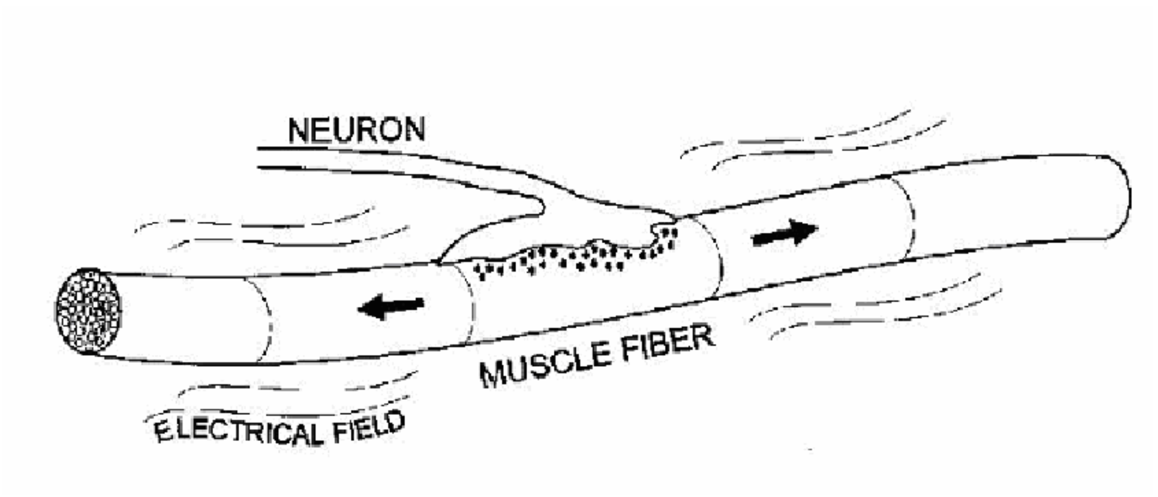


Figure 2.1 – Motor Neuron Induced Electrical Field in Muscle Fiber [25]

Muscle fibers are excited by nerve branches from one motoneuron in groups known as motor units. These motor units are defined as the fundamental unit of contraction and can range from a few muscle fibers for small muscles such as those in the hand and fingers, to thousands of muscle fibers in large muscles. Because each motor unit contains a number of muscle fibers that are attached to the motor neuron at various points, the electrical signal of a motor unit is the summation of the action potential of each muscle fiber, which may be phase shifted from the other muscle fibers in that unit. This notion is reinforced in Figure 2.2. The electrical potential due to contraction of all fibers in a motor unit during a single activation is referred to as the motor unit action potential (MUAP). This MUAP can be recorded by using electrodes placed on the surface of the skin above the muscle. [1]

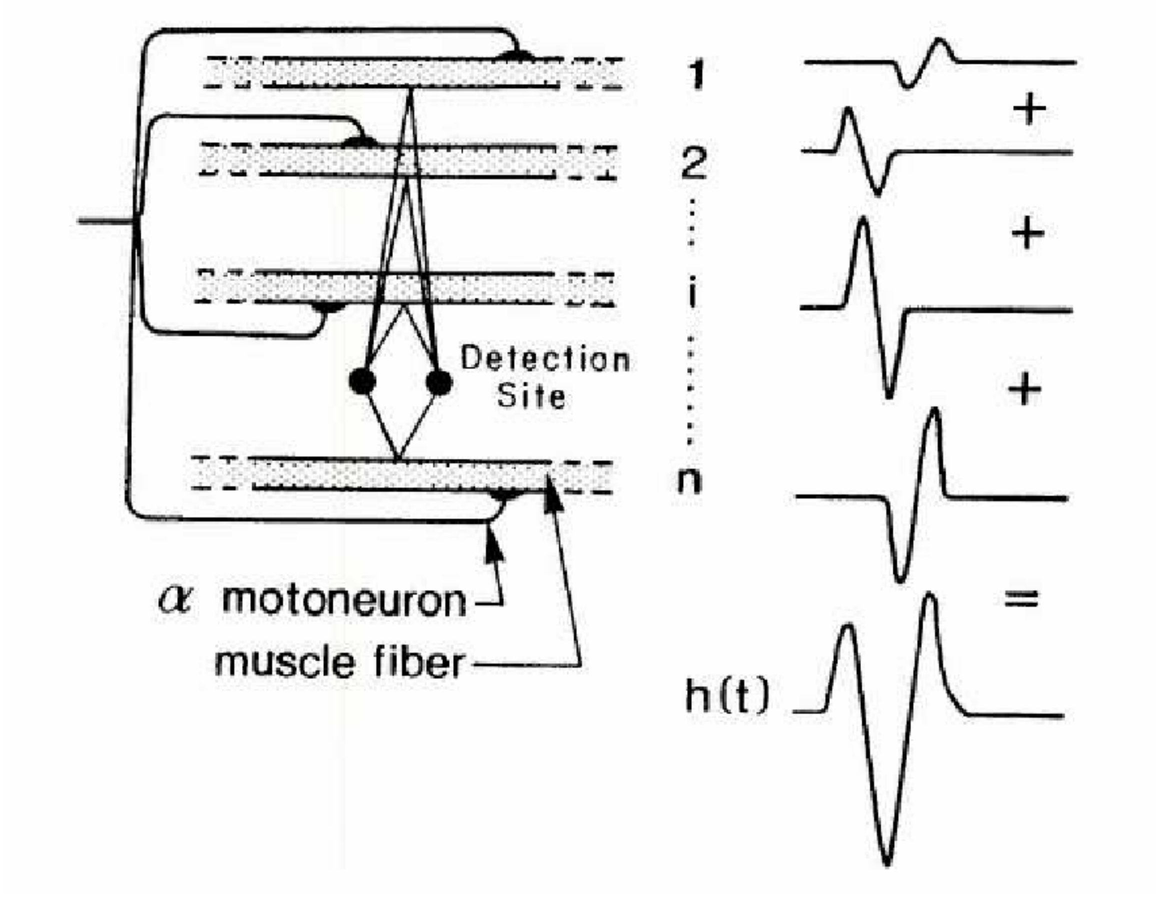


Figure 2.2 – Motor Unit Action Potential [14]

Also, a muscle is not typically excited by only one action potential. In order to hold a contraction for any length of time, the motor units must be repeatedly activated. This repeated activation gives rise to a series of MUAPs that can be modeled as a pulse train in classical signal processing terms. This series of MUAPs that is produced is referred to as a motor unit action potential train (MUAPT). When measured using a surface electrode, the electromyogram can be defined as the superposition of numerous MUAPTs firing asynchronously. Figure 2.3 reinforces the notion that the superposition of motor unit action potentials gives rise to surface EMG. [1]

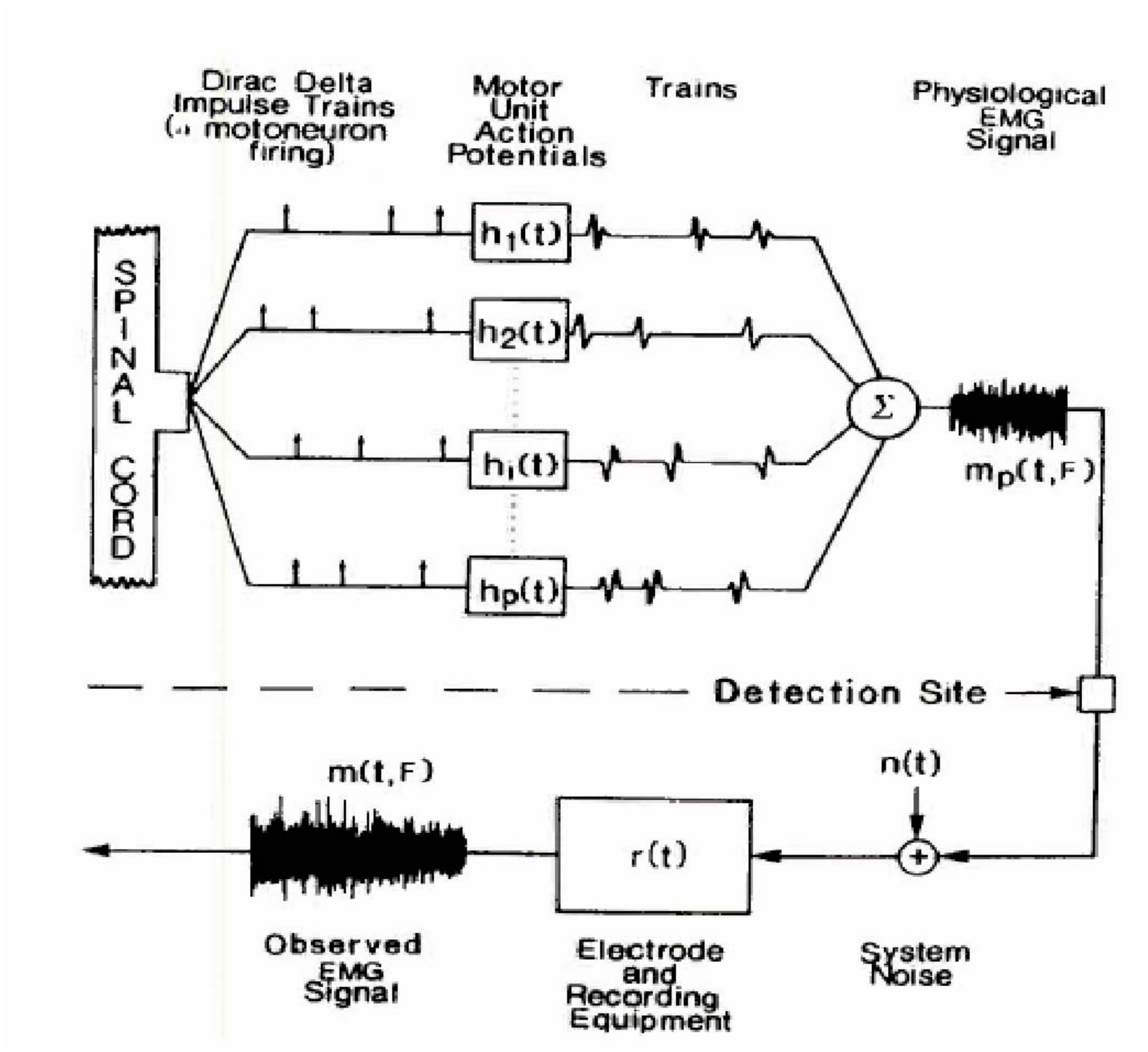


Figure 2.3 – Superposition of Motor Unit Action Potential Give Rise to Surface EMG [12]

The surface electromyogram signal typically does not exceed 5-10 mV in amplitude with the majority of signal information being contained between the frequencies of 15 and 400 Hz. The signal is an interference pattern which grows with muscle effort. As a result, the amplitude of the EMG contains a great deal of the signal information which can be modeled as a Gaussian random process. The EMG amplitude can thus be defined as the time-varying standard deviation of the EMG signal and is a measure of the activity level of the muscle under observation [8]. The next section contains further details regarding techniques used to perform EMG amplitude estimates.

2.1.2 Methods for Amplitude Estimation

Currently, the most popular EMG amplitude estimation techniques are the statistical moving average mean absolute value (MAV) and moving average root mean square (RMS). Below are the equations for these two estimation techniques.

$$MAV_t = \frac{1}{N} \sum_{i=t-N+1}^{t} |x_i| \qquad (1)$$

$$RMS_t = \sqrt{\frac{1}{N} \sum_{i=t-N+1}^{t} x_i^2} \qquad (2)$$

In the above equations *N* is the number of samples in each smoothing window, *t* is the time at which the interval starts and x_i is the signal being smoothed in the time-domain. Research has shown that amplitude estimates using MAV and RMS calculations exhibit similar results. [8]

Other signal processing techniques are often using in conjunction with MAV and RMS calculations to improve EMG amplitude estimates. In particular, research has indicated that whitening an EMG signal prior to MAV and RMS calculations results in enhanced estimates. Figure 2.4 displays a six stage process that may be used to compute EMG amplitude estimates with whitening prior to detection of the signal. This six stage process is comprised of noise rejection, whitening, gain normalization, detection, smoothing and relinearization.

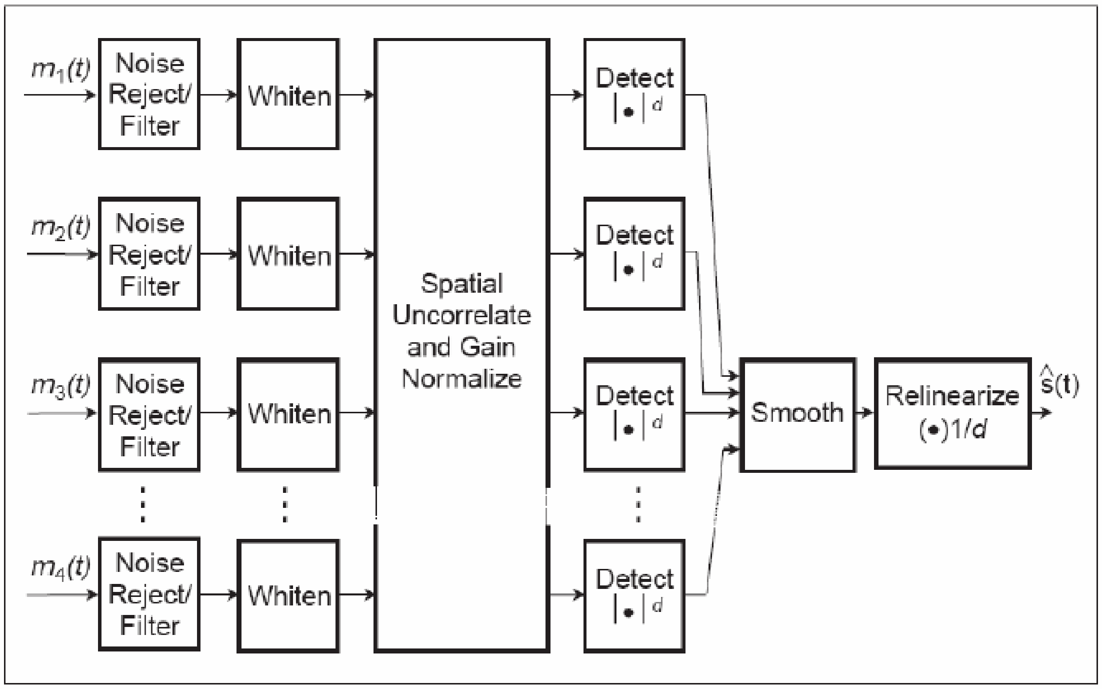


Figure 2.4 – Standard Six Stage Filtering Process for EMG Amplitude Estimation [10]

2.1.2.1 Noise Rejection

As stated previously, the majority of EMG signal information is contained in the frequencies between 15 and 400 Hz. However, when EMG signals are measured, noise is also present with the valid signal information. There are various sources for this noise present in EMG signals. One source for this EMG noise is a result of electrode motion artifact. This electrode motion artifact can be caused by mechanical disturbance of the electrode charge layer and deformation of the skin under the electrodes. Artifact due to electrode motion is present below 20 Hz. Also cable motion artifact is another possible source for EMG noise. The cables that connect the electrodes to the electrode amplifiers have an intrinsic capacitance. Consequently, if these wires are subjected to an electric or magnetic field, current is generated which, in turn, will induce a voltage. The cable motion artifact typically has a frequency range of 1 to 50 MHz. [10]

In order to suppress the noise present in EMG signal, a high-pass filter is commonly used. This high-pass filter can be incorporated into the hardware instrument that is acquiring the EMG or it may be implemented in software. Implementing the filter in software generally makes it easier to implement higher order filters. Consequently, these filters can have cutoff frequencies as close to 15 Hz as possible and, thus, the loss of useful signal information is minimal. However, even if a highpass filter is modeled in software, a hardware highpass filter is still typically necessary in order to avoid saturation.

2.1.2.2 Whitening

A whitening filter is used to improve EMG amplitude estimation by removing EMG signal correlation and measurement noise. A fixed whitening filter is a finite impulse response (FIR) filter that has the characteristic shape similar to a high-pass filter. Ideally, the optimal shape of the complete whitening and noise-rejection filter is a function of measurement noise and muscle activation level. However, for contraction below 10% of the MVC, it is hard to distinguish between signal and noise. Adaptive whitening filters that whiten the signal for strong contractions but progressively turn off whitening for weak contractions have been developed. These adaptive whitening techniques have resulted in an amplitude estimation improvement of approximately 10% when compared to unwhitened amplitude estimates. [8]

In order to alleviate as much additive noise as possible, Clancy and Farry recently modeled the noise levels of the EMG and modified the adaptive whitening process. Figure 2.5 shows the functional model of the EMG proposed by Clancy and Farry. The signal w_i is a zero mean random Gaussian process that is used as a start for modeling the EMG. The signal w_i is passed through a shaping filter, H_{time}, to create the signal n_i. This shaping filter serves to create the low-pass characteristic shape of an EMG signal while still maintaining unit standard deviation. Next, n_i is multiplied by the amplitude of the EMG, s_i, to give the model of the noise-free EMG signal, r_i. The additive measurement noise is represented as the zero mean random process, v_i, and is lastly summed with r_i to complete the model of the surface EMG. [8]

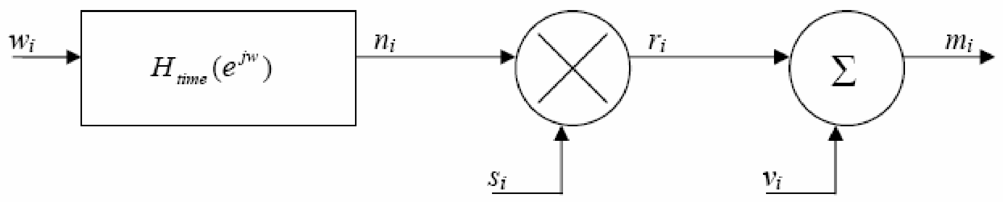


Figure 2.5 – Model of EMG Used for Adaptive Whitening Filter [8]

Figure 2.6 shows the three stage of the adaptive whitening filter proposed by Clancy and Farry. The first stage of this process whitens the noiseless EMG amplitude, s_i, along with a filtered version of the additive noise v_i. The second stage optimally estimates the noiseless whitened signal by adaptively removing the noise via a Weiner filter. The final stage of this process applies an adaptive gain. This gain is determined by the transformations of the EMG signal in the previous steps.

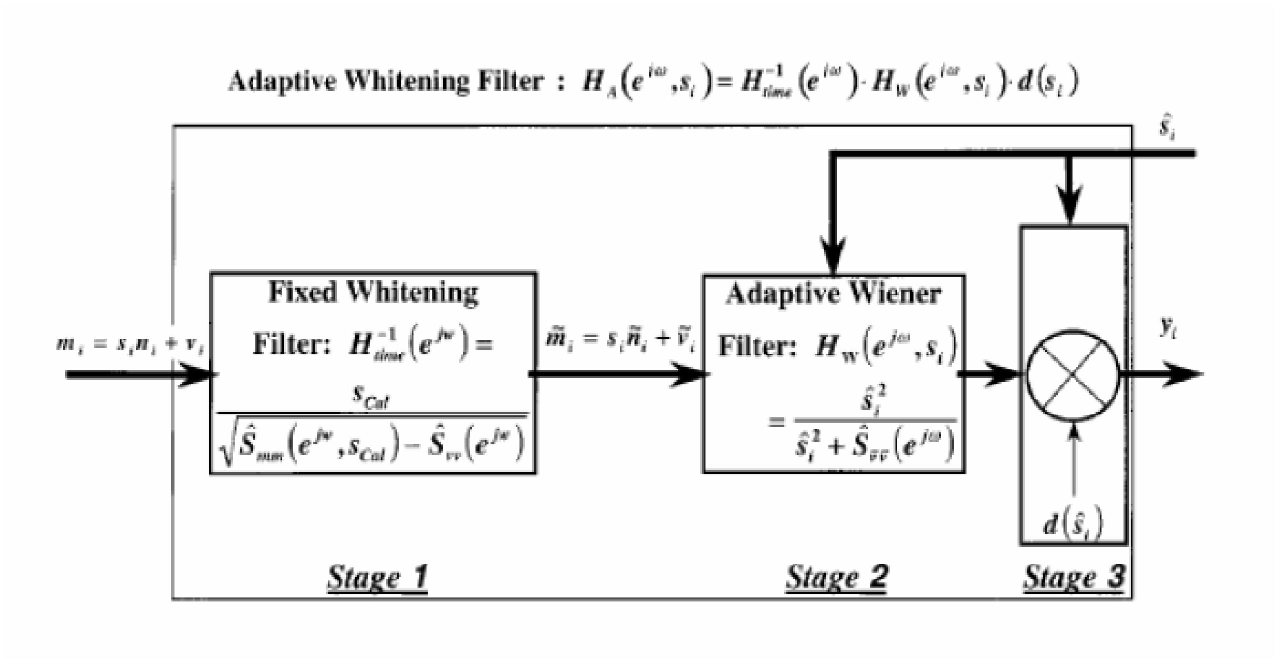

Adaptive Whitening Filter : $H_A(e^{j\omega}, s_i) = H^{-1}_{time}(e^{j\omega}) \cdot H_W(e^{j\omega}, s_i) \cdot d(s_i)$
$\hat{s}_i$
$m_i = s_i n_i + v_i$
Fixed Whitening Filter: $H^{-1}_{time}(e^{jw}) = \frac{s_{Cal}}{\sqrt{\hat{S}_{mm}(e^{jw}, s_{Cal}) - \hat{S}_{vv}(e^{jw})}}$
$\tilde{m}_i = s_i \tilde{n}_i + \tilde{v}_i$
Adaptive Wiener Filter: $H_W(e^{j\omega}, s_i) = \frac{\hat{s}_i^2}{\hat{s}_i^2 + \hat{S}_{\tilde{v}\tilde{v}}(e^{j\omega})}$
y_i
$d(\hat{s}_i)$
Stage 1
Stage 2
Stage 3

Figure 2.6 – Clancy and Farry's Adaptive Whitening Stage of EMG Amplitude Estimation [8]

For the purpose of our project, this entire process can be implemented as a time-varying FIR filter. Consequently, numerous different FIR filters will be pre-compiled and stored in a matrix. Each sample is filtered by one of these many FIR filters stored in the matrix. The filter chosen to operate on a given sample is selected depending on the current value of the amplitude estimate.

2.1.2.3 Multiple Channel Combination and Gain Normalization

This next step involves the combination of EMG recordings obtained from several electrodes placed adjacent to each other on the skin surface above the same muscle. The combination of multiple channels is performed because the signal-to-noise ratio (SNR) improves with an increase in the volume of muscles recorded. Since the gain and the distance from the muscle differ for each electrode used, the gain normalization process occurs after the channels are combined.

Research has indicated that using several electrodes for measuring the EMG of a muscle greatly increases EMG amplitude estimation accuracy. In particular, SNR performance improvements of up to 91% have been observed when using multiple channels compared to using a single channel [8]. While there are benefits of using multiple channels, there are also some disadvantages. In particular, the chance of defects that may arise due to noise, shorted electrodes and other abnormalities increases when the amount of channels used is increased.

2.1.2.4 Detection, Smoothing and Relinearization

Once the signal is whitened, and combined if multiple channels are used, the actual amplitude estimation process begins. The remaining three parts of amplitude estimation are detection, smoothing and relinearization. The methods used to implement these three processes vary depending on whether RMS or MAV is being used to provide an amplitude estimate. [2]

When an RMS amplitude estimate is conducted, detection consists of squaring all the points in the signal window. Once detection is complete, the information is then placed through a smoothing filter. A smoothing filter can be implemented as a moving average filter or any other FIR lowpass filter. The estimate is then relinearized by taking the square root of the smoothing function output.

When an MAV amplitude estimate is conducted, detection consists of taking the absolute value of each point in the signal window. Again, once detection is complete, the information is smoothed by utilizing either a moving average filter or another FIR lowpass filter. When a MAV amplitude estimate is conducted, there is no relinearization step.

2.2 Uses of EMG

With a basic overview of EMG signals presented, we will now give an outline of some of the various manners that EMG signals are used in research activities. These are by no means all of the applications of EMG signals. Rather, this section serves to give an idea of the type of experiments that our workstation will help to conduct.

2.2.1 Prosthetic Limb Development

As previously discussed EMG signals provide a non-invasive measure of ongoing muscle activity. Therefore, EMG signals can be potentially used for controlling robotic prosthetic devices. Most prosthetic devices that are currently available usually only have one degree-of-freedom. As a result, these devices provide nowhere near the amount of control as the original limb which they are intending to replace. Through clinical research, it has been shown that amputees and partially paralyzed individuals typically have intact muscles that they can exercise varying degrees of control over. As a result, research is being conducted in regards to utilizing the signals from these intact muscles to control robotic devices with multiple degrees of freedom. [11]

The EMG has been used in two manners in the area of prosthetic limb development. The first approach is for the subject to exert a force with a particular muscle. This force results in a steady-state EMG signal amplitude estimate. A degree of freedom of a robotic limb is then moved in proportion to the EMG amplitude. This described approach is used in the control of a standard prosthetic gripper that has one degree-of-freedom. [11]

The second manner that EMG signals are used involves discrete actions. When a discrete action is performed, such as the quick movement of the hand or arm, the surface EMG is obtained from various muscle cites. The temporal structure of the transient EMG activity is then analyzed. Upon analyzing the transient EMG activity, various movements can be classified. For instance, a research group at the University of Washington was able to use EMG signals to control a robotic device with four degrees-of-freedom [11]. Moreover, researchers at the University of New Brunswick have done a tremendous amount of work in the area of prosthetic limb control, which can be seen in [4], [5] and [16]. Hence, EMG signals can be used in the development of advanced prosthetic devices that have various degrees-of-freedom.

2.2.2 Gait Rehabilitation

Another area in which EMG signals are utilized is in gait rehabilitation for brain injury and stroke patients. A typical neurorehabilitation program involves manual assisted treadmill therapy. While clinical research has shown that this type of therapy is beneficial, it has several limitations. Most notably, manual-assisted treadmill therapy places a significant burden on physical therapists because the rehabilitation procedure requires several physical therapists to physically move the legs of the patient through each step [32].

An alternative to manual-assisted treadmill therapy is to use robotic devices to automate gait therapy. Automating gait therapy has several advantages. For instance, only one physical therapist is needed for each rehabilitation session. Also, automated therapy sessions result in longer sessions that are not only safer for patients but are easily reproducible. While there are several advantages to automated gait therapy, there is a glaring disadvantage. The human gait cycle is complicated and the robotic system must limit the degrees of freedom through which the subject is able to walk due to safety and control complexity. As a result, robotic devices are unable to take into account all of the key gait determinants. [32]

Consequently, testing needs to be done to determine if muscle activation patterns exhibited during treadmill walking are altered when a robotic aid is used. This analysis of muscle patterns is performed by analyzing EMG signals. Such tests have been performed at Catholic University in Washington DC, where devices are being developed for gait rehabilitation. One of these devices is a robotic gait orthosis known as the Lokomat. In order to test the effectiveness of this device, EMG signals generated in normal subjects while walking on a treadmill were compared to those generated by individuals using the Lokomat. Therefore, EMG signals provide a mechanism for testing the effectiveness of gait therapy. [32]

2.2.3 Other Control Systems

While EMG signals have been used in the areas of prosthetic limb development and gait rehabilitation, EMG signals have also been used in a variety of other control applications for the paraplegic and partially paralyzed. One application has been developed that enables disabled individuals to utilize computers. One of the chief components of a computer is the mouse. In order to operate many computer applications, the user is expected to move the mouse cursor on the screen and click on various graphical buttons. Individuals with severe muscle disabilities are unable to move their arms or feet and, therefore, are not able to use the traditional point-and-click features of a mouse. [22]

While these individuals are unable to move their arms or legs, many of these patients still have control over their facial muscles. As a result, a system has been developed that allows the user to move the cursor on a screen by the movement of their cranial muscles. In particular, the current system detects the EMG signals associated with the temporalis muscles and muscles associated with the raising and lowering of the eyebrow through three electrodes. These signals are then processed through a digital signal processing board and result in the movement of the cursor on the screen. [22]

2.2.4 Discussion

One of the chief biomedical signals that will be analyzed on our workstation is the electromyogram (EMG) signal. As shown throughout this chapter, EMG signals are used for a variety of activities, ranging from the development of prosthetic limbs to the analysis of gait rehabilitation therapy. By showing some of the research activities that EMG signals are used in, the function of our workstation becomes more apparent. Our workstation is to acquire biomedical signals and perform analysis to allow for the various experiments associated with the different applications of EMG signals to be conducted. Furthermore, our workstation should also be able to analyze other similar biomedical signals in other biomedical experiments. In the next section, the two basic forms of biomedical signal acquisition systems will be discussed.

2.3 Biomedical Signal Acquisition Systems

From the previous sections of this chapter, it is apparent that EMG signals have a wide array of applications in various research areas. While only EMG signals were discussed in the previous section, other biomedical signals are equally important in research. In order for these biomedical signals to be used in their various applications, a system must be available that can acquire, analyze and display these various signals. Although there are many design options available for creating such a workstation, two of the most common approaches are to either develop an embedded or PC-Based System.

2.3.1 Embedded System

Biomedical signal acquisition systems are conventionally implemented as an embedded system. An embedded system is some combination of computer hardware and software that is specifically designed for a particular application. For instance, using digital signal processing (DSP) boards, individuals are able to design and implement customized signal processing algorithms. Although embedded systems typically have a real-time operating system, individuals working with embedded systems have to be aware of the low-level details of how to program and interact with all of the hardware components of the system. Consequently, programming embedded systems to perform signal analysis routines is often cumbersome and difficult.

Also, there are several EMG amplitude estimator systems that rely on the development of custom hardware to implement their algorithms. Such customized hardware systems have several drawbacks. First, it is not only fairly expensive but often difficult to duplicate such systems. Systems based on customized hardware are hard to modify for different applications. For instance, while the basic signal acquisition systems are very similar, an individual using hardware to develop an EMG workstation would have to develop an entirely new system to analyze ECG signals. Similarly, when new EMG amplitude estimation algorithms are developed the entire hardware configuration of the system will have to be modified to accommodate the new algorithm. [7]

2.3.2 PC-Based System

Another option available for creating a system to acquire and analyze biomedical signals is to develop a system that is based around a standard PC. A PC-based system is comprised of a standard desktop computer that contains signal analysis software. However, in order for the computer to acquire the biomedical signals that are analyzed via software, a commercially available data acquisition (DAQ) board must be installed in the computer.

2.3.2.1 Data Acquisition Board Fundamentals

A DAQ board is a basic A/D converter that is coupled with an interface that allows a PC to control the actions of the A/D and capture the digital output information from the converter. A DAQ board is designed to plug directly into a personal computer's bus, with all the power required for the A/D converter and associated interface components being directly obtained from the bus. Moreover, it should be noted that a DAQ board is more than a simple A/D function on a board. A data acquisition may include discrete digital bi-directional I/O lines, counter timers, and D/A converters for outputting analog signals for control applications.

The purpose of a DAQ board is to convert analog data to digital data that a computer is able to manipulate. There are three primary methods available to transfer digitized data between the DAQ board and computer memory. These three methods for data transfer are direct memory access (DMA), interrupt, and programmed I/O transfers. For programmed I/O transfers, data are transferred between the CPU and the PC whenever the CPU receives a software code to either acquire or generate a single data point. Interrupt data transfers occur when the DAQ board sends an interrupt to the CPU. This interrupt causes the CPU to either read acquired data from the DAQ board or write data to the DAQ board. DMA transfers use a DMA controller instead of the CPU to move acquired data between the board and computer memory. Even though high-speed data transfers can occur with interrupt and programmed I/O transfers, they require the use of the CPU to transfer data. Consequently, DMA transfers are able to acquire data at high speeds and keep the CPU free for performing other tasks at the same time.

2.3.2.2 Benefits of a PC-Based System

While using standalone embedded systems is a viable method of acquiring and analyzing biomedical signals, a PC-based system is a cost effective alternative. Individuals are able to share the cost of buying the PC by utilizing it for other general purpose applications, such as word processing or web browsing. Moreover, the other external components that are utilized in the system can be purchased commercially. Hence, the development of customizable hardware can be avoided or reduced. However, if a PC-based system is implemented, another piece of software must be taken into account that is not utilized in a system designed to execute only one task. That piece of software is known as an operating system and it will be discussed in the following section.

2.4 Operating System Information

In the previous section, the notion that a biomedical signal acquisition system can be either embedded or PC-based was discussed. In both implementations, a piece of software is capable of being utilized, known as the operating system. An operating system allows for numerous applications to be executed on a single workstation, instead of just one customized application. In the following sections some of the basic functions of an operating system will be discussed.

2.4.1 Operating System Functions

As discussed in the previous section, the key software component that allows for a workstation to execute more then one application is the operating system. An operating system has two basic functions: extending the machine and managing resources. In the following sections these two seemingly unrelated functions will be discussed.

2.4.1.1 Extended Machine

At the machine language level, the architecture of most computers is primitive and awkward to program. While the designer of an embedded computer system needs to be aware of the low-level details of how to program and interact with all of the hardware components of an embedded system, an individual using an operating system does not need to be fully aware of these details. Thus, the function of the operating system is to present the user with the equivalent of an extended machine that is easier to program than the underlying hardware. The operating system achieves this goal by providing a variety of services that programs can obtain using special instructions called system calls. [29]

2.4.1.2 Resource Manager

An alternative, but equally important function of an operating system is to manage all of the pieces of a complex system. Modern computers consist of wide array of devices, including processors, memories and I/O devices. The job of the operating system is to provide for an orderly allocation of these various devices among the various programs competing for them.

Resource management includes multiplexing resources in two ways: in time and in space. When a resource is time multiplexed, different programs take turns using it. Determining how the resource is time multiplexed is the job of the operating system. The other kind of multiplexing that the operating system is responsible for is space multiplexing. In some instances, devices get parts of numerous resources at a time. For instance, main memory is normally divided up among several running programs or processes. Similarly, allocating disk space and keeping track of what process uses each disk block is also the job of the operating system. [29]

2.4.2 Types of Operating Systems

In the previous sections the general functions of an operating system were presented. However, there are a variety of different types of operating systems that are available, each with their own strengths and characteristics. For instance there are operating systems that are appropriate for use with mainframes, servers and multiprocessor computers. The most widely known operating system is the Personal Computer Operating System. The main job of this operating system is to provide a good interface to a single user. Microsoft Windows is the most widely known of this type of general purpose operating system. Moreover, as previously discussed, there are operating systems for embedded systems. These operating systems do not provide the simple interface that PC operating systems provide but, instead, are concerned with deterministic control over multiple routines. [29]

Another type of operating system is the real-time system. These systems are distinguished by having time as a key consideration. For instance, in control applications, real-time computers have hard deadlines that must be met. For example, in an assembly line at a car manufacturing plant, if a welding robot welds too late or too early then the car will be ruined. Hence, if an action absolutely must occur at a certain moment or within a certain time interval, the system is deemed to be a hard real-time system.

Another kind of real-time system is a soft real-time system. In these types of system, missing an occasional deadline is acceptable. For instance, an example of a soft real-time system would be a live audio-video system. In this type of system, violation of timing constraints can result in degradation of audio and video quality but the system will still continue to operate. [14]

2.4.3 Scheduling

When a computer is multiprogrammed, it frequently has multiple processes that are competing for the CPU at the same time. If only one CPU is available in a system, a choice has to be made regarding which process is to be run next. The part of the operating system that makes the choice about which process is to be run next is called the scheduler. Furthermore, the algorithm that the scheduler uses is called the scheduling algorithm. Information regarding general scheduling procedures and scheduling procedures in two specific types of operating systems can be seen in the following sections.

2.4.3.1 General Scheduling Approaches

A key issue related to scheduling is when to make scheduling decisions. There are various instances when a schedule decision needs to be made. For instance, a scheduling decision needs to be made when a new process is created, when a process exits, when a process is blocked and when an I/O interrupt occurs. Moreover, scheduling algorithms need to make a decision regarding how they deal with clock interrupts. Generally scheduling algorithms can be divided into two categories regarding how they deal with clock interrupts. A non-preemptive scheduling algorithm picks a process to run and then lets it run until it blocks or until it voluntarily gives up the CPU. To the contrary, a preemptive scheduling algorithm picks a process and lets it run for a maximum of some fixed time. If the process is still running at the end of the time interval, the process is suspended and the scheduler picks another available process to execute. [29]

Now, with an understanding of when scheduling decisions need to be made, scheduling algorithms can now be discussed. There a wide variety of scheduling algorithms. The most common scheduling algorithms include round-robin scheduling, first-come first-served and priority scheduling. In the following sections, the scheduling algorithms that are utilized in real-time operating systems and Windows XP will be discussed.

2.4.3.2 Scheduling in Real-Time Systems

Hard real-time systems are characterized by having deadlines that must be met while soft real-time operating systems are characterized by having deadlines that should be met. Moreover, process scheduling in real-time systems must be highly predictable and regular. In real-time systems, priority scheduling is the type of scheduling algorithm that is implemented.

The basic idea of priority scheduling is that each process is assigned a priority. After a process is assigned a priority, the process with the highest priority will be allowed to execute. There are variations regarding how this priority scheduling algorithm is implemented. Most real-time operating systems utilize a non-preemptive priority scheduling algorithm. In this scheduling algorithm, the highest priority process continually runs until it blocks or voluntarily releases the CPU. Then, of the available processes, the process with the highest priority executes when the CPU becomes available. In the case when two processes have the same priority, one process is randomly scheduled to execute and the other process has to wait for the other process to finish execution before it can run. [29]

2.4.3.3 Scheduling in Windows XP

The Windows operating system is currently the most widely used general purpose operating system. The Windows operating system does not have the same operational criteria as real-time operating systems. As a general operating system, Windows is not predictable but instead attempts to allow for each available process to have a fair share of CPU time.

The latest version of Microsoft’s operating system is Windows XP. Windows XP uses a preemptive thread based priority scheduling algorithm. While a process is traditionally perceived to only have one thread of execution, Windows allows for multiple threads to exist in the same process. Consequently, Windows XP has 32 priority levels, with 0 being the lowest and 31 being the highest. The highest 16 levels (15-31) are characterized as real-time levels. Although threads at these levels are characterized as real-time, the name is misleading. Threads running on these priority levels are not guaranteed to receive process time. On the contrary, threads running on the highest priority may receive no processors cycles because the processor may be busy handling interrupts. Moreover, system level threads are not running on these levels. Consequently, setting an application to a real-time level may block a system task and cause for system instability. [32]

During execution, the difference between real-time levels and lower levels (called dynamic levels) is that the scheduler will never change the priority of a thread running on a real-time level. Processes running on lower levels will occasionally have their priorities boosted by the operating system to avoid process starvation. Therefore, before a thread is run on Windows XP, it must first wait for hardware and software interrupts to finish. After the interrupts finish, the waiting thread then must wait for other threads with higher priorities to execute. Then, when a thread is finally allowed to execute, whenever an interrupt arrives or a higher priority thread becomes ready, the running process will become preempted and have to wait for the other interrupts or threads to execute. Consequently, the current kernel architecture for Windows XP is not ideal for high precision real-time applications.

2.5 Existing System

From the previous chapter it is apparent that a PC-based system, which incorporates some form of operating system, can potentially be used to acquire and analyze biomedical signals. Currently there is a PC-based system that is used by the Center for Sensory and Physiologic Signal Processing [C(SP)^2] that can acquire and analyze EMG signals. This system can acquire EMG signals and process them in real-time. Furthermore, this workstation can display selected signals on the computer screen, provide feedback information to a user and store signals to the hard drive. A description of the existing workstation’s hardware and software components follows.

2.5.1 Hardware Overview

Figure 2.7 provides a block diagram of the hardware used in the EMG workstation. EMG signals can be observed if a differential electrode pair is applied to the skin surface above a muscle. In order to analyze and manipulate EMG signals more effectively, the voltages acquired by the electrodes must be amplified. This amplification occurs by connecting electrode-amplifiers to each electrode. In the currently available system, up to sixteen EMG electrode-amplifiers can be used. While these electrode amplifiers can be commercially bought, electrode amplifiers designed by WPI students have also proven to be effective. These amplifiers are made from an Analog Devices AD620BR Instrumentation Amplifier. [26]

These electrode-amplifiers are connected to a signal conditioning box where each EMG signal is first placed through an eighth-order Sallen-Key, high-pass filter with an embedded gain of 10 and a cutoff frequency of 15 Hz. This filter is used to eliminate motion artifacts and offset potentials and allows for more accurate signal measurements. After being high-pass filtered, the signals are then amplified, with separate gain settings available for each channel. In the current system, there are selectable gains of 1, 2, 4, 8, 16, 32, 64 and 128. Each signal is then electrically isolated for patient safety. Electrical isolation is achieved by using an Agilent HCNR201 Photo coupler in conjunction with operational amplifiers. Next, each signal is low-pass filtered with a fourth-order, Sallen-Key, low-pass filter with a cutoff frequency of 1800 Hz. The signals are then available to be acquired by an A/D converter on a PC. In this system a 16-bit Measurement Computing PCI-DAS6402/16 A/D converter card with 32 differential input channels was used.

Consequently, the hardware used in this existing EMG workstation does not require the formation of customized circuits that perform algorithms in hardware. Rather, the system is PC-based and all of the computing hardware is readily available for purchase. As a result, all of the customizable algorithms can be developed in software. Now that the hardware components of the EMG workstation have been described, this customized software can be discussed.

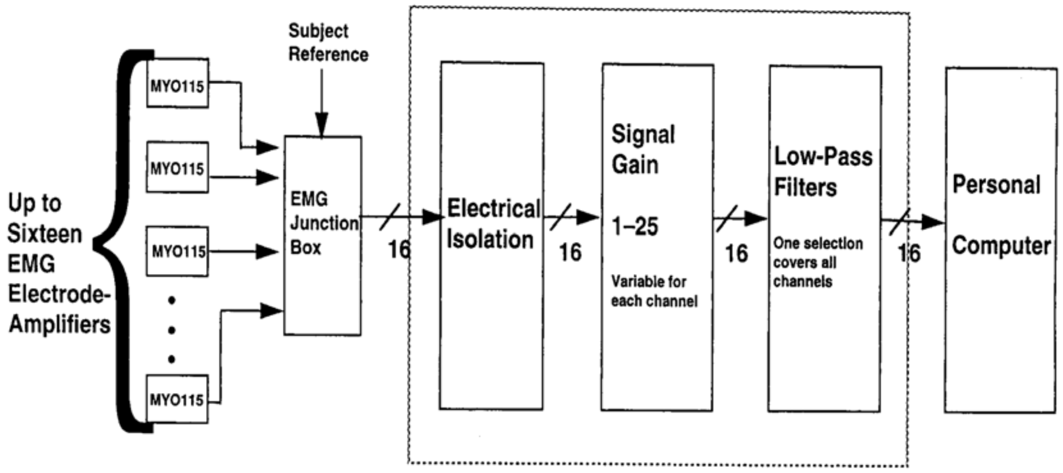


Figure 2.7 – Block Diagram of EMG Workstation Hardware[7]

2.5.2 Software Overview

The design of the current EMG workstation allows the user to design custom EMG algorithms via software. The software developed for the workstation was programmed in Borland C version 4.5 and all software is run under DOS. It should also be noted that all interaction with the A/D board was programmed in C using a software function library provided by the board manufacturer.

The EMG workstation software program works as a configurable state machine. Generic signal processing and EMG analysis modules can be connected in any desired sequence to create a customizable signal processor. For instance, there is a module called “SumDif” that performs simple addition and subtraction and another module named “MAVadp” that implements an advanced EMG amplitude estimation algorithm with an adaptive smoothing window length. Each of these modules, and modules to be made in the future, are designed so that it contains input and output buffers. Consequently, any desired sequence of modules can be made by connecting the output of one processing module to the input of another processing module. This customized processing configuration is developed by entering specific command-line arguments or by having the program read a configuration file when the workstation program is initialized. [7]

A flowchart of the major activities that the software performs is depicted in Figure 2.8. Once the program is initiated, it enters display mode. Display mode functions as a continuous loop that can only be interrupted if the user presses a particular button on the keyboard or if the capturing of a section of data has been completed. In display mode the computer first acquires a scan of A/D samples. Then, if desired, the acquired data can be down sampled. Next, the data processing specified during the initial program configuration is performed and the display screen is continuously updated showing the most recent data. Then outputs are written to any appropriate devices, such as D/A converters, and data can even be saved to the RAM disk. Once this loop is ended, the program then enters edit mode. In edit mode, the user is presented with a menu of command options. These commands include numerous options, such as exiting the program and redisplaying recently captured data. [7]

Hence, this software system allows for the user to create customizable EMG signal processors. Moreover, the basic architecture of this system can be used to analyze other types of biomedical signals as well. In systems where algorithms are executed via hardware, adapting a system is very difficult if not impossible. In the currently available system, a user must merely create new software modules, in the same basic form as the other modules, and connect them with other modules until the desired processing sequence is formed. Thus, this PC based system is highly convenient for the user because all customization can be achieved via software. [7]

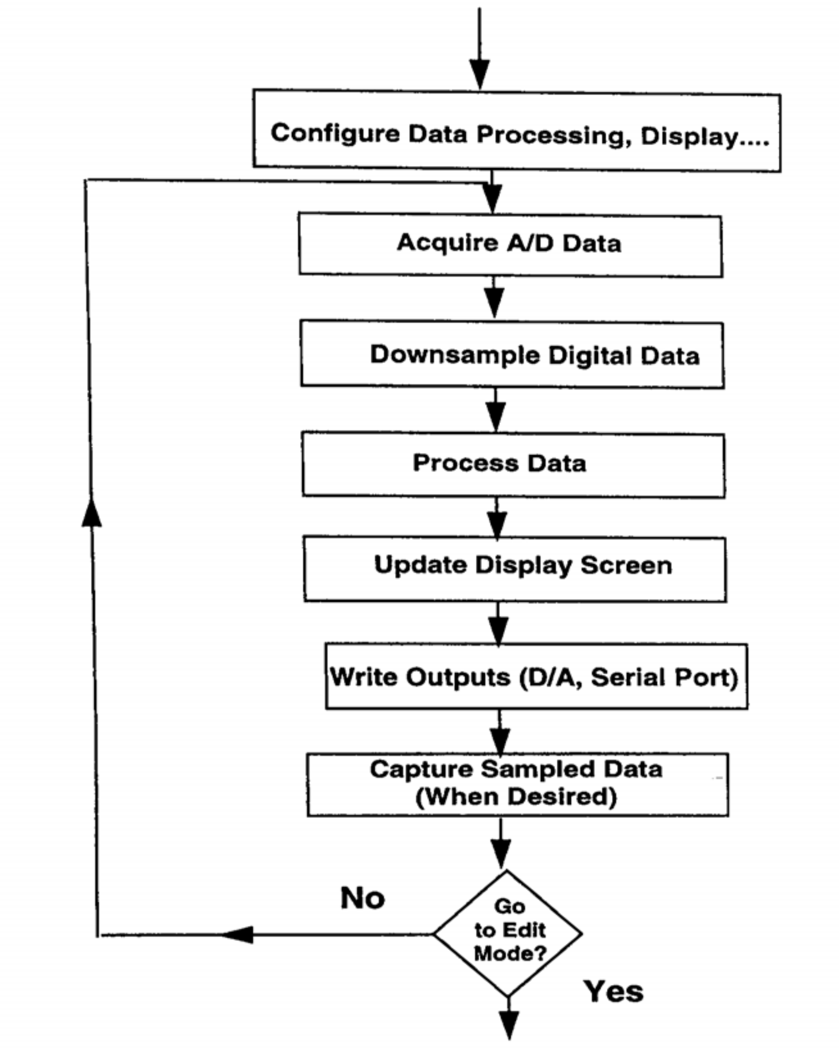

Configure Data Processing, Display....
Acquire A/D Data
Downsample Digital Data
Process Data
Update Display Screen
Write Outputs (D/A, Serial Port)
Capture Sampled Data (When Desired)
Go to Edit Mode?
No
Yes

Figure 2.8 – Software Flowchart for EMG Workstation [7]

2.5.3 Discussion

The previously described system’s design allows for the user to implement EMG signal analysis algorithms via customizable software instead of customizable hardware. Although the current system functions, it has some limitations. Most notably, the software is a DOS-based program. Although DOS can guarantee real-time because it can only run one task at a time, support for DOS is now very difficult to maintain. For instance, DOS lacks modern software and features available in other operating systems and drivers for new hardware components are not readily available in DOS [17]. As a result, in order for this workstation to be functional well into the future, new operating systems and software packages need to be investigated that can potentially be used to upgrade this system. In the following section, one of these options will be discussed.

2.6 Programming in LabVIEW

One programming language that is readily used in soft real-time PC-based signal processing systems is National Instrument’s LabVIEW. LabVIEW is a graphical programming language utilized on the Windows operating system, which can perform many of the same features of other programming languages while allowing for an extensive library of data acquisition board drivers. In this chapter some of the basic and advanced features of LabVIEW that will be utilized in our workstation will be discussed.

2.6.1 Basics

Programs in LabVIEW are designed to appear and function like physical instruments, such as oscilloscopes and multimeters. As a result, LabVIEW programs are called Virtual Instruments (VIs). Building a VI in LabVIEW is a two step process. First, the programmer must create a user interface, known as the front panel. This user interface is comprised of virtual controls and indicators. Controls, such as pushbuttons and dials, provide inputs that can be used in the program. Indicators, such as LEDs and graphical displays, serve as the output devices of the system. While the user is free to create their own controls and indicators, LabVIEW comes with a palette of these front panel objects that the user is able to choose from. [18]

The next step in generating a VI is formulating the block diagram of the instrument. The block diagram is where the VIs are actually defined. The block diagram method of coding is not the typical sequential programming that most other programming languages utilize. Instead, it is a type of object oriented programming, known as dataflow programming. In this type of programming, the user places blocks into their program, which represent controls and indicators on the front panel as variables, or functions that will work on the data supplied by these blocks. The user then uses virtual wires to connect these blocks together. This type of programming is known as dataflow programming, because the wires are used to represent the flow of data in the program. Program execution consequently occurs as data are available on the wires; when a block has new data available at its inputs, it will process these data, and produce an output that can be passed to another function object or stored in a front panel variable. [18]

Similar to the palette of front panel objects, there is a palette that contains functions that can be used in LabVIEW block diagrams. These functions range from standard mathematical processing (such as adding and subtracting), to array manipulation to signal processing. LabVIEW even provides some advanced functions for generating reports and saving data to a given location on a disk. Moreover, this palette of block diagram contains execution control and branch constructs that can be used in a given application. For instance, the user is able to use while and for loops as well as typical case structures that can be found in other text based programming languages.

Figure 2.9 displays both the front panel and block diagram of an example VI. This example VI, known as AddandSubtract, allows the user to enter two numbers in the front panel. The sum and difference of these two numbers are then displayed as output on the front panel. The block diagram contains the functions that actually implement the addition and subtraction of the two numbers entered on the front panel.

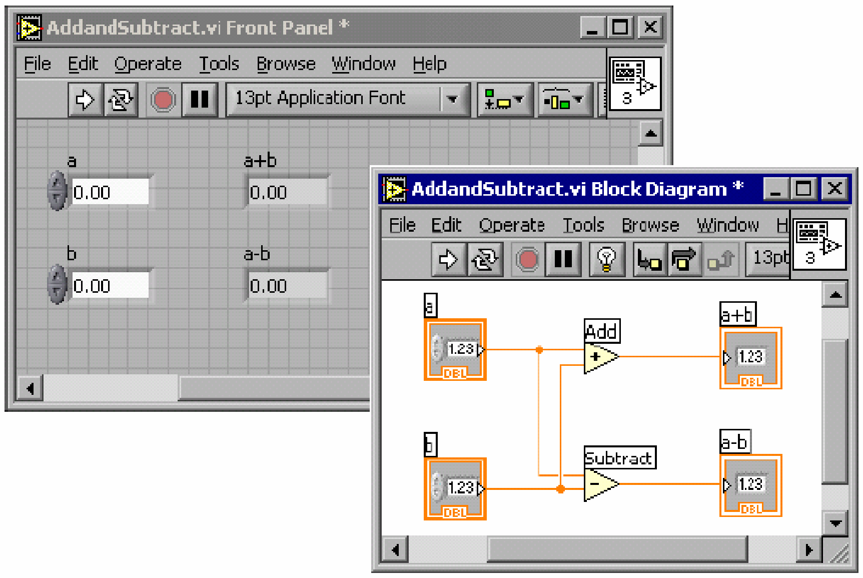

AddandSubtract.vi Front Panel *
File Edit Operate Tools Browse Window Help
13pt Application Font
a
a+b
b
a-b
0.00
AddandSubtract.vi Block Diagram *
Add
Subtract
1.23
DBL

Figure 2.9 – Example Front Panel and Corresponding Block Diagram [21]

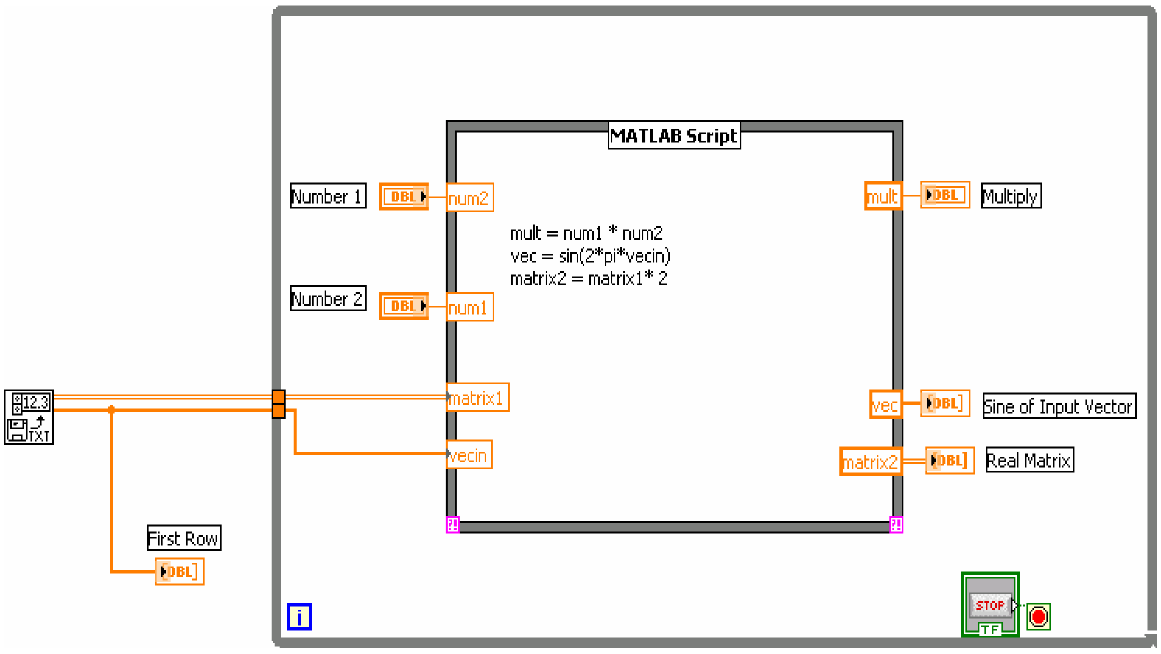
2.6.2 Calling Code from Other Languages

One of the distinctive features of LabVIEW is that LabVIEW has the ability to link with programs written in text languages and in other software packages. As a result, functions that have been created on other development platforms can be used in LabVIEW. Similarly, if it is easier to write a function that is needed in a LabVIEW VI in another software application or even in a text based language, LabVIEW allows you to write the function in another language and use it in LabVIEW. In particular, our workstation will use LabVIEW to interact with both C code and MATLAB. The following sections will give an overview of how LabVIEW can be linked with C and MATLAB.

2.6.3 Linking MATLAB with LabVIEW

To use MATLAB in LabVIEW, one must create a MATLAB script node inside the block diagram of a VI. This node, which can be found in the standard LabVIEW functions library, first allows the user to define input and output variables. The user is then able to connect and wire other functions, controls and indicators to these inputs and outputs like any other standard function, control or indicator found in LabVIEW. After inputs and outputs have been defined, the user can write MATLAB code inside the script node using the inputs and outputs that were previously defined in LabVIEW. When the VI executes, the MATLAB routine entered in the script node will be run and executed in MATLAB. [31]

Figure 2.10 displays an example of how LabVIEW and MATLAB can be linked together. In this particular configuration the user is able to enter two numbers on the front panel. MATLAB will then compute the product of these two numbers and LabVIEW will display the corresponding result. The example also shows that MATLAB can manipulate vectors and matrices that it receives from LabVIEW. In this example, a spreadsheet file is read by LabVIEW and stored as a matrix. The first row of this matrix is connected to the vecin terminal. MATLAB will then compute the sine of this vector and the answer will be displayed in LabVIEW. Similarly, the entire matrix is manipulated as well. Each element inside the matrix in this example is multiplied by a factor of two and then displayed to the front panel. Consequently, LabVIEW allows the user to utilize all of the standard functions in MATLAB, along with custom M-files, in a VI.



```
mult = num1 * num2
vec = sin(2*pi*vecin)
matrix2 = matrix1* 2
```

Figure 2.10 – Example of Linking LabVIEW with MATLAB

2.6.4 Linking C-code with LabVIEW

Another very useful feature of LabVIEW is its ability to link with C code. Although it is convenient to program in the graphical format of LabVIEW, or higher level text formats of MATLAB, there are some programming tasks which are simply left better to C. One example is hardware interfacing. By allowing the use of C code, any piece of hardware that can be interfaced with in C, can also be interfaced with in LabVIEW.

Linking LabVIEW with C code is similar to linking with MATLAB. First, the user must place a Code Interface Node (CIN) on the block diagram and set up its input and output terminals. Next, the user must wire the inputs and outputs of the CIN as if it were a normal function. Figure 2.11 displays an example of a CIN that is connected to two 32-bit signed integers. After everything has been connected in LabVIEW, the C code (.c file) can now be created. By right-clicking on the CIN, a .c file template will be opened. This template provides the structure in which the C code must be written to compile correctly. After the .c file is created, the user must compile the CIN source code and load the CIN object code. [31]

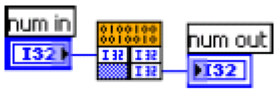


Figure 2.11 – Example of CIN Configuration [31]

Appendix A offers detailed instructions on how to compile and load the source and object code using Microsoft Visual Studio .NET. Once the C code has been successfully loaded into the CIN, the CIN will act as a standard function in LabVIEW that will perform the specified function outlined in the .c file.

2.6.5 LabVIEW Analysis Methods

Another unique feature of LabVIEW is that there are two distinct types of analysis methods available. The standard analysis method used in LabVIEW is known as array-based analysis. In array-based analysis all computations are array oriented. Consequently, the first step in using this type of analysis is to define a buffer size. Once a buffer unit of data is available, this buffer will be able to be analyzed by LabVIEW functions and routines. After a buffer unit of data has been analyzed, LabVIEW VIs then wait for another buffer unit of data to become available. Analysis is only performed again when another full buffer unit of data becomes available [20]. An obvious problem with this method of analysis is that long delays can occur during simple processing routines.

For example, suppose that 100 samples of a signal sampled at 1000 Hz are captured and then passed through two cascading array-based FIR filters. The delay induced from signal passing alone for this example would be:

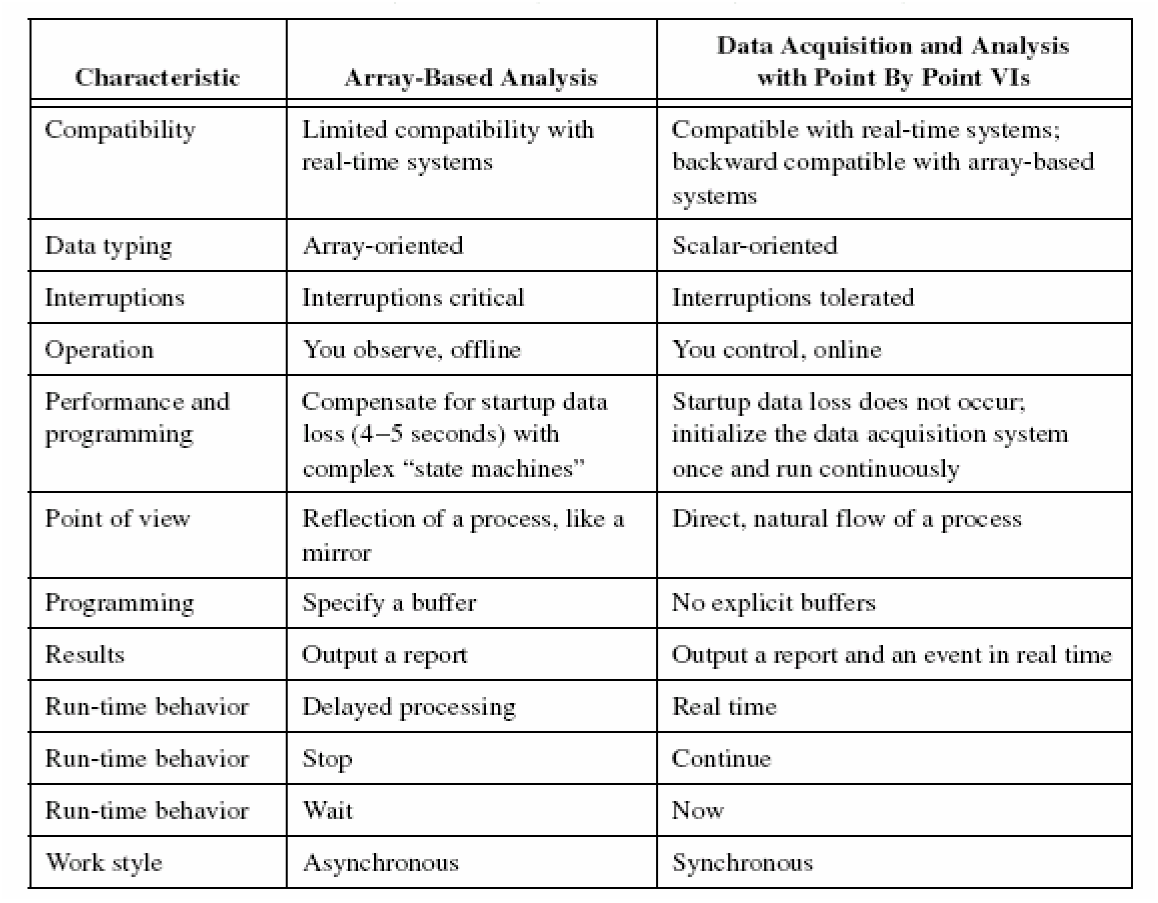
3 * (100 samples / 1000 Hz) = 0.3 seconds

Consequently, for time-critical tasks, such a large delay without even performing any analysis can be unacceptable. However, it should be noted that if a smaller buffer size is utilized, such as a size of one, this large signal passing latency can be decreased.

However, while there are only two analysis methods available in LabVIEW, array-based analysis can be performed two different ways. The first implementation of array-based analysis uses standard LabVIEW functions that take in a standard array of data input. The second implementation of array-based analysis uses VIs that are referred to as Express VIs. Express VIs take time-stamped waveforms as input parameters instead of arrays of numeric data types. A time-stamped waveform is an array of data that has each piece of datum stamped with an associated time value from the clock of the computer. Moreover, an Express VI, unlike standard LabVIEW functions, is configured by a graphical interface. Once an Express VI is placed on the block diagram, a window opens that prompts the user to configure the virtual instrument. For instance, when a Filter Express VI is placed on the block diagram a configuration window opens asking the user to specify whether an FIR or IIR is desired along with other filter configuration parameters.

The chief benefit of using Express VIs over standard array-based functions is that Express VIs are simple to configure and provide numerous different function capabilities in a single module. For instance, all of the standard array-based FIR and IIR functions are available in a single Filter Express VI. However, despite these benefits, Express VIs do have some glaring limitations. In particular, Express VIs do not allow for the level of parameter control that standard array based functions allow. For instance, using a standard FIR array-based filter the user is able to model a filter by providing an array of filter coefficients. However, when using Express VIs, the user is not able to specify filter coefficients and is limited to selecting cutoff frequency values. Similarly, Express VIs do not allow for as complex calculations as regular array-based VIs. For example, when choosing the taps for an FIR filter for a standard array-based filter, the user is able to select any integer number for the size of the taps. However, with an Express VI, the user is only able to select a taps value up to 511. Consequently, while Express VIs offer a simpler and more convenient user interface than standard array-based VIs, Express VIs have limited functionality.

The second type of analysis that is available in LabVIEW is point-by-point analysis. Point-by-point analysis is a scalar oriented process that analyzes each datum point as soon as it becomes available. The buffering associated with array-based analysis can potentially cause large delays, thus making array-based analysis the non-ideal method for low latency applications. However, point-by-point analysis is designed specifically for low latency applications [3]. Table 2.1 outlines further the differences between point-by-point and array-based analysis.



Characteristic	Array-Based Analysis	Data Acquisition and Analysis with Point By Point VIs
Compatibility	Limited compatibility with real-time systems	Compatible with real-time systems; backward compatible with array-based systems
Data typing	Array-oriented	Scalar-oriented
Interruptions	Interruptions critical	Interruptions tolerated
Operation	You observe, offline	You control, online
Performance and programming	Compensate for startup data loss (4–5 seconds) with complex "state machines"	Startup data loss does not occur; initialize the data acquisition system once and run continuously
Point of view	Reflection of a process, like a mirror	Direct, natural flow of a process
Programming	Specify a buffer	No explicit buffers
Results	Output a report	Output a report and an event in real time
Run-time behavior	Delayed processing	Real time
Run-time behavior	Stop	Continue
Run-time behavior	Wait	Now
Work style	Asynchronous	Synchronous

Table 2.1 Difference Between Array-Based and Point-By-Point Analysis [20]

Choosing which type of analysis method to use in an application simply involves choosing which function library on the functions pallet to use. The standard functions available use array-based analysis. If array-based analysis is desired using Express VIs, one must pick the Express VI which maps to the standard array-based function on the Express VI function palette. Similarly, if point-by-point analysis is desired, one must merely chose from the functions listed in the point-by-point subsection of the functions palette. Most of the functions available to perform array-based analysis have an analogous function available to perform point-by-point analysis. Figure 2.12 displays three types of filter functions that utilize each of the three analysis techniques. While the array-based and point-by-point filter displayed only implement a Butterworth filter, the Filter Express VI implements various types of FIR and IIR filters.

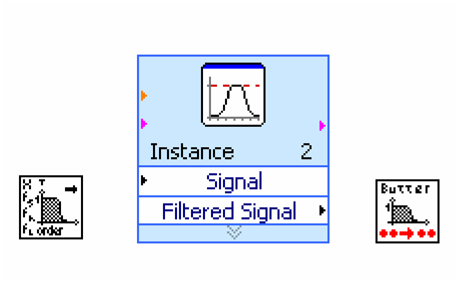


Figure 2.12: Array-based, Express VI and Point-by-Point Butterworth Filters

2.7 Chapter Summary

This chapter has detailed background information that is associated with our project. In particular, information is presented detailing the type of signals that our workstation will be analyzing along with the types of applications that our workstation will be involved in. One design option for creating a biomedical-signal acquisition system is to develop a PC-based system that runs a particular software package. One widely used software package is National Instrument’s LabVIEW. An overview of LabVIEW and some of its advanced features were presented. In the following chapter, our specific project goals and objectives will be presented.

3. Problem Statement

While there are a myriad of ways to implement a signal acquisition, display and feedback system to be used in biomedical experiments, the goal of our project was to determine if a PC-based system running LabVIEW can meet the hard real-time constraints that are necessary for various EMG control applications. This chapter presents the problem statement of our project, along with the overall goal of our project team. Further, objectives that needed be met to accomplish the overall goal are presented.

3.1 Problem Statement and Project Goals

Biomedical signals, such as the EMG, can be used in various applications. Consequently, our system will be used to acquire and analyze EMG signals. Moreover, multiple EMG signals are typically recorded at a time during experiments. Hence, our system will be configured to acquire and analyze as many EMG signals as possible at a time. As discussed in the background section of the report, the bulk of EMG signal information is at frequencies below 100 Hz. As a result, a sampling frequency of 256 Hz, for each analog channel utilized on the DAQ board, was chosen to be utilized in experiments conducted with this workstation.

Moreover, our system will be configured to be able to conduct control applications, such as those involved with prosthetic limb development and gait therapy. In order to accurately control prosthetic limbs, the limb should respond within a specified interval to the stimulus. Similarly, if feedback is needed regarding the muscle activation of an individual participating in rehabilitation gait therapy by a certain amount of time, our system should be able to meet this deadline for providing feedback. Therefore, our workstation must process signals in hard real-time. By definition, a hard real-time system is a system whose correctness of computations not only depends on the logical correctness of the computation but also upon the time in which the results are produced. Hence, if the specified timing constrains are not met, system failure is said to occur [14].

As a result of the applications in which our workstation will be used for, it is desired that our system function as a hard real-time system. Besides acquiring and synthesizing signals, our system should be able to display results, give appropriate feedback to the user and, when prompted, save selected signals to the hard drive. Also, since our system will be used for a variety of biomedical experiments, it needs to be easy to use and operate. Similarly, the system must also be easy to modify since it will be continually modified and updated when different biomedical experiments are formulated.

Now that we know what the system must do, the question we ask in this project is: “Can a hard real-time biomedical signal acquisition, display and feedback system be implemented on a commodity Windows-based PC, using National Instruments LabVIEW software? If so, how functional can we make this system for the end user?” In particular, a PC with 1.68 GHz Intel Pentium 4 CPU and 512 MB of RAM running Windows XP Profession Version 2002 Service Pack 2 is the computer that is being utilized to answer the previously stated question. In the following section, specifics regarding the hard real-time requirements of our system will be discussed.

3.2 Hard Real-Time Requirements

As stated previously, the goal of our project is to determine if we can develop a PC-based system to conduct biomedical experiments in hard real-time. The experiments that will be conducted with this workstation involve having graphical data displayed on a computer monitor. As a result, careful attention must be made to the screen refresh rate of the monitor. The screen refresh rate refers to how fast the images displayed on a computer monitor are updated. Most computer monitors operate with a screen refresh rate of 60 Hz. However, if the time it takes to output data onto the monitor exceeds approximately 17 milliseconds, the screen refresh rate for the monitor will scale down to 30 Hz, or one update every 30.33 milliseconds. [28]

Most humans are able to visibly notice slowdown in data being displayed on a monitor when the screen refresh rate approaches 30 Hz. As a result, convention holds that 60 Hz is an acceptable screen refresh rate for real-time systems while a 30 Hz refresh rate is deemed highly unacceptable [28]. Consequently, the goal of our system is to determine if LabVIEW can be configured to conduct routines that operate with a maximum worst case latency of 5 milliseconds. Operation at or below this worst case latency will allow for the screen to be updated at the conventional 60 Hz and thus allow for our system to maintain real-time operation.

3.3 Objectives and Tasks

The previous two sections identify the major goals of our project. In order to achieve this goal, numerous other objectives must be met. These objectives can be seen below:

1. Determine the effect using different data acquisition cards has on system performance,
2. Develop a method that indicates how fast LabVIEW functions operate,
3. Develop a method that determines the latency associated with acquiring data via the DAQ board utilizing LabVIEW tasks and indicate if processing can keep up,
4. Develop EMG amplitude estimation VIs,
5. Determine if Windows XP Priority setting affects system performance.

The following chapter in the report will describe how we met these objectives and the results that we obtained.

3.4 Summary

This chapter has described, broadly, the goal of our project as determining if a PC-based system executing LabVIEW can be used to conduct biomedical experiments that require hard real-time execution. In particular, our goal will be to determine if the system can operate with a maximum latency of 10 milliseconds. The objectives and tasks necessary in achieving this goal are also described.

4. Data Acquisition Board Performance Analysis

In section 2.3.2 the concept of using a PC-based system to conduct biomedical experiments was introduced. In particular, the notion that a DAQ board needs to be installed in a PC in order to allow for analog signals to be acquired and then analyzed in software by the PC was discussed. In chapter three the main goal of our project and the various objectives that we met to achieve these goal was presented. One of the questions that was answered in order to accurately determine if a hard real-time biomedical signal acquisition, display and feedback system can be implemented on a commodity Windows-based PC, using National Instrument's LabVIEW software, was if the DAQ board used with the PC affected system performance. In order to meet this objective a comparison of two commercial DAQ boards was performed. The two boards that were analyzed were Measurement Computing's PCI-DAS6402/16 and National Instruments' PCI-6229. In the following sections a description of these two boards and a comparison of the performance of both boards will be presented.

4.1 Measurement Computing PCI-DAS6402/16

The first DAQ board that was analyzed to determine if the choice of DAQ board affects the ability of a PC-based system to operate in hard real-time was the PCI-DAS6402/16 developed by Measurement Computing. In section 4.1.1 a review of the hardware of the board will be given. Then, in section 4.1.2, the LabVIEW Universal Library function set, which is used to configure Measurement Computing DAQ boards, will be described.

4.1.1 Hardware Overview

The PCI-DAS6402/16 has a total of 32 differential analog input channels with a 16 bit A/D resolution and a 200 kHz maximum sampling rate. However, while there are multiple analog input channels the PCI-DAS6402/16 only contains one A/D converter. Consequently, when more then one channel is being utilized to acquire data, the sampling rate for each channel is defined as: **(Sampling Frequency)/(Number of Channels Used)**. Hence, when all 32 differential analog input channels are used, the maximum sampling rate for each channel is 6.25 kHz.

Acquired data are stored in a 2048 sample First-In First-Out (FIFO) buffer located on the DAQ board. Data are then transferred, using the PCI bus of the computer, via direct memory access (DMA) in either demand or non-demand mode. Data are transferred from the FIFO to the computer once the amount of samples stored in the FIFO reaches a specified value, which is known as the count value. The minimum count value for this particular Measurement Computing board is half of the size of the FIFO, which is 1024 data samples. [30]

The PCI-DAS6402/16 also has additional functionality besides acquiring analog signals. This DAQ board contains two 16-bit analog output channels. Each of these analog output channels is able to simultaneously output data at a maximum rate of 100 kHz per channel. Analog output is also stored in the 2048 sample FIFO and, like analog input, data are transferred out of the FIFO buffer once a specified count value is reached. Again, this count value must be at least half of the size of the FIFO. Although digital input and output signals will not be used during the course of this project, it should be noted that the PCI-DAS6402/16 four digital input and output channels. This board utilizes a single DMA controller to control all of the various functions of this board. Further information regarding the hardware of the PCI-DAS6402/16 can be found in [30].

4.1.2 LabVIEW Universal Library Functions

The PCI-DAS6402/16 is completely plug-and-play. Consequently, there are no switches or jumpers to set on the board and, instead, the hardware is configured using software. Hence parameters such as the number of analog input and output channels used and the sampling frequency and count value are set in software. Other parameters that must be set in software include whether the board will be operated in background or foreground mode. In foreground mode, the PC is not allowed to do any form of processing while the DAQ is acquiring data. In background mode, the computer is still able to function while data are being collected. The user also must select whether the board will operate in continuous mode or in default mode. In default mode the DAQ board will stop acquiring data after one scan. In continuous mode, the board will continue to acquire data in an endless loop until the acquisition program is halted. Lastly, it must be selected in software if the board will operate in burst mode. When burst mode is enabled the sample data is acquired on adjacent channels at the maximum sample rate. This mode is used to minimize time between samples, or channel skew, on different channels. [30]

Measurement Computing has provided a library of software functions in C that can be used to configure the previously described settings of the board. This function library is referred to as the Universal Library. The Universal Library can in turn be utilized in LabVIEW by using the CIN. Figure 4.1 depicts a LabVIEW VI that was created that will acquire data from the data acquisition board in background continuous mode. The primary function used is AlnScBg. This VI scans a range of A/D channels in the background and then stores the samples into an array. In order to manipulate data as it is being collected, another VI needs to be used called GetStatus. This virtual instrument takes, as input, a cluster data structure, referred to as context, from AlnScBg. GetStatus will check the status of a background operation that is currently running and will output a Boolean, called running, which indicates the status of the background operation. GetStatus also outputs the data array containing the inputted data that were acquired via the AlnScBg VI.

While GetStatus outputs the array containing the acquired data, this data is represented as an array of A/D count values. In some instances, in order for more meaningful results that can be analyzed, these A/D count values should be converted to equivalent voltage values. The Universal Library VI ToEng accomplishes this conversion by taking the data array outputted from GetStatus and outputting a multidimensional array containing the appropriate voltage values. Each row in this multidimensional array contains data collected from each analog input channel.

Lastly, in order to stop any background operation that is in progress the VI StopBg must be used. This VI simply takes the context data structure as input that the GetStatus VI outputs. The StopBg function will then output an error code that is passed as input to the ErrMsg VI. This ErrMsg VI will then translate the error message into a readable string.

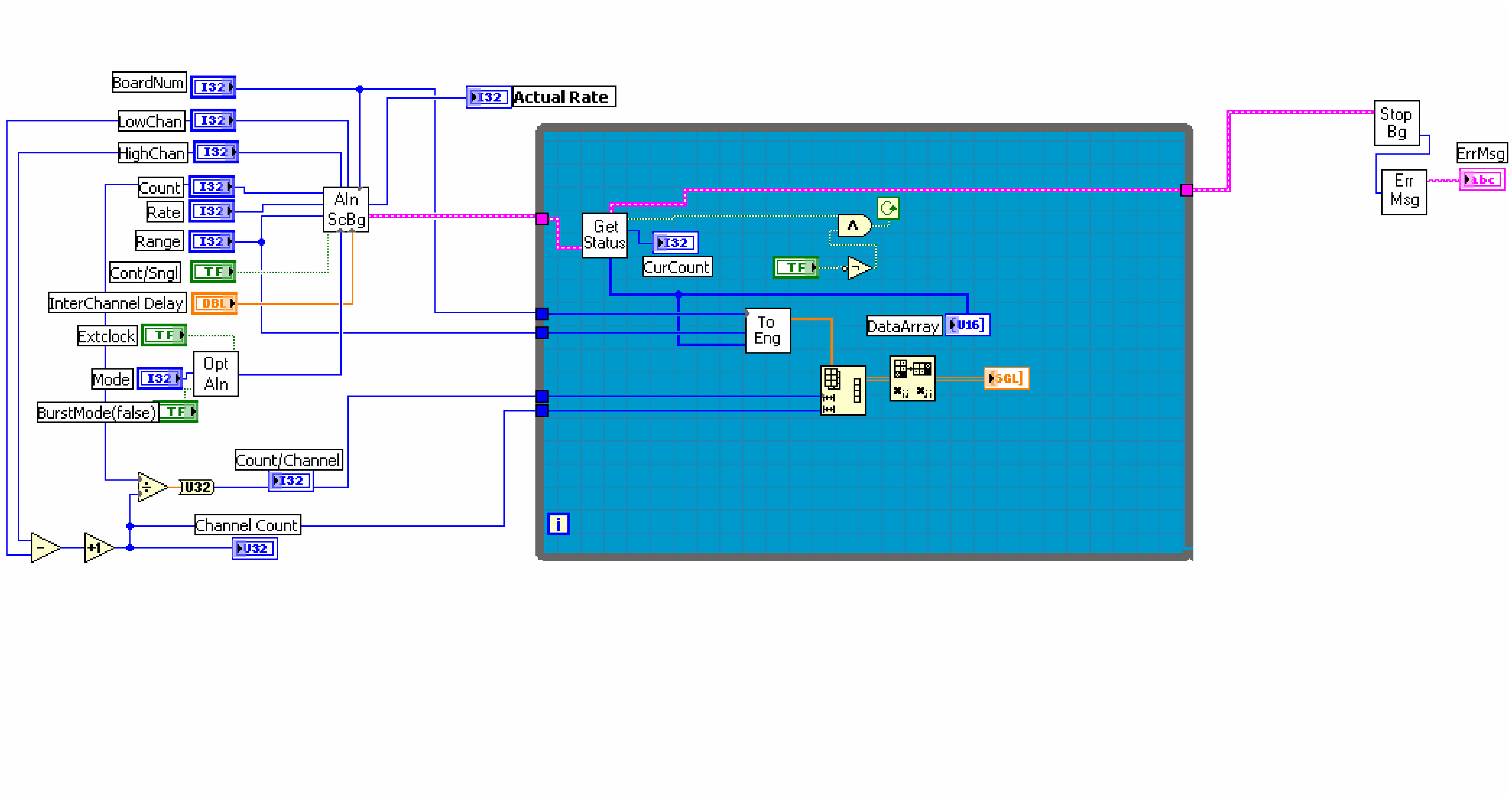


Figure 4.1: VI That Acquires Data in Background Continuous Mode

The previously described module successfully acquires data from the DAQ board. However, this method acquires data as an array and, thus, these acquired data can only be used with LabVIEW’s array based analysis. As a result, a slight modification needs to be made to the previously described model in order to make the data collected compatible with point-by-point VIs. In order to extract one piece of datum at a time, the multidimensional array of voltage values is placed through two nested for loops. By placing the multidimensional array through two for loops, a single point of datum from each of the analog input channels will be extracted during each loop iteration. Figure 4.2 displays this modified module that can be used to perform point-by-point analysis.

Using the analog output channels requires the same basic procedure as depicted in Figure 4.1. However, the functions AOutScBg and FromEng are used instead of AInScBg and ToEng. FromEng takes voltage values and converts them to A/D count values. Then, the function AOutScBg outputs these created A/D count values to specified output channels at a specified sampling frequency.

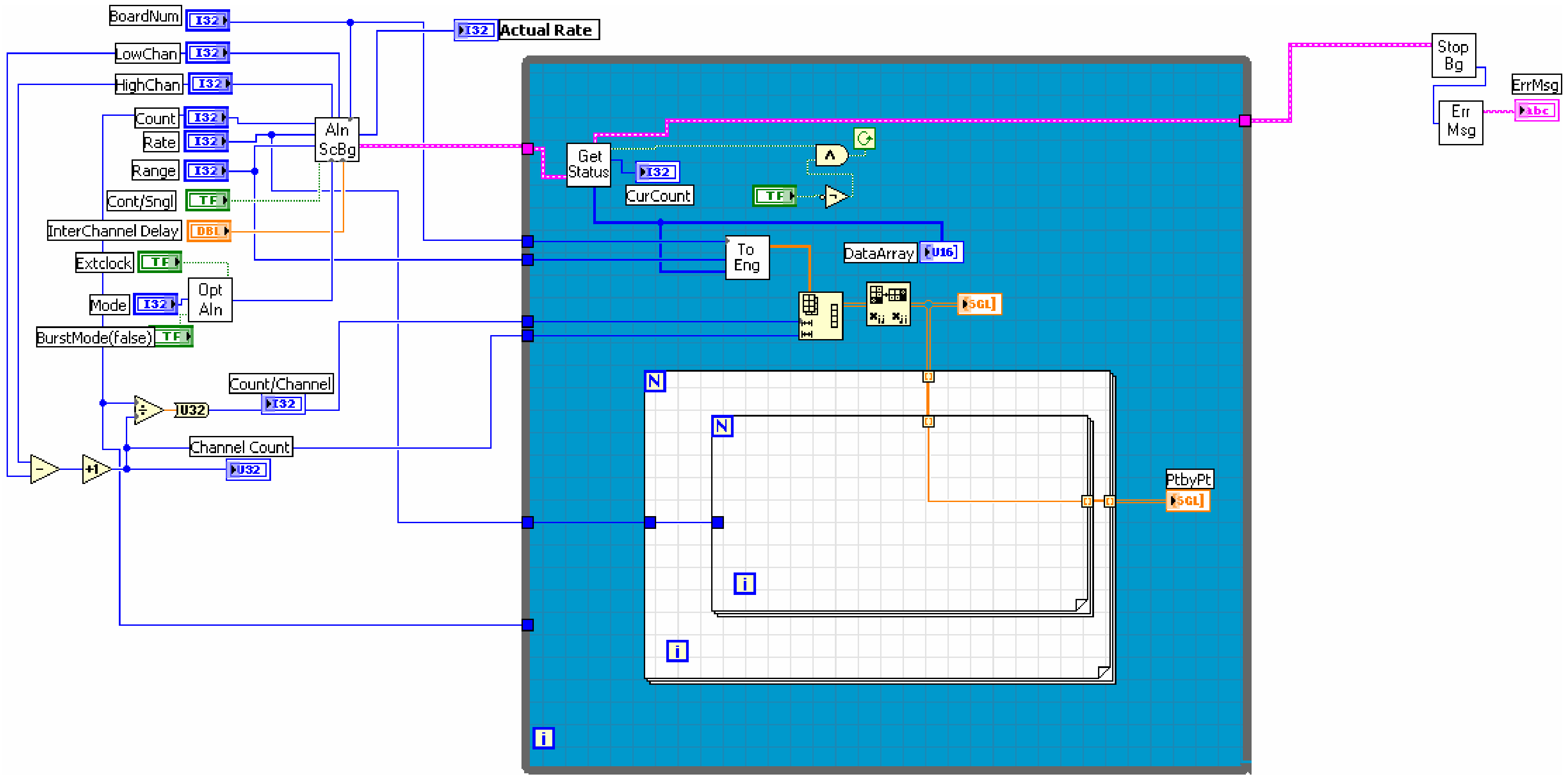


Figure 4.2: VI That Acquires Data in Background Continuous Mode for Use with Point-By-Point Functions

4.2 National Instrument’s PCI-6229

The other commercial DAQ board that was analyzed to determine if the choice of DAQ board would affect system performance was the PCI-6229 M-series device manufactured by National Instrument’s. In section 4.2.1 a review of the hardware of the board will be given. Then, in section 4.2.2, National Instruments’ DAQmx LabVIEW function set, which is used to configure NI M-series DAQ boards, will be described.

4.2.1 Hardware Overview

The PCI-6229 has 16 differential analog input channels with a resolution of 16 bits and a maximum sampling frequency of 250 KHz. Again, like the Measurement Computing device, the PCI-6229 only has one A/D converter. Consequently, when more then one channel is being utilized to acquire data, the sampling rate for each channel is again defined as: **(Sampling Frequency)/(Number of Channels Used)**.

There are two methods available for transferring analog input data from the PCI-6229 and the computer. First, like the Measurement Computing board, input data can be stored in a 4095 sample First-In First-Out (FIFO) buffer located on the DAQ board. Data is then transferred, using the PCI bus of the computer, via direct memory access (DMA). Data is transferred from the FIFO to the computer once the amount of samples stored in the FIFO reaches a specified value, which is referred to as the buffer size. The minimum buffer size for this NI board is two samples. The other method that is available for transferring analog input data from the PCI-6229 to the PC is referred to as hardware-timed single-point. In this transferring mode configuration, a single sample of datum is transferred using the PCI, via DMA, every sampling period. In this configuration, the FIFO is completely bypassed by utilizing the hardware clock present on the DAQ board. [24]

The PCI-6229 also has additional functionality besides acquiring analog signals. This DAQ board contains four 16-bit analog output channels. When only one of these analog output channels is used, the maximum sampling rate is 833 kHz. When all four analog output channels are used, the maximum update rate is 625 kHz per channel. Again, there are two methods of transferring analog output between the PC and DAQ board. The method of storing the data into a FIFO buffer on the board until a certain buffer count value is obtained is the first option. For analog output there is 8191 sample FIFO buffer that is shared with all of the output channels. The second option for acquiring data is the hardware-timed single-point configuration that was described previously.

Although digital input and output signals will not be used during the course of this project, it should be noted that the PCI-6229 has 48 digital input and output lines, two 32-bit general purpose timers and a phase-locked-loop (PLL). In order to control all of the various functions of the DAQ board, this NI device uses six DMA controllers. Consequently, there is an individual DMA controller dedicated to analog input, analog output, digital input, digital output, the general purpose timers and the PLL.

Also, in order to gain access to all of the pins of the DAQ board, a SCB-68 noise rejecting, shielded I/O connector block must be utilized. This I/O connector block is connected to the DAQ board via the SHC68-68-EPM shielded cable. Further information regarding the hardware specifications of the PCI-DAS6402/16 and these two other peripheral components can be found in [24].

4.2.2 NI-DAQmX LabVIEW Functions

The NI PCI-6229 is completely plug-and-play. Consequently, there are no switches or jumpers to set on the board and, like the Measurement Computing device, the hardware is configured using software. Since LabVIEW is a National Instruments' product, this software package contains a library of functions that is used to configure NI DAQ products. This library of functions can be divided into two sets. The first set of functions can be utilized with older NI E-series devices while the second set of functions can be utilized with newer NI M-series devices. Since the PCI-6229 is an M-series device, the function set applicable to only M-series devices will be discussed. This particular function set is referred to as the NI-DAQmx function library.

There are two main ways to configure the PCI-6229 device using the NI-DAQmx function library. The first choice is to use the DAQ Assistant Express VI. The DAQ Assistant Express VI can be seen in Figure 4.3. The NI-DAQ Assistant provides a graphical interface for configuring the DAQ board.

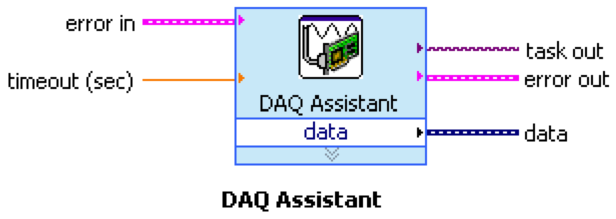
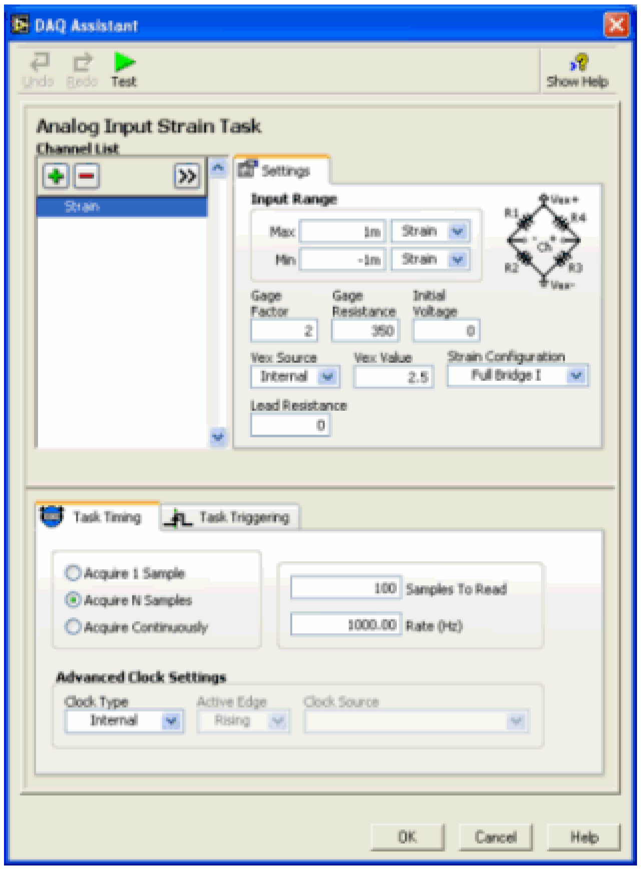


Figure 4.3: NI-DAQ Assistant

By placing the DAQ Assistant on the block diagram and right-clicking it with the mouse, the configuration palette that can be seen in Figure 4.4 will be launched. Using this graphical palette, the user is able to select which analog input channels and output channels will be used, the desired sampling frequency and method for data transfer can be selected.



Analog Input Strain Task

Channel List: Strain

Input Range
Max: 1m Strain
Min: -1m Strain

Gage Factor: 2
Gage Resistance: 350
Initial Voltage: 0
Vex Source: Internal
Vex Value: 2.5
Strain Configuration: Full Bridge I
Lead Resistance: 0

Acquire 1 Sample [✓ Acquire N Samples] Acquire Continuously

100 Samples To Read
1000.00 Rate (Hz)

Advanced Clock Settings
Clock Type: Internal
Active Edge: Rising
Clock Source:

Figure 4.4 NI-DAQ Assistant Configuration Palette

While the DAQ assistant provides a simple graphical user's interface, using this method does not allow the user to configure certain aspects of the DAQ board in regards to analog output. In particular, when the option is chosen to utilize the FIFO when performing an analog output task, the DAQ assistant configures the board to operate in regeneration mode. In regeneration mode, data that is in the DAQ FIFO is continuously reused. For instance, a single period of a sine wave can be written to the FIFO buffer. Regeneration allows for this period to be continually generated, thus giving the appearance of a continuous waveform. While such a regeneration feature might be appropriate in some applications, if it is desired to have the output waveform change while the board is outputting, then this mode is highly undesirable.

Consequently, in order to gain greater control and flexibility, there are individual DAQmx functions that are present. By linking these various functions together, such as with the Universal Library function set, the board can be configured completely. Although using these various functions adds a certain amount of complexity compared to using the single DAQ assistant function, by using these various functions the user is able to have full control of their particular application.

In order to configure an analog input or output task there are five basic functions that must be utilized. The first function is the Create Virtual Channel function. A virtual channel maps in software to a physical channel. Consequently, this function specifies what particular analog input or analog output channels are being utilized in the application. The second basic function is the Sample Clock Timing function. This function is used to specify what type of data transfer method is going to be used in the application. For instance, one can either specify that a continuous acquisition utilizing the FIFO is going to be used and the size of the buffer or that a single-point hardware-timed acquisition will be performed.

Next, the Start Task function is the third function that must be utilized to configure an analog input or output task. The Start Task function takes a task that has been defined and actually executes it. The next two functions that are necessary are the DAQmx Read and DAQmx Write functions. The Read function reads samples from the specified acquisition task while the write function writes samples to the specified task. The final function that is necessary is the Clear Task function. This function will clear a specified task and, consequently, release all system resources.

4.3 DAQ Board Comparison

The previous two sections have given an overview of the functionality of the two DAQ boards that were chosen for comparison. In this section architectural differences of the two boards will be examined. In particular, the manner in which each board signals for data to be transferred between the PC and the board as well as how each board handles simultaneous input and output will be discussed.

4.3.1 Data Transfer

As discussed earlier in this chapter, in order for data that is stored in the FIFO of the Measurement Computing PCI-DAS6402/16 board to be transmitted to the PC for analysis, the board must issue a DMA transfer request. The PCI-DAS6402/16 issues this request when the onboard FIFO is filled with half of a packet of data. The minimum packet size for the PCI-DAS6402/16 is 2048. Hence, a minimum of 1024 data samples must be collected before the board can generate an interrupt and send data to the PC [30].

A possible application for our system is for it to be used during experiments that involve real-time control. As a result, it is desired that our system can acquire, process and analyze data in hard real-time. However, since the board must obtain a minimum of 1024 data samples before issuing a data transfer request, this will cause a latency to incur before any processing is done. Assuming that a sampling rate of 4096 Hz is used with all 32 analog input channels (128 Hz per channel) with the minimum count of 1024 one can find that the latency incurred before doing any processing is:

(1024 /4096)*(1/32) = 7.8 msec

While an initial latency of approximately 8 msec does not seem significant for a soft real-time system, this latency is quite large for a system dependant on real-time control. Therefore, the minimum count requirements imposed by the Measurement Computing hardware makes the DAQ board seem insufficient to handle hard real-time processing. To the contrary, NI's PCI-6229 is able to transfer a single piece of datum every sampling period using single-point hardware-timed. As a result, the National Instruments' board does not contain this initial latency before processing.

4.3.2 Simultaneous Input and Output

As previously discussed, it is desired for the workstation that is being created to be utilized in experiments involving control applications. As a result, it is desired that the DAQ board will be able to perform simultaneous input and output. That is, while data is being collected, it should be processed and then sent to analog output.

The Measurement Computing PCI-DAS6402/16 board uses a single FIFO and DMA controller for both analog input and analog output [30]. Thus when one background operation is using the FIFO for either analog input or output, another background operation cannot be initiated. While this would not be a problem if one wanted to simply input data, and process it at a later time, the presence of a single FIFO and DMA controller for both analog input and output tasks is a large problem when it comes to creating a hard real-time system with analog output and input. As a result, the lack of simultaneous input and output limits the board's uses in areas such as prosthetics and control.

While the PCI-DAS6402/16 is incapable of doing simultaneous input and output, the PCI-6229 contains a separate DMA controller for analog input and analog output. Consequently, the PCI-6229 is architecturally designed to conduct simultaneous input and output routines. However, if simultaneous analog input and output is desired, one will need to perform single-point hardware-timed analog input and output. Figure 4.5 depicts how simultaneous input and output is conducted using the hardware-timed single-point transfer method. The process data module located in this function block diagram is a generic process that is meant to represent any data processing that can take place to the data acquired from the analog input channels before the data is passed as output to one of the analog output channels.

However, in the beginning of this chapter, it was noted that there was another data transfer method available with the PCI-6229. This other data transfer method allowed for the user to specify a buffer size. Once this buffer size is reached, data are then transferred between the PC and DAQ board. However, using this data transfer method is not optimal for simultaneous input and output routines. The buffers that are used to hold analog output before it is transferred to the DAQ board are formulated in software in LabVIEW. As a result, while the analog input is controlled by the clock located on the DAQ device, the analog output is controlled by the operating system and the PC. As a result, the analog input and output routines, although they might have the same sampling-frequency, are not guaranteed to operate and update at the same time since Windows XP is not a real-time operating system and a thread context switch can result in system delay.

The hardware-timed single-sample transfer method is thus desired for simultaneous input and output routines because the same clock located on the DAQ device is utilized. As a result, hardware-timed single-sample transfer is the data transfer that should be utilized if simultaneous input and output is desired. This notion was further cemented by Garrett Earnshaw, an applications engineer for National Instruments, who reiterated that single-point hardware-timed data transfers are the only type of transfer that should be utilized when performing a routine that requires simultaneous input and output. [15]

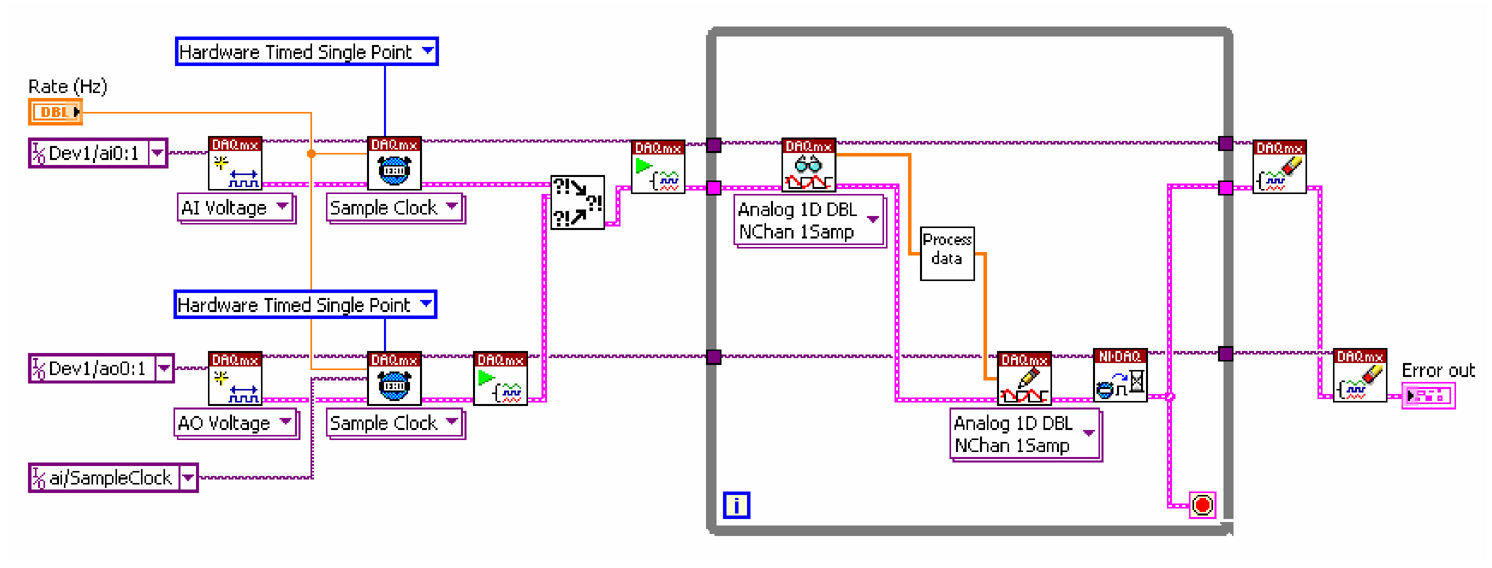


Figure 4.5: Simultaneous Input and Output Using PCI-6229

4.4 Discussion

As discussed in the previous two sections, the PCI-DAS6402/16 has two limitations that hinder the development of our workstation. First, the board places a limitation on the minimum of samples that must be acquired before an interrupt can be generated. This limitation causes a latency that will not allow for our workstation to be used with hard real-time control experiments. Second, the PCI-DAS6402/16 does not support simultaneous input and output. Since the goal of this project is to determine if LabVIEW can be used to implement a workstation to conduct biomedical systems, some of which involve hard real-time control, it is apparent that the PCI-DAS6402/16 is not a suitable DAQ board to create a workstation to conduct such biomedical experiments.

On the contrary, the PCI-6229 manufactured by National Instruments is designed to perform simultaneous input and output. Moreover, the NI device does not place restrictions on the minimum number of points that must be acquired before data is transferred between the board and the PC. Our comparison between these two devices thus shows that all DAQ boards do not follow the same conventions. Hence, the DAQ board chosen to create a PC-based system for conducting biomedical experiments does contribute to system performance. Consequently, we have decided to utilize the PCI-6229 device to perform our latency analysis measurements in order to determine to what extent a workstation can be created to conduct biomedical experiments using LabVIEW and a commodity PC.

4.5 Chapter Summary

In this chapter a comparison between two different DAQ boards was conducted to determine if the choice of DAQ board played an impact on system performance when creating a PC-based system to conduct biomedical experiments. As our comparison displays, all DAQ boards do not follow the same conventions. For instance, the manner in which DAQ boards trigger a request to transfer data between the DAQ board and PC is unique for different boards. Moreover, every DAQ board does not support simultaneous input and output functionality. As a result, the DAQ board chosen plays an impact on system performance. For our application, to create a workstation to conduct biomedical experiments that may involve control experiments, the PCI-6229 DAQ board was chosen.

5. LabVIEW Analysis Methods Comparison

In section 2.6 the notion that LabVIEW has different analysis methods was discussed. In particular, LabVIEW has array-based analysis and point-by-point analysis. Furthermore, array-based analysis can be further subdivided into two function types: standard array-based and Express VI functions. In order to investigate the feasibility of creating a hard real-time workstation, it is important to investigate the latency associated with using each type of LabVIEW VI. Consequently, EMG amplitude estimation algorithms developed using all three of the analysis methods will be presented. These three different EMG amplitude estimation algorithms were tested to determine the latency induced by each method. Once the results of these tests for each method is displayed, observations regarding if any of the analysis methods operate under our maximum latency of five milliseconds can be made.

5.1 EMG Amplitude Estimation VIs

In the background section of the report, the theory behind EMG amplitude estimation techniques was presented. When implementing the previously discussed algorithm, we can represent the amplitude estimation process utilizing adaptive whitening with nine stages. A block diagram of this entire nine stage process can be seen in Figure 5.1.

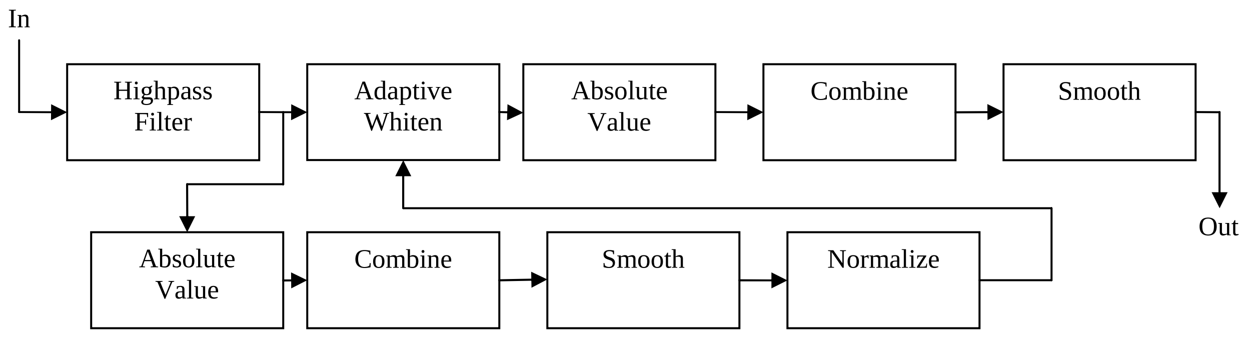


Figure 5.1: EMG Amplitude Estimation Block Diagram Utilizing Adaptive Whitening

While three versions of the above algorithm were created using the three different LabVIEW function types, all three routines were constructed in the same basic manner. Hence, the only significant difference between each implementation of the EMG amplitude algorithm was that point-by-point, Express VI and standard array-based functions were respectively utilized in each implementation of the algorithm.

Consequently, now knowing that the same algorithm was utilized for each routine, a discussion of how each stage of the algorithm was implemented can be discussed. First, the highpass filter stage in all three routines was implemented by utilizing one of the three versions of the FIR highpass filter that is available in LabVIEW. Second, as discussed previously in the background section of the report, a smoothing filter can be implemented as a moving average filter or another FIR lowpass filter. In all three modules, this smoothing stage was chosen to be implemented by utilizing a standard FIR lowpass filter. Third, the absolute value stage was simply constructed by taking the absolute value of a sample. Fourth, the combining stage consists of taking an average of all of the signals that are used in the routine. Fifth, the normalization step was executed by merely dividing each sample by the maximum voluntary contraction (MVC). Last, the adaptive whitening filter was represented as a matrix of pre-processed FIR filters. The filter chosen to analyze a given datum point is determined by the current normalized MVC value.

In all three versions of the EMG amplitude estimation routine, the user is able to enter values for the cutoff frequencies for the highpass filter and the smoothing filter. The user also is allowed to enter the value for the MVC and the amount of filter coefficients that will be used with the adaptive whitening filter. Also, our VI not only outputs the graph of the final signal but it plots the output after each of the nine major steps of the algorithm. For instance, the user can look at the graph of the waveform after it has passed through the highpass filter or even after it has been whitened. Appendix H provides a more detailed view of how each routine was implemented in LabVIEW.

While three EMG amplitude estimation routines have been created, all three routines have some limitations. First, the gain normalization routine, which was discussed in the background section of the report, is not included in these functions. This step was originally developed using a MATLAB script node and the two MATLAB functions e_uncorr and e_cal_sp, which were previously designed by Ted Clancy, who advised this project. However, MATLAB was not designed to be utilized in real-time applications. In order to support the inability to use MATLAB in real-time applications, a simple test was conducted to measure the time it takes to execute a MATLAB script node computing the simple addition of two real-numbers. The results of this test can be viewed in Appendix E. Therefore, in order to obtain a more realistic comparison of the latencies utilizing each of the different analysis methods, this gain normalization step was removed. In the future, this step can be implemented in LabVIEW to obtain a more complete EMG amplitude estimation routine.

5.2 Latency Test Procedure

In the previous section, three modules developed to perform an EMG amplitude estimation routine utilizing three different LabVIEW analysis methods were presented. These algorithms represent the type of computation and analysis that will be conducted with our PC-based workstation. As a result, it is important to grasp the latency associated with running a typical processing application and the affect that utilizing different LabVIEW analysis techniques and Windows XP priority levels has on these levels. Consequently, in this section a testing procedure created to determine the latency associated with running each of these versions of the EMG amplitude estimation algorithm is presented.

5.2.1 Timed-Loop VI

In order to make a comparison between the latency induced using three different LabVIEW analysis methods, we have created a VI to measure the time that it takes for a LabVIEW function to execute. Figure 5.1 depicts the VI that was formulated to conduct this testing. LabVIEW has a module that allows access to a 1 kHz system clock. Hence, our VI allows a routine to be placed inside of a loop and measures, using this system clock, the amount of time that it takes for the routine to execute a single iteration. Moreover, in order to gain statistical results over multiple trials, the VI allows the user to specify how many times this test is performed. Once all of the trials are complete, the results are displayed in a histogram and the mean and standard deviation of the results are also presented.

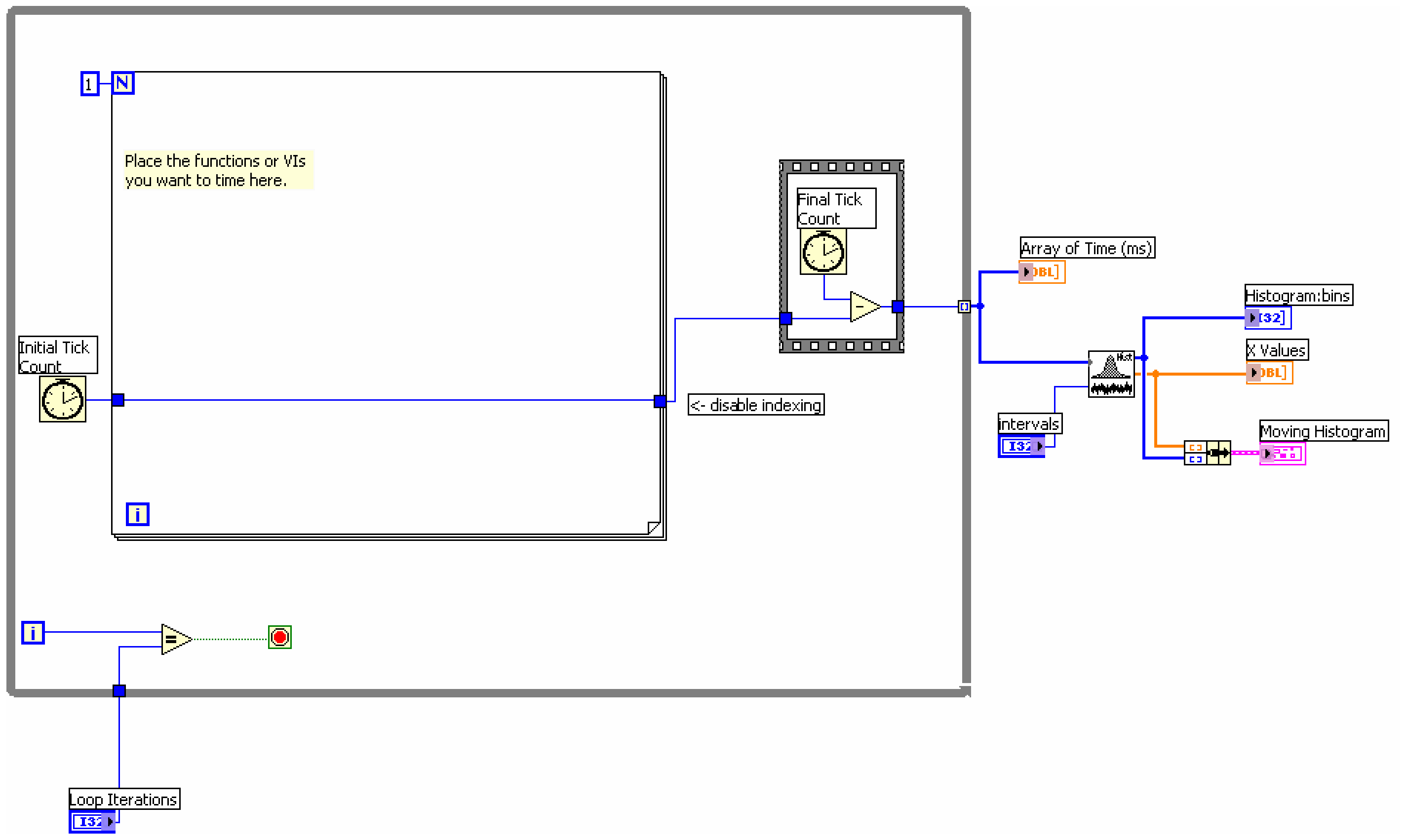


Figure 5.1: VI to Test Latency of LabVIEW Functions

5.2.2 Analysis Method Comparison Procedure

With a method created to measure the time it takes to execute a LabVIEW algorithm, an analysis of computational latencies induced by LabVIEW can be conducted. First, a 100 Hz square produced by a function generator was acquired from two analog input channels, at a sampling frequency of 500 Hz for approximately five minutes, and stored in a buffer. Next, using the timed-loop VI described earlier in the chapter, each EMG amplitude estimation VI was tested by utilizing these stored data. During testing, we kept the amount of filter coefficients consistent between the smoothing filter, high-pass filter and whitening filter. Moreover, the MVC value was set to a constant value rather than calculating it from acquired data. This was to ensure that external calls to C and MATLAB code did not affect the elapsed times, and that these time delays were strictly due to the DAQ board hardware, operating system and LabVIEW processing. Also, in order to conduct experiments under the same conditions, LabVIEW was the only foreground application running on the PC while testing was conducted.

For each EMG amplitude estimation routine, the recorded square waves were used as input. Next, using 512 filter coefficients all three filter values, the time-loop VI was executed 100 times. The default priority for a process in Windows XP is Normal. Consequently, in order to determine the effects of using different priorities has on the latency induced by the EMG algorithm, these testing procedures were performed using high and Real-Time priorities as well. Moreover, in order to get results using increasing amounts of filter coefficients, this procedure was repeated using filter coefficient values of 1024, 1536 and 2048. Again, each of these various tests were conducted at all three priority levels. Furthermore, while the routine implementing point-by-point analysis operates on one piece of datum every interval, the array-based and Express VI routines operate on packets of arrays. Consequently, for these two routines, two different array sizes were utilized: array sizes of one and ten. While the processing routines were configured to process data as quickly as possible, these graphs were configured to update after every ten datum points were analyzed.

5.3 Timed-Loop Test Results

The results of these various tests can be seen in Tables 5.1, 5.2 and 5.3. In particular, Table 5.1 indicates all experimental results from the point-by-point EMG function, Table 5.2 indicates all experimental results from the Express VI function and Table 5.3 indicates all experimental results from the standard array-based VI. It should be noted that for the Express VI version of the algorithm, test results are only displayed for utilizing 512 taps for all three filters. The reason for this limitation is that 512 is the maximum amount of filter coefficients that are allowed to be used by this function. The raw results from all of the experiments described in the previous section can be seen in their entirety in Appendix F.

Taps for all Three Filters	Windows Priority Setting	Mean (ms)	Standard Deviation (ms)
512	**Normal**	0.12	0.33
	High	0.11	0.31
	Real-time	0.10	0.30
1024	**Normal**	0.13	0.34
	High	0.14	0.35
	Real-time	0.16	0.37
1536	**Normal**	0.19	0.39
	High	0.16	0.37
	Real-time	0.18	0.38
2048	**Normal**	0.23	0.44
	High	0.18	0.38
	Real-time	0.22	0.41

Table 5.1: Experimental Results of Point-by-Point EMG Function

Array Size	Windows Priority Setting	Mean (ms)	Standard Deviation (ms)
1	**Normal**	2.36	3.49
	High	2.18	3.62
	Real-time	2.13	3.33
10	**Normal**	2.39	3.60
	High	2.71	3.72
	Real-time	2.40	3.60

Table 5.2 Experimental Results of Express VI EMG Function where 512 Taps were used For All Three Filters

Array Size	Taps for all Three Filters	Windows Priority Setting	Mean (ms)	Standard Deviation (ms)
1	512	**Normal**	0.70	0.78
		High	0.61	0.79
		Real-time	0.67	0.92
	1024	**Normal**	1.18	0.77
		High	1.10	0.84
		Real-time	1.30	1.94
	1536	**Normal**	2.25	1.49
		High	1.92	1.25
		Real-time	1.91	1.28
	2048	**Normal**	2.87	1.51
		High	2.69	1.39
		Real-time	2.88	1.41
10	512	**Normal**	0.94	1.17
		High	0.81	0.95
		Real-time	0.80	0.92
	1024	**Normal**	1.65	1.19
		High	1.50	1.20
		Real-time	1.48	1.04
	1536	**Normal**	2.25	1.38
		High	2.07	1.34
		Real-time	2.04	1.34
	2048	**Normal**	2.96	1.76
		High	2.62	1.49
		Real-time	2.59	1.42

Table 5.3 Experimental Results of Array-Based EMG Function

5.4 Discussion

Upon analyzing the results from the previous section, several observations can be made regarding the latency associated with utilizing each analysis method. In particular, one can see that the point-by-point analysis version of the EMG algorithm executes at a significantly lower average latency when compared to the two array based method, even when an array buffer size of one is utilized. By looking at the average latency and standard deviation of these data collected during the point-by-point analysis routine, preliminary results suggest LabVIEW may be suitable for low-latency applications.

First, it is rather apparent that, as background research initially suggested, that the EMG amplitude estimation VI utilizing point-by-point analysis exhibits the lowest latency. This notion is enforced in Figures 5.2, 5.3 and 5.4. These three figures display the average latency, in milliseconds, versus the number of filter coefficients used for the highpass, smoothing and adaptive whitening filters in the EMG amplitude estimation routine. These three graphs respectively depict results for when LabVIEW is run with normal, high and real-time priority.

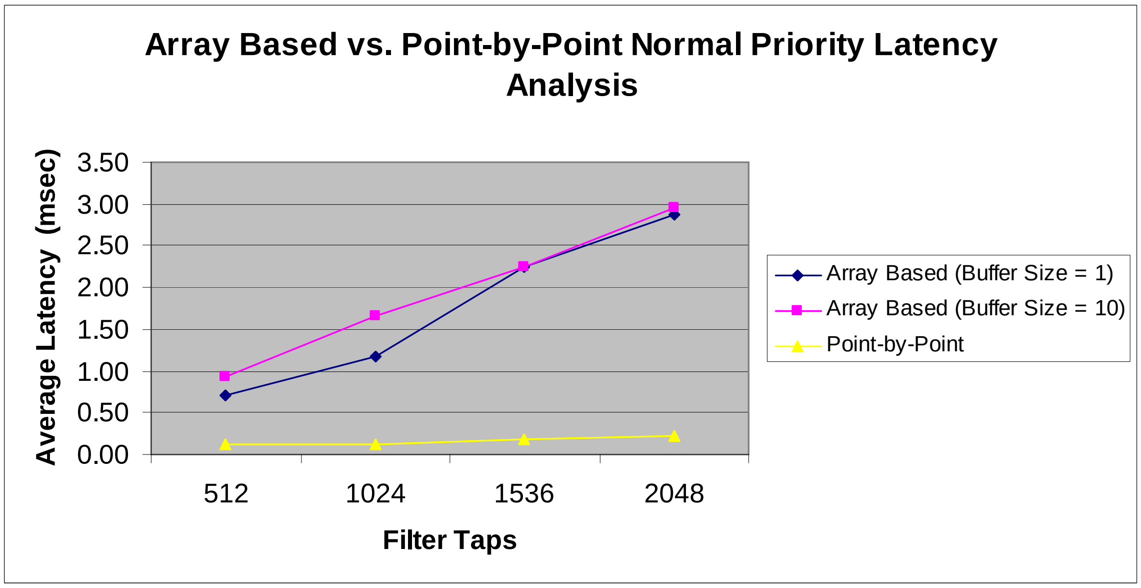


Figure 5.2: Latency of EMG Array Based and Point-by-Point Algorithm, Normal Priority

As all three graphs indicate, the point-by-point routine always executes with the lowest latency. In fact, the average latency for the point-by-point routine remains relatively close in value while the latency incurred while running the array-based algorithm grows steadily when an increased number of filter coefficients are utilized. Moreover, even when a buffer size of one is utilized for the array based routine, the point-by-point routine executes at a substantially lower latency. This latency difference can be attributed to the extra overhead that is associated with executing an array based function in LabVIEW. However, even with this additional overhead, it is apparent that both the array based and point-by-point EMG amplitude estimation algorithms have latencies that are below our defined five millisecond maximum.

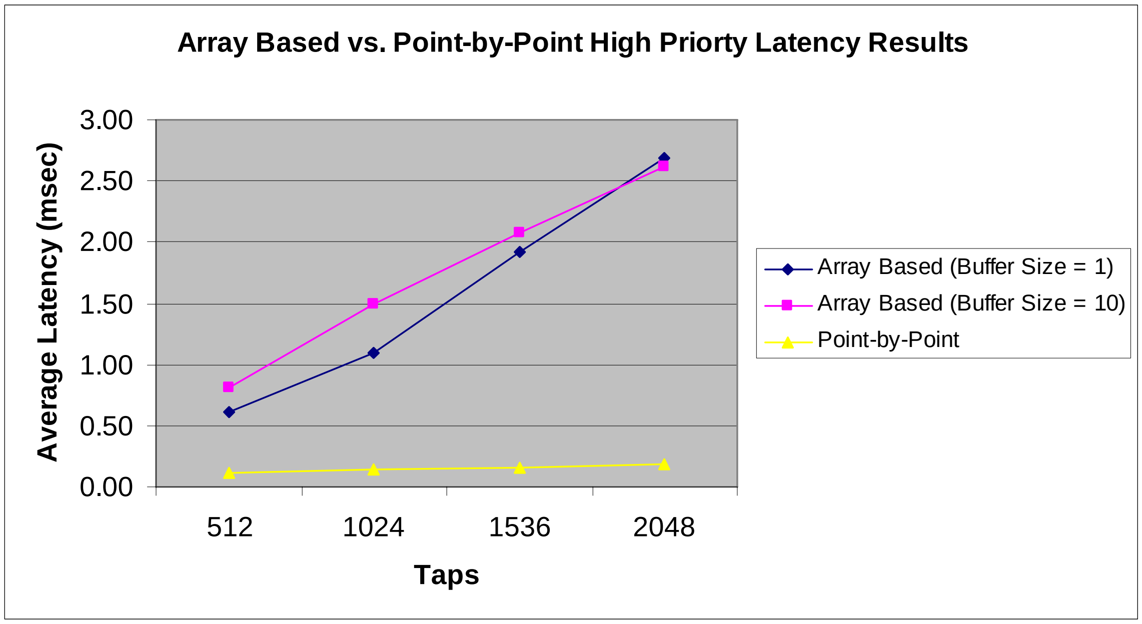


Figure 5.3: Latency of EMG Array Based and Point-by-Point Algorithm, High Priority

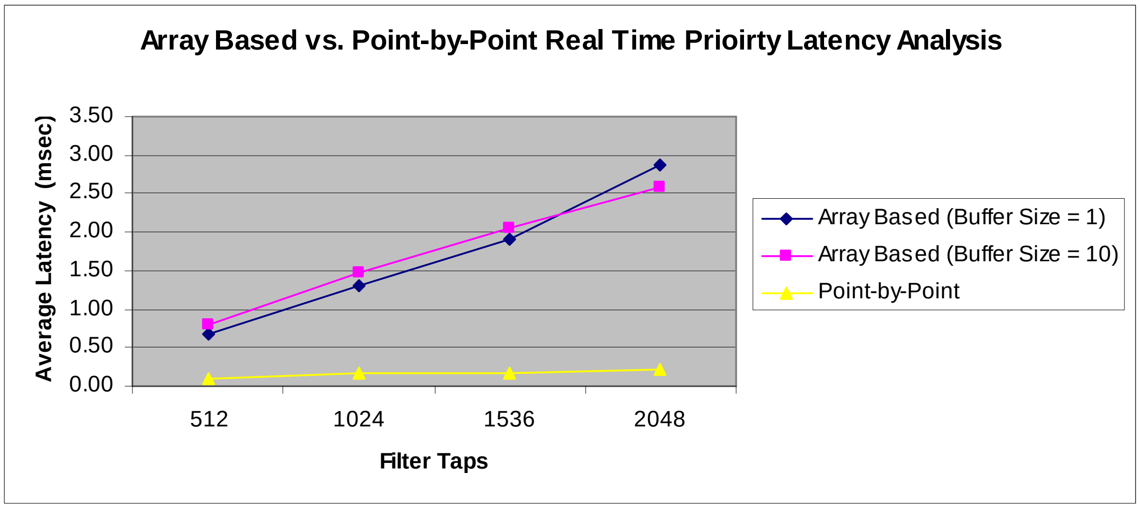


Figure 5.4: Latency of EMG Array Based and Point-by-Point Algorithm, Real-Time Priority

The previous three graphs did not depict results obtained when using the Express VI implementation of the EMG routine since Express VIs, as discussed previously, are limited to utilize only a maximum of 512 taps for filters. Consequently, Figure 5.5 depicts results for executing all three versions of the EMG algorithm with a constant value of 512 taps for each filter utilized in the routine. Although, a buffer size of one is utilized for both the array based and Express VI routines, the point-by-point routine is shown to execute at a lower-latency. Moreover, this graph displays that Express VIs, while convenient, incur substantially larger latencies then the other analysis methods.

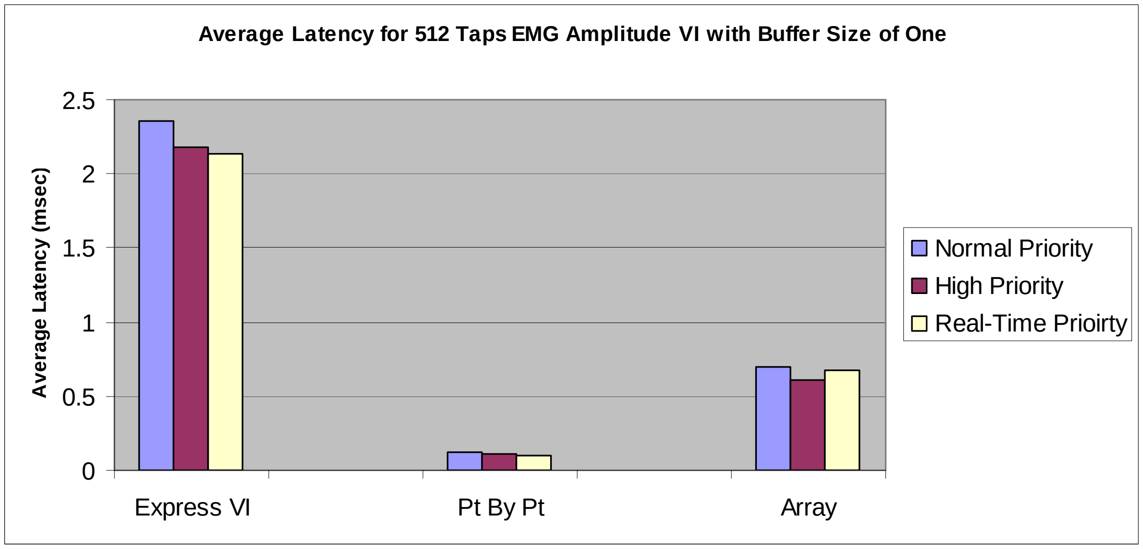


Figure 5.5: Average Latency for Buffer Size of One

Moreover, Figure 5.5 indicates that the priority that LabVIEW is executed at does not have a great effect on the average latency of a LabVIEW routine. Furthermore, running LabVIEW at a higher priority does not guarantee that a given function will execute at a lower latency. The idea that running LabVIEW at higher priorities does not guarantee the smallest latency can be seen in Figure 5.6, which displays the average latency of the point-by-point algorithm running at varying priorities. As the graph indicates, while the latency incurred while executing the LabVIEW application in real-time priority induced a minimum latency when utilizing only 512 taps, executing

LabVIEW in Windows in real-time priority garnered the highest latency when 1024 taps were used.

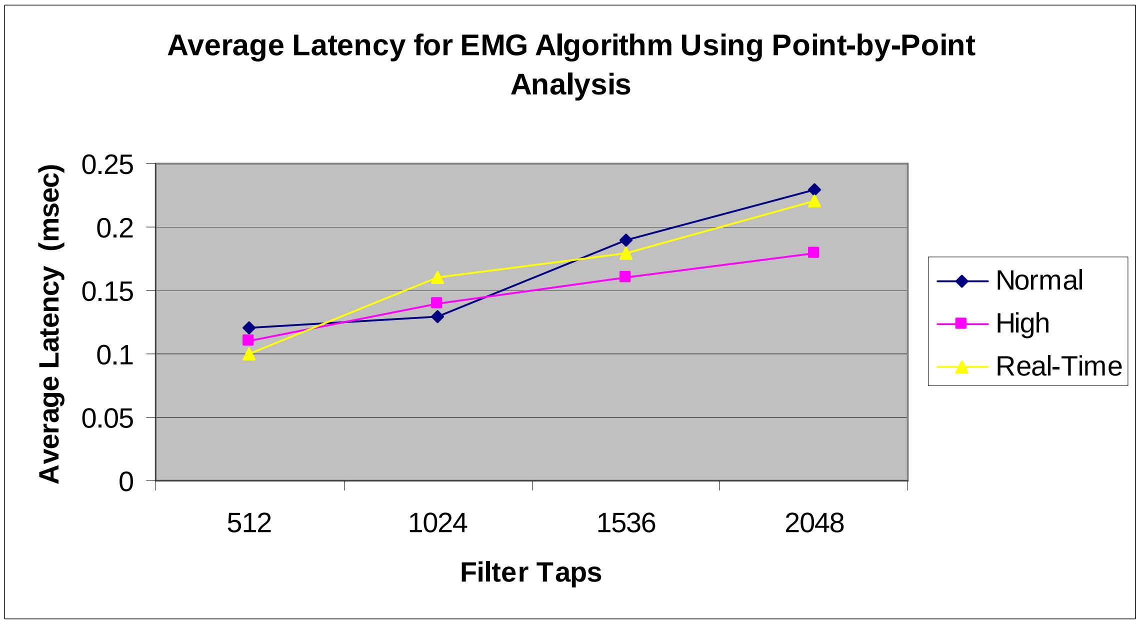


Figure 5.6: Average Point-by-Point Latency for Different Priority Levels

Therefore, it is apparent that the average latency for the point-by-point algorithm and the standard array-based algorithm utilizing a buffer size of one are quite small and indicate that LabVIEW may be an acceptable option to conduct the hard real-time biomedical experiments that were previously described. However, the average latency is not the only statistical figure that must be configured in analyzing these data. In particular, the standard deviation is also a statistical measurement that is used to analyze data. By definition, 68 percent of data are captured one standard deviation away from the mean in either direction. Moreover, between 95 and 99 percent of data are respectively captured two and three standard deviations away from the mean.

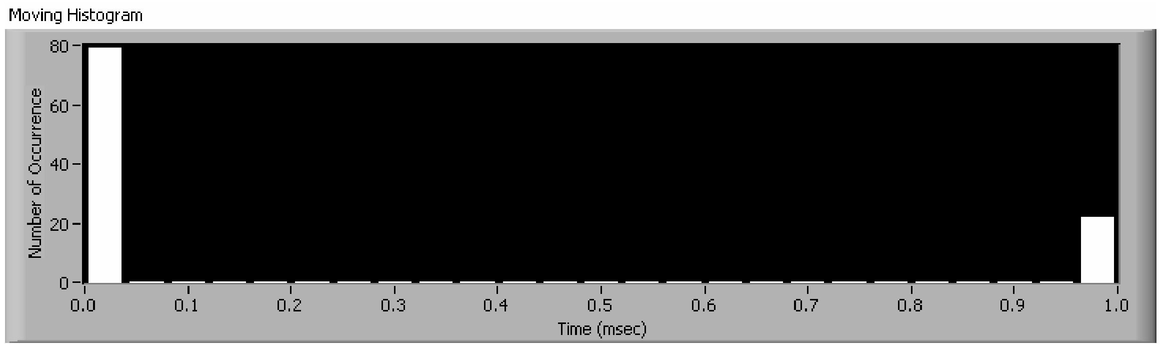


Figure 5.7: Raw Data Histogram Point-By-Point 2048, Real-Time Priority

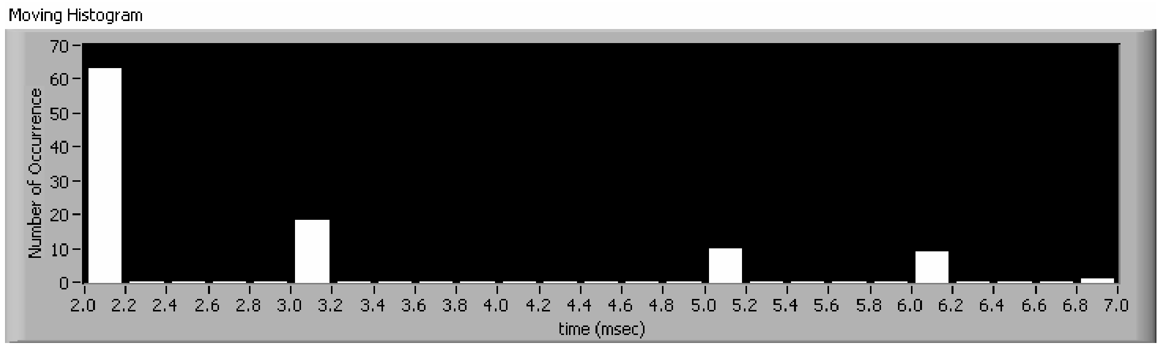


Figure 5.8: Raw Data Histogram Array-Based (Buffer Size = 1) 2048, Real-Time Priority

The standard deviation for all experiments conducted can be seen in Tables 5.1, 5.2 and 5.3. As these data indicate, the standard deviation for all of the point-by-point experiments is quite small, on the order of fractions of milliseconds. However, the standard deviation from the mean for the other experiments is quite large, giving evidence that there should be a concern for outliers. Outliers are defined as datum points that are far away from the rest of a set of data. Figure 5.7 and 5.8 displays the raw data collected when 2048 taps were utilized for the array-based algorithm with a buffer size of one and for the point-by-point routine when the priority of LabVIEW was set to real-time. While the average latency for this point-by-point experiment was computed to be

.22 milliseconds, the raw data indicates that outliers of one millisecond are possible. For our hard real-time requirements, this is not a concern. However, when examining the array-based routine using a buffer size of one, the concern of outliers can be seen. Although the average latency for this routine was 2.88 milliseconds, which was below our defined maximum latency, outliers can be viewed that are as great as 7 milliseconds.

As this example shows, while the average latency is an important characteristic of our system, the worst-case latency is even more critical. Although our algorithms may execute quicker then our average latency most of the time, a large latency that exceeds our maximum latency indicates system failure. However, the frequency with which these outliers occur is an important measurement to acquire. While these large outliers cause for system failure, if they occur infrequently, such as on the order of hours, LabVIEW may still prove capable to meet our hard real-time needs. As a result, another test must be conducted to examine the frequency with which these outliers occur.

5.5 Chapter Summary

In this chapter we investigated ways to determine the latency delay caused by LabVIEW functions. We developed a module that measures this latency and then analyzed an EMG amplitude function utilizing three different methods of processing. We found that functions that utilize point-by-point analysis exhibited the lowest latency, and functions that make use of Express VIs exuded the highest latency. However, while the average latency is acceptable for the point-by-point and array-based routine using a buffer size of one, these algorithms occasional may exhibit outliers that are above our maximum latency threshold. As a result, further tests must be performed to determine how frequently these large outlier values occur. If these outliers occur infrequently, such as on the order of hours, then it is possible that this LabVIEW system can still be utilized to meet our hard real-time requirements.

6. Hard Real-Time LabVIEW Performance

In the previous chapter latency tests were conducted that compared system performance when different function types were utilized to conduct an EMG amplitude estimation algorithm. Results from these tests indicated that the average latencies of the point-by-point and array-based routine using a buffer size of one were below our maximum latency threshold. However, while the average latency was viewed as acceptable, by looking at the raw data it is apparent that outliers that exceeded our maximum latency threshold were present. Thus, another latency test was conducted that served as a measurement of how frequently these outliers occurred. If the frequency of these outliers is small, then it is still possible for this PC-based system to conduct our previously described hard real-time experiments. This chapter presents our testing procedure and the results that were garnered when this test was executed.

6.1 Final Testing Procedure - MTBF

For our final latency test procedure, we wanted to determine how long our system could run, at various sampling rate, before losing a data point due to a large outlier. In many hard real-time systems, a dropped point can be tolerated without issue, as long as too many are not dropped in a specific time period. In order to find out how often our system was dropping points, we came up with a system to measure the Mean Time Between Failures (MTBF) of our system. This created procedure can be seen in Figure 6.1.

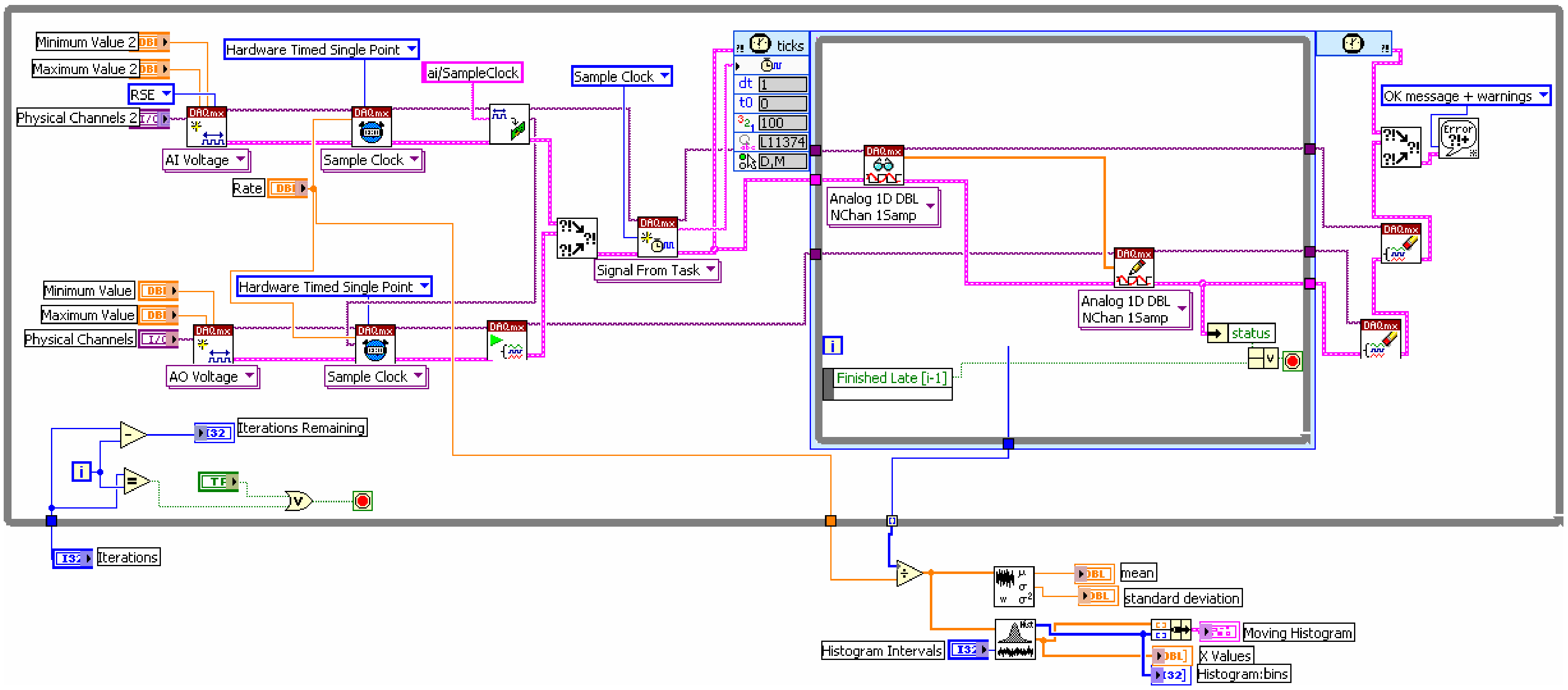


Figure 6.1: MTBF Test VI

In the VI that is depicted in Figure 6.1, an analog input and an analog output task are first created. The output routine is located inside a loop, whose timing is controlled by the hardware timer on the DAQ board (unlike output routines utilizing the FIFO, which are dependent on the timing of the operating system). When a loop is configured with a hardware time, there is a signal available in the loop, labeled *"finished late,"* that is asserted when the loop has not completed by the time the hardware clock triggers it to run again. In our VI, this signal is used to stop the internal, hardware timed loop. When the loop is stopped, the number of iterations that was completed before failure is passed to an outer loop. These data are then passed to a function which converts the number of iterations completed to a length of time that the loop ran before error. Like the timing VI created for measuring function latency, this routine can be configured to run multiple times. As a result, a histogram exhibiting the results from multiple loop executions is presented along with the mean and standard deviation of these multiple iterations.

In our testing we varied the sampling rate of the DAQ board (200, 500, 1000 and 2500 Hz), and the Priority Level LabVIEW was running at (Normal, High and Real-time), and produced histograms of times between failures for an input signal to write to analog output without any computation. We then used these data to calculate MTBF and standard deviation of these data sets. Next, to determine if the number of channels utilized affected performance, we performed the MTBF test described above using one, two and four analog input channels. Lastly, we executed this VI with a sampling frequency of 200 Hz. Postulating that the MTBF would be small for this sampling frequency, we then added a point-by-point FIR filter (since previous results indicated point-by-point filters induced the lowest latency) with varying lengths of filter coefficients (500, 5000 and 50000) to determine the effect on the MTBF. The results calculated from these experiments can be viewed in the following section.

6.2 Results

The following section displays the results that were garnered by conducting the tests described in the previous sections. Tables 6.1, 6.2 and 6.3 respectively display the results that were garnered when the MTBF test was conducted when using 1, 2 and 4 analog input channels. Again these tests involved calculating the MTFB when an input signal was directly passed to an output channel with no processing involved.

Sampling Frequency (Hz)	Windows Priority Setting	MTBF (sec)	Standard Deviation (sec)
500	Normal	4.09	4.11
	High	5.21	4.26
	Real-Time	7.74	3.62
1000	Normal	0.36	0.18
	High	0.35	0.19
	Real-Time	0.54	0.04
2000	Normal	0.20	0.15
	High	0.30	0.20
	Real-Time	0.48	0.15
2500	Normal	0.01	0.02
	High	0.22	0.19
	Real-Time	0.34	0.22

Table 6.1: MTBF Results Obtained Using 1 Analog Input/Output Channel

Sampling Frequency (Hz)	Windows Priority Setting	MTBF (sec)	Standard Deviation (sec)
500	Normal	4.54	4.22
	High	3.37	2.64
	Real-Time	4.24	2.45
1000	Normal	0.32	0.20
	High	0.36	0.19
	Real-Time	0.56	0.02
2000	Normal	0.15	0.15
	High	0.31	0.20
	Real-Time	0.51	0.14
2500	Normal	0.01	0.01
	High	0.25	0.20
	Real-Time	0.33	0.22

Table 6.2: MTBF Results Obtained Using 2 Analog Input/Output Channels

Sampling Frequency (Hz)	Windows Priority Setting	MTBF (sec)	Standard Deviation (sec)
500	Normal	4.19	3.23
	High	5.84	2.27
	Real-Time	3.15	3.40
1000	Normal	0.35	0.19
	High	0.36	0.19
	Real-Time	0.56	0.04
2000	Normal	0.21	0.17
	High	0.35	0.18
	Real-Time	0.50	0.14
2500	Normal	0.01	0.02
	High	0.34	0.21
	Real-Time	0.34	0.21

Table 6.3: MTBF Results Obtained Using 4 Analog Input/Output Channels

For small sampling frequencies, it was expected that the MTBF would be quite large. Table 6.4 displays the results garnered from calculating the MTBF at the small sampling frequency of 200 Hz. These results were garnered when LabVIEW was operating with normal priority. The FIR filter used to induce processing was a point-by-point FIR windowed filter, set to operate as a lowpass filter with no window and an arbitrary cutoff frequency value. The raw results for all of these tests can be viewed in Appendix G.

Sampling Frequency (Hz)	Number of FIR Filter Coefficient	MTBF	Standard Deviation (sec)
200	None	17 Hours Without Failure	N/A
	500	13.7	14.59
	5000	14.01	13.77
	50000	1.24	.54

Table 6.4: MTBF Results for 200 Hz Sampling Frequency using Varying Computation Loads at Normal Priority with One Analog Input Channel

6.3 Discussion of Results

The previous section displayed the results that were garnered from our hard real-time testing of standard LabVIEW running on the Windows XP operating system. These results establish several interesting trends. Figures 6.2 graphically display the MTBF results obtained when running the previously described timing test with no computational load for one analog input and output channel. As the graph indicates, the MTBF is dependent on the sampling frequency utilized. For low sampling frequencies, the MTBF is approximately a maximum of eight seconds. However, as the sampling frequency increases, the MTBF drops to levels on the order of milliseconds.

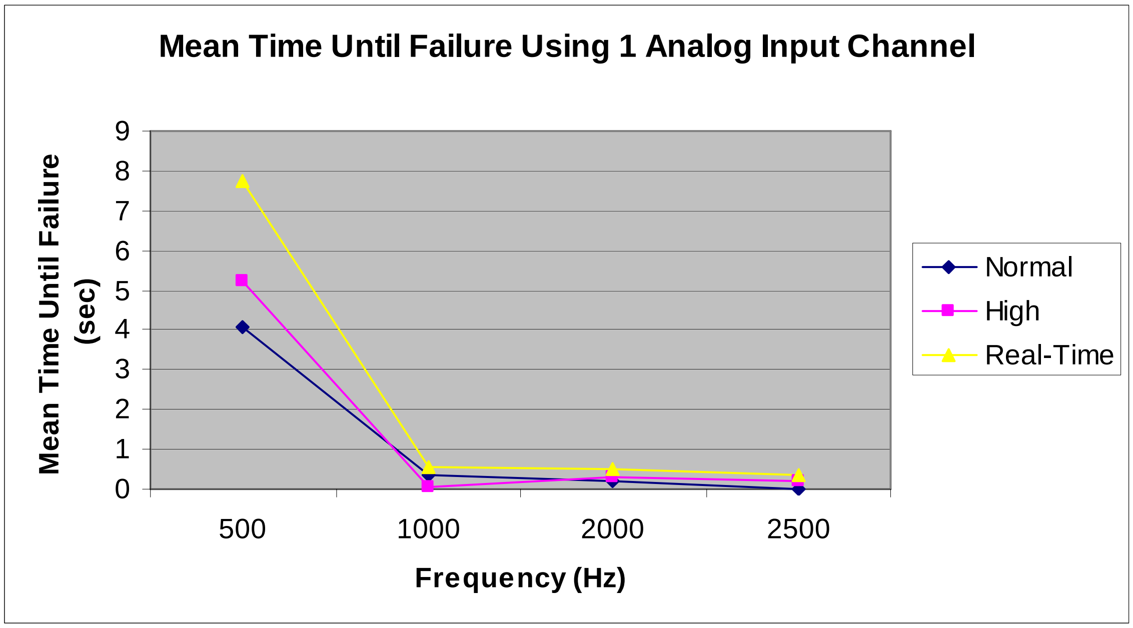


Figure 6.2: MTBF 1 Analog Input/Output Channel

Moreover, as Figures 6.3 and 6.4 indicate, these low MTBF results are almost identical for utilizing 2 and 4 analog input and output channels. These similar results are appropriate due to the architecture of the PCI-6229. As previously discussed there is only a single DMA controller to control the analog input task and a single DMA controller to control the analog output task and only one A/D and D/A converter for input and output routines. As a result, for multiple channels, the sampling frequency, per channel, is equal to the total sampling frequency divided by the number of channels. These garnered results reinforce this notion and indicates that the DAQ has the same processing delay no matter the amount of analog input and output channels used.

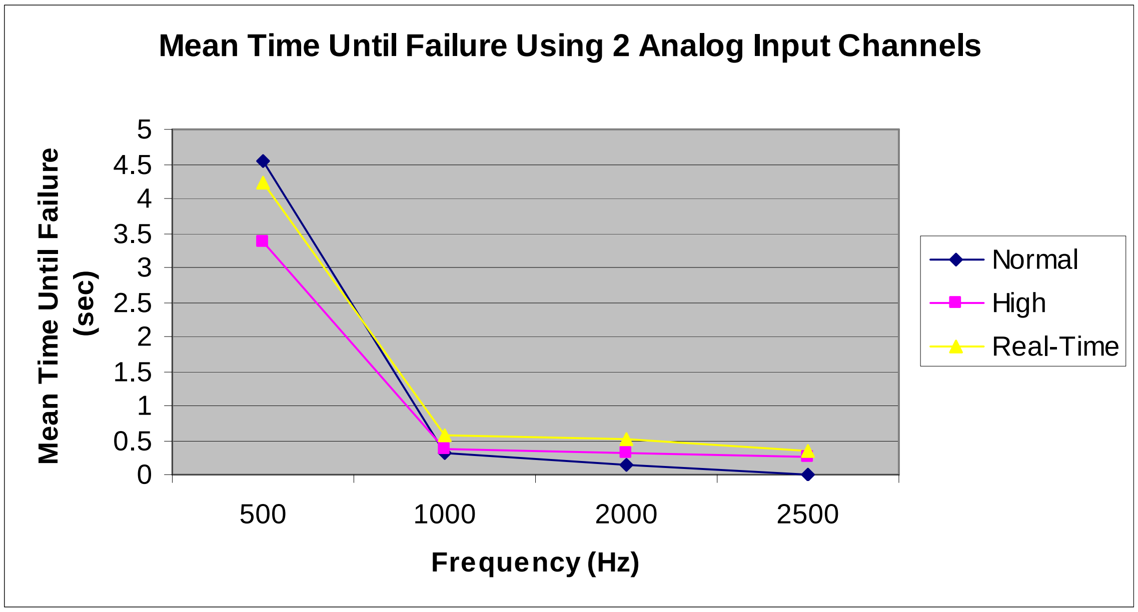


Figure 6.3: MTBF 2 Analog Input/Output Channels

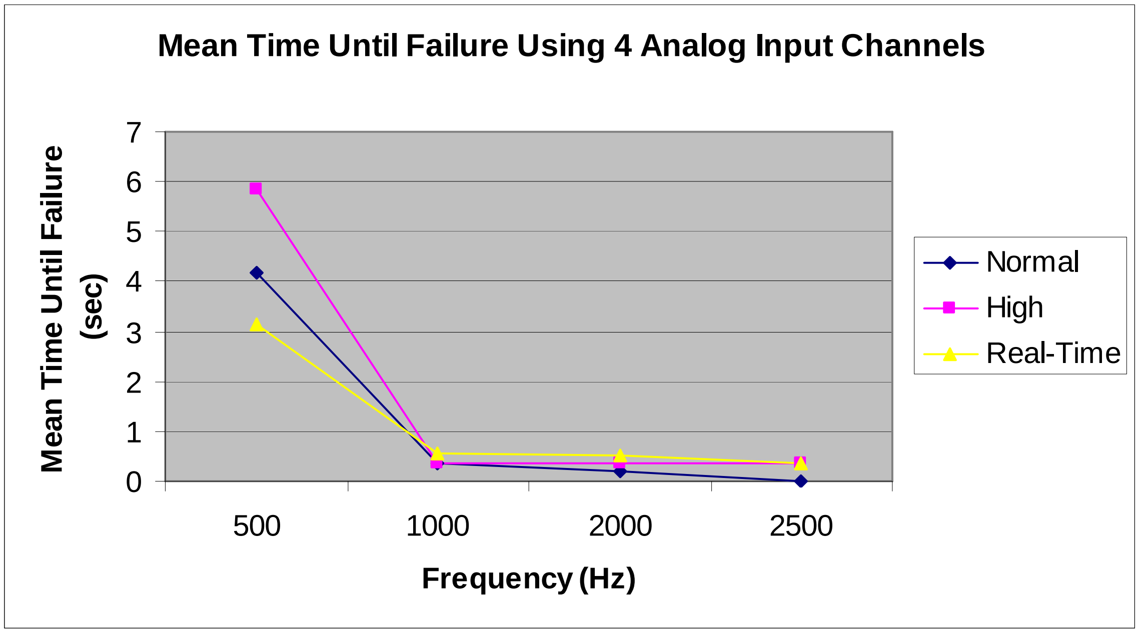


Figure 6.4: MTBF 4 Analog Input/Output Channels

Consequently, for frequencies above 500 Hz, it is apparent that LabVIEW is not appropriate for conducting hard real-time control experiments as a result of the frequency of data points being missed even without a computational load. Moreover, increasing priority, as shown on these three previous graphs, does not guarantee an increase in MTBF but, instead, decreases the standard deviation from the MTBF.

While it is apparent that LabVIEW can not conduct hard real-time control experiments with sampling frequencies greater then approximately 500 Hz without losing numerous data points, preliminary evidence suggested that a PC-based system might be able to perform hard real-time control applications at low frequencies. In particular, when the MTBF test was run with no computation at a sampling frequency of 200 Hz with LabVIEW operating with normal priority, seventeen hours elapsed and no failures occurred. As a result, this would seem to indicate that a hard real-time control experiment could be conducted at low frequencies. However, a FIR filter was added as computation to see if these impressive results could remain constant. As Figure 6.5 shows, when computation was added, the MTBF went to not having any failures after seventeen hours, to having an average failure in the magnitude of seconds. With a simple point-by-point filter which was shown to induce the lowest computational latency) causing such dramatic MTBF changes, it is readily apparent that LabVIEW is not optimal for control experiments requiring hard real-time control. However, if an application is capable of missing data points on the order of every second, or less, then a LabVIEW based system certainly not be used..

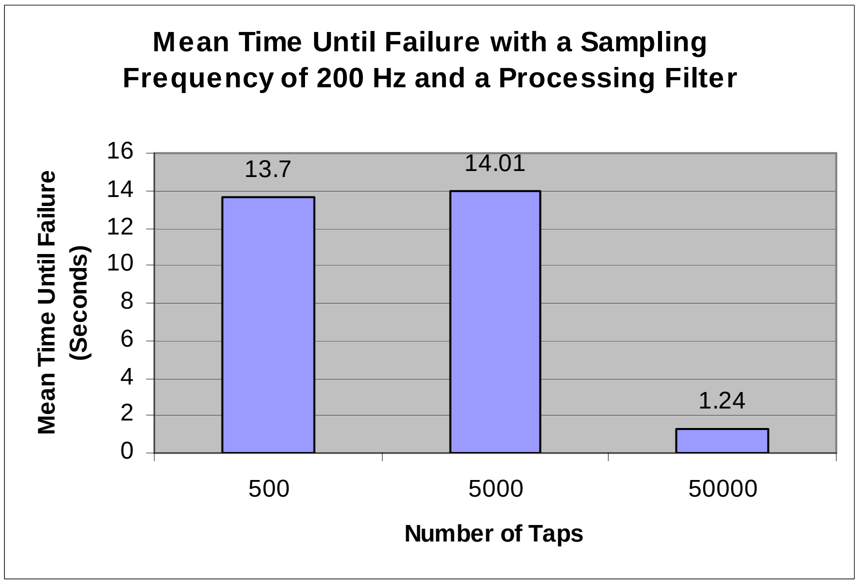


Figure 6.5: MTBF w/ Computation Load at 200 Hz Sampling Frequency

6.4 Limitation of Windows Operating System

As shown in the discussion of our results, the MTBF for our system is entirely too small. While it may be acceptable, for some of the experiments that would be conducted with our workstation, for data points to be missed on the order of once every few hours, losing data points on the order of every second is unacceptable. The reason for the high frequency of data points being missed can directly be attributed to the limitations of the Windows operating second.

The basic scheduling procedures of Windows XP were discussed in the background section of the report. In particular, the fact that Windows XP utilizes a non-preemptive scheduling policy was presented. Upon further review of the operating system, it is apparent that Windows based systems use clocks with a resolution of one millisecond. In particular, the time quantum, or amount of time that a given thread is allowed to execute before a decision is made to either perform a context switch or continue execution, is approximately nine milliseconds. Hence, if a context switch occurs while LabVIEW is executing, a latency of at least nine milliseconds will occur. This large latency periodically introduced by the operating system is thus the cause of the low MTBF. [13]

6.5 Chapter Summary

In this chapter we discussed our method for testing the performance of LabVIEW as a real time signal processing system. Our results indicate that data samples are missed on the order of every second with no computational load for frequencies greater then 500

Hz. For frequencies below 500 Hz, the MTBF was rather high when there was no computational load. However, when a computational load, in the form of a simple point-by-point FIR, was introduced at 200 Hz, the MTBF decreased dramatically. These low MTBF are a result of the high latency outliers that frequently occur as a result of Windows XP performing context switches. Hence, a PC-based system utilizing LabVIEW would be well-suited for non real-time systems, or soft real-time systems that allow for several data samples to be missed, and not the hard real-time tasks that we desire to perform.

7. Using LabVIEW in a Data-Logging Application

Although the past few chapters have suggested that LabVIEW is not suitable for the types of hard real-time control application we were aiming to implement, namely for real-time signal processing of EMG signals, a PC-based system using LabVIEW does have some other potential uses. In particular, such a system can be used for offline processing. In offline processing, as the name suggests, data are not processed in real-time. Rather an amount of data are collected and then processed at a later time, with delay not being a concern.

With the NI PCI-6229, a FIFO of 4095 samples is available. Consequently, approximately one second of buffering can be obtained when a sampling frequency of 4096 Hz. With this kind of delay between data coming in, some processing in between new buffers of data being available for processing can be conducted. The reason that some processing can occur between new packets of data being available is that most processing, while not wholly deterministic, takes on the order of milliseconds rather than seconds to complete in Windows. [15]

Another option available using an offline system is to stream recorded data to RAM. While data can be saved to hard disk, performing such an operation can cause for additional latencies to occur. In particular, when disk seek operations must be performed, latencies can occur that are on the order of tens of milliseconds. By streaming data to RAM and then transferring to disk in non real-time, these latency issues can be averted [30]. With data saved in memory, later processing by LabVIEW, MATLAB or any other sort of processing program can be conduct. Hence, while it is apparent that LabVIEW is not suitable to conduct hard real-time control experiment, LabVIEW can be utilized as a data logger for offline applications.

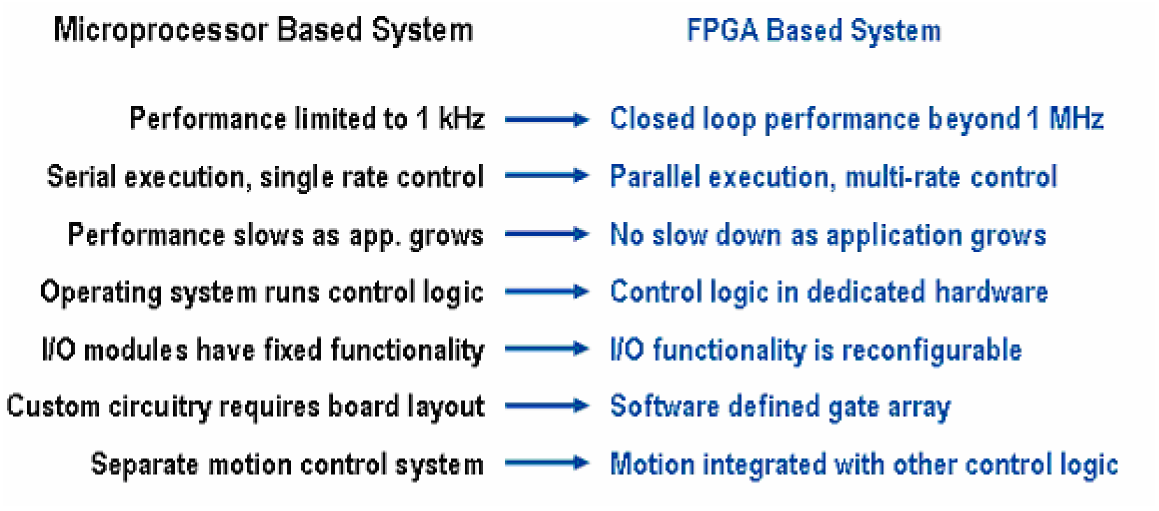
8. Future Recommendations

From our work described in the previous chapters, it is apparent that the PC-based system that we designed is not optimal for conducting experiments that involve hard real-time control. In this chapter, we will investigate further analysis and platform options that can be chosen to further advance the creation of a work station to conduct biomedical experiments.

8.1 Other LabVIEW Design Options

While the current system is not capable of meeting our hard real-time needs as a result of the Windows operating system, it may be possible to still utilize the standard LabVIEW software package to implement our design. In particular, LabVIEW is now available for the LINUX operating system. By using the LINUX version of LabVIEW, one can make a determination if a change in operating systems will allow for LabVIEW to be a more affective and viable option.

While using LabVIEW on another operating system is an option, there are other National Instruments products that are available for use with standard LabVIEW. These products can be utilized on a Windows PC that can potentially allow for our real-time specification to be met. Two of these products that can be utilized to help achieve hard real-time performance are the LabVIEW FPGA (Field-Programmable Gate Array) and LabVIEW RT (Real-Time) modules.



Microprocessor Based System		FPGA Based System
Performance limited to 1 kHz	→	Closed loop performance beyond 1 MHz
Serial execution, single rate control	→	Parallel execution, multi-rate control
Performance slows as app. grows	→	No slow down as application grows
Operating system runs control logic	→	Control logic in dedicated hardware
I/O modules have fixed functionality	→	I/O functionality is reconfigurable
Custom circuitry requires board layout	→	Software defined gate array
Separate motion control system	→	Motion integrated with other control logic

Figure 8.1: Impact of FPGA based system [18]

The LabVIEW FPGA module offers additional command over NI hardware by giving the user control of FPGA chips on select NI RIO (Reconfigurable I/O) hardware. This module allows the user to program the FPGA chip with user created hardware, execute multiple tasks simultaneously and customize timing with a resolution of up to 25 nanoseconds. Implementation of an FPGA system would require purchase of a NI R series DAQ board which contains the RIO hardware used by LabVIEW FPGA. An outline of some of the benefits of using an FPGA based system over a microprocessor based system can be seen in Figure 8.1.

Another design option is to utilize the LabVIEW Real-Time (RT) module. LabVIEW RT delivers increased reliability in time control by utilizing an embedded real-time operating system on a National Instruments' hardware device. This embedded hardware device is used to execute a LabVIEW program that is present on a PC. The benefit of using LabVIEW RT is that deterministic results can be garnered. In particular, unlike standard LabVIEW, a system of internal LabVIEW priority controls can be utilized. As a result, the speed of which certain aspects of LabVIEW code executes can be configured. Hence, a processing routine can be configured to run at a frequency of 1 KHz while a graphical output routine can be configured to run at a frequency of 60 Hz [7]. Clearly, such a system would be ideal for our hard real-time processing needs.

Hence, LabVIEW RT and FPGA offer two alternatives for utilizing LabVIEW to conduct our hard real-time experiments. However, while these options appear to offer a viable solution for creating our workstation, the cost of these devices is rather high. In particular, a LabVIEW FPGA board with 8 analog input and output channels can cost approximately $4000, without including the price of any of the necessary FPGA software drivers,. Hence, spending this much money on a component eliminates the benefit of using a cost effective PC-based system. Similar large prices are apparent for the other RT devices as well.

8.2 Other Alternatives

While possible alternatives for implementing a PC-based system were shown that use additional LabVIEW modules, many other alternatives exist, each solution possessing unique advantages and disadvantages. For instance, proprietary systems such as VXWORKS, QNX and REAL/IX are potential options because they are designed specifically for applications similar to the ones that we wish to conduct. While these systems do provide the functionality that is desired, their major drawback is that they are expensive and not widely available.

Another viable option is to utilize a real-time Linux based system, known as RT-Linux. RT-Linux is a free, open-source operating system that provides the deterministic results that are the chief characteristic of real-time operating systems. RT-Linux has already been adopted by researchers who are conducting biomedical applications that are similar to the tasks we have attempted to create. For instance, a research group at the University of Texas developed a RT-Linux system to detect and minimize heart arrhythmias by using electrical stimuli on the heart muscle [6]. While RT-Linux does prove to be a reasonable alternative to LabVIEW, the Linux operating system is not as widely used as Windows. As a result, people would need to become familiar with this new operating-system and programming in it if RT-Linux was chosen.

Yet another option is to utilize an embedded system, such as a digital signal processing (DSP) board, to implement this system. For instance, in 1999, students at the University of Alabama built a holter system using a Texas Instruments data acquisition board [19]. However, as previously discussed, embedded systems are often more difficult and cumbersome to configure then PC-based systems. In particular, individuals programming embedded systems must be more familiar with all of the hardware components of the board and, in most instances, write their own personal drivers for other input and output devices. Hence, an embedded system, although it provides the capability to execute hard real-time routines, is difficult to maintain.

8.3 Chapter Summary

Our results and analysis indicate that a PC-based system using LabVIEW can be used in data logging applications. However, standard LabVIEW on a Windows machine provides nondeterministic results and, therefore, is not suitable for hard real-time control experiments. The proceeding section outlines future recommendations to individuals who will continue working on creating a workstation to conduct biomedical experiments.

These design options include using different LabVIEW modules, such as LabVIEW FPGA and LabVIEW RT, or even utilizing a different operating system, such as RT-LINUX. While all of these options are feasible alternatives to a PC-based system using standard LabVIEW, each design option offers a tradeoff between functionality and maintainability.

9. Conclusion

In order to help continue research related to the study of EMG signals, it is helpful to researchers to have a system that can conduct biomedical experiments. While there are a myriad of design options available to create a workstation to conduct biomedical experiments, our task was to investigate to what extent a PC-based system, utilizing the Windows XP operating system and the LabVIEW software package, can be used to conduct biomedical experiments.

A possible desired application for our PC-based system would be to conduct control experiments that involve simultaneous input and output, our work has suggested that using standard LabVIEW in conjunction with Windows XP is not currently suitable for such an application. For instance, although our PC-based workstation was successfully able to input and then output a function sampled at 200 Hz for 17 hours without conducting any processing, once computation in the form of a FIR filter was added, it was determined that every 14 seconds a datum point was missed. Consequently, losing points of data so frequently is not acceptable for a hard real-time system and, therefore, it is apparent that using LabVIEW in conjunction with Windows XP does not provide desired results for control experiments.

Although a PC-based system utilizing LabVIEW does not seem appropriate to conduct control experiments, this PC-based system can be utilized in data-logging applications. That is, due to the large FIFO buffer on the PCI-6229, LabVIEW can be utilized to acquire and save data on a PC. Then, after these data have been recorded, analysis can be performed on these data using LabVIEW in a non real-time application.

Therefore, while LabVIEW appears inappropriate for such applications as controlling a prosthetic limb in hard real-time, our PC-based system can be utilized in other off-line experiments. However, if it is desired to use LabVIEW in a PC-based hard real-time application, NI FPGA and other real-time boards can potentially be utilized. These boards contain on-board processors, which are solely dedicated to performing desired real-time tasks, and thus can potentially eliminate the deficiencies caused by utilizing a computer running a non real-time operating system.

Works Cited

[1] Basmajian, J.V and De Luca, C.J., “Muscles Alive: Their Functions Revealed by Electromyography,” Williams & Wilkins Company, 1985.

[2] Bida, Oljeta. “Influence of Electromyogram (EMG) Amplitude Processing in EMG-Torque Estimation.” WPI MS Thesis. (2005)

[3] Campellone, Joseph, M.D. .*MedLine Plus Medical Encyclopedia - Electromyography* (2004) http://www.nlm.nih.gov/medlineplus/ency/article/003929.htm

[4] Chan, Adrian and Englehart, Kevin. “Continuous Myoelectric Control for Powered Prosthesis using Hidden Markov Models.” IEEE Transactions on Biomedical Engineering, Vol. 52, No. 1, January 2005.

[5] Chan, Francis. “Fuzzy EMG Classification for Prosthesis Control.” IEEE Transactions of Rehabilitation Engineering. Vol 8, No 3, Sept 2000.

[6] Christini, David J., Kenneth M. Stein, Steven M. Markowitz, and Bruce B. Lerman. “Practical Real-Time Computing System for Biomedical Experiment Interface“ (1998).

[7] Clancy, Edward. “A PC-Based Workstation for Real-Time Acquisition, Processing, and Display of Electromyogram Signals.” Biomedical Instrumentation & Technology, 1998.

[8] Clancy, Edward A. and Farry, Kristin A., “Adaptive Whitening of the Electromyogram to Improve Amplitude Estimation”, IEEE Transactions on Biomedical Engineering, Vol. 47, No. 6, June 2000

[9] Clancy, E.A., Bouchard, S. and Rancourt, D., “Estimation and Application of EMG Amplitude During Dynamic Contractions,” IEEE Engineering in Medicine and Biology, Vol. 20, No. 6, Nov. – Dec. 2001.

[10] Clancy, E.A., Morin, E.L. and Merletti R., “Sampling, Noise-Reduction and Amplitude Estimation Issues in Surface Electromyography,” Journal of Electromyography and Kinesiology 12 (2002)

[11] Crawford, Beau, “Real-Time Classification of Electromyographic Signals for Robotic Control,” Technical Report No. 2005-03-05, Department of Computer Science, University of Washington, March 2005

[12] DeLuca, Carlo. "Surface Electromyography: Detection and Recording." DelSys Incorporated (2002).

[13] "Description of Performance Options in Windows." <http://support.microsoft.com/?kbid=259025>

[14] Duwairi, B. "Combined Scheduling of Hard and Soft-Real Time Tasks in Microprocessor Systems,"Department of Electrical and Computer Engineering, Iowa State University. <http://www.informatik.uni-trier.de/~ley/db/conf/hipc/hipc2003.html>

[15] Earnshaw, Garret. National Instruments Applications Engineer. Interview: 2/18/2006

[16] Englehart, Kevin. "A Robust Real-Time Control Scheme for Multifunction Myoelectric Control." IEEE Transactions of Biomedical Engineering, Vol. 50, No. 7, July 2003.

[17] Fuld, Matthew. "Design of a System for Data Acquistion and Computer Control of a Pulmonary Physiology Lab." <http://orchid.hosts.jhmi.edu/lunglab//CVs/mfuld_senior_design.pdf>

[18] "Getting Started With LabVIEW". Apr. 2003. <http://ni.com>.

[19] Jovanov, Emil, Pedro Gelabert, Reza Adhami, Bryan Wheelock, and Robert Adams. "Real Time Holter Monitoring of Biomedical Signals." (1999).

[20] "LabVIEW Analysis Concepts." March 2004. <http://ni.com>

[21] "LabVIEW User Manual." Apr. 2003. <http://ni.com>.

[22] Lyons, E.C., "Development of a Hybrid Hands-off Human Computer Interface Based On Electromyogram Signals and Eye-Gaze Tracking," 2001 Proceedings of the 23rd Annual EMBS International Conference, Oct. 25-28, Istanbul Turkey.

[23] Malik, Anna, Girlie Legacion, Roni Estein, and Saiful Huq. "Development of a Prosthetic Arm for Rehabilitation of Arm Muscles Using EMG Signals." (2005). <http://www.ee.umanitoba.ca/?

[24] NI 622X Specification, <http://www.ni.com/pdf/products/us/20044546301101dlr.pdf>

[25] Perry J, Bekey GA. "EMG-Force relationships in skeletal muscle." Critical Reviews in Biomedical Engineering, 1981.

[26] Salini, Tranquilli and Prakash. “Design of Instrumentation for Recording the Electromyogram.” MQP Project EXC-0302. 2003.

[27] Stallings, William. “Computer Organization & Architecture: Designing for Performance” Pearson Prentice Hall. Upper Saddle River NJ, 2006.

[28] Strom, J. “Real-Time Tracking and Modeling of Faces: An EKF Analysis by Synthesis Approach.” <http://www.cs.ucsb.edu/~cs281b/winter2002/papers/Strom.pdf>

[29] Tanenbaum, Andrew S.. Modern Operating Systems, Prentice Hall Inc.,Upper Saddle River NJ

[30] “User Manual: PCI-DAS6402/16.” <www.measurementcomputing.com>

[31] *“Using External Code In LabVIEW”*. April 2003 Edition. <http://www.ni.com>

[32] Wall, A.E. and Hidler, J.M., “Alterations in EMG Patterns During Robotic Assisted Walking,” IEEE Engineering in Medicine and Biology, April 2004.

[33] Xiang Feng. “Towards Real-time Enabled Microsoft Windows” Proceedings of the 5th ACM International Conference on Embedded Software. Pgs 142-146, 2005 <http://portal.acm.org/citation.cfm?id=1086228.1086256>

Appendix A: Compiling CIN Source Code

Complete the following steps to build CINs using the Visual C++ integrated development environment (IDE) on WINDOWS.

1. Launch Microsoft Visual Studio.
2. Select **File»New Project** to create a new DLL project.
3. Select **Win32 (from the Visual C++ menu)** as the project type. You can name your project whatever you want.
4. Select **An empty DLL project** when prompted to choose the type of DLL that you want to create and click **Finish**.
5. Select **Project»Add To Project»Files** and navigate to your `.c` file. Select your `.c` file and click the **Open** button to add the `.c` file to the project.
6. Select **Project»Add To Project»Files** to add CIN objects and libraries to the project. Select `cin.obj`, `labview.lib`, `lvsb.lib`, and `lvsbmain.def` from the `Cintools` subdirectory. You need these files to build a CIN.
7. Select **Project»Settings** and change **Settings For** to **All Configurations**. Click the **C/C++** tab and set the category to **Preprocessor**. Add the path to your `Cintools` directory in the **Additional include directories** field. Be sure to place the path to the directory in double quotations.
8. Select **Project»Settings** and change **Settings For** to **All Configurations**. Click the **C/C++** tab and set the category to **Code Generation**. Select the **Struct member alignment** tab and select **1 byte**.

9. Choose a run-time library. Select **Project»Settings** and change **Settings for** to **All Configurations**. Select the **C/C++** tab and set the category to **Code Generation**. Select **Multithreaded DLL** in the **Use run-time library** control.

10. Make a custom build command to run `lvsbutil`. Select **Project»Settings** and change **Settings for** to **All Configurations**. Select the **Custom Build** tab and change the **Commands** field as follows, with the code all on a single line:

```
<"your path to cintools">\lvsbutil "$(TargetName)" -d
"$(ProjectDir)\$(OutDir)"
Note that the path to cintools must be placed in double
quotations ("")
```

Change the **Output** fields to `$(OutDir)$(TargetName).lsb`.

11. Add `<CINTOOLSDIR>\lvsbmain.def` to the **Linker»Input»Module Definition File** field.

12. Click the **File View** tab in the **Work Space** window.

13. Open your `.c` file and replace **`/* Insert code here */`** with your code.

14. Select **Build»Build *projectName*.dll**, where ***projectName*** is the name of your project.

Appendix B: C Code for Choosing An Array of Filter Coefficients

Below is the C code that, when given a matrix of filter coefficients, will output one of the coefficient vectors depending on the scaled amplitude that is given as input.

```
/* CIN source file */

#include "extcode.h"

#define typeCode 0x0A
#define numDims 2
#define paramNum 2

/* Typedefs */

typedef struct {
      int32 dimSizes[2];
      float64 elt[1];
      } TD1;
typedef TD1 **TD1Hdl;

typedef struct {
      int32 dimSize;
      float64 Numeric[1];
      } TD2;
typedef TD2 **TD2Hdl;

/* Macros */


MgErr CINRun(TD1Hdl InputArray, float64 *NormalizedValue, TD2Hdl
Array2);

MgErr CINRun(TD1Hdl InputArray, float64 *NormalizedValue, TD2Hdl
Array2)
      {
      int32 index;
                  // Index of the filter array (selects which row to
use)
      int32 i;
      MgErr err = noErr;
      float64 *result;
      /* Insert code here */
      int32 numOfFilters = (*InputArray)->dimSizes[0];
      // Number of Rows in the array = the number of filters defined
      int32 numOfTaps = (*InputArray)->dimSizes[1];
      // Number of Columns in the array = number of filter taps

      if (err = SetCINArraySize((UHandle)Array2, paramNum, numOfTaps))
            {return err;}
      (*Array2)->dimSize = numOfTaps;

      //index = (int)(*NormalizedValue);
      index = (int32)((*NormalizedValue) * numOfFilters);
      if (index < 0) {index = 0; }
      if (index > (numOfFilters - 1)) {index = (numOfFilters - 1); }

      result = (*Array2)->Numeric;
      *result = 0;

      for (i = 0; i < numOfTaps; i++)
            {
                  *result = (*InputArray)->elt[(index*numOfTaps) + i];
                  result++;
            }

      return noErr;
      }
```

Appendix C: MATLAB Code to Simulate Adaptive Whitening Filter Parameters

Below is a MATLAB M-file that was created to simulate the parameters that need to be entered into Ted Clancy’s M-file to create a matrix of filter coefficients for adaptive whitening.

```matlab
function [mvc0, mvc50] = cal_sim(lnth,rt)
%lnth - length of sample in samples
%rt - sampling rate in Hz

randn('state',sum(100*clock)); %states random number generator
mvc0 = 0.03 * randn(1,lnth);
[b,a] = butter(2,200 * 2/rt);
mvc50 = 0.5 * randn(1,lnth + rt);
mvc50 = filter(b,a,mvc50);
mvc50 = mvc50(end-lnth+1:end) + 0.03 * randn(1,lnth);
return
```

Appendix D: MATLAB Code that Creates Adaptive Whitening Filter Coefficients

Below is the MATLAB M-file, created previously by Ted Clancy, that creates a matrix of filter coefficients for adaptive whitening.

.

```
function [B, sCalVec] = e_cal_wh(SandN, sCal, Noise, varargin)
%E_CAL_WH Design/calibrate adaptive and fixed whitening filters.
%
% [B, sCalVec] = e_cal_wh(SandN, sCal, Noise[, Order][, sSpec][,
Option-Pairs])
%
% Design a set of FIR adaptive whitening filters based on the three-
% stage cascade technique of Clancy and Farry (In Preparation).
%
% SandN: Calibration sample of signal plus noise (vector).
% sCal:  MVE level corresponding to SandN (scalor).
%        (Typically 0.5 for 50% MVC.)
% Noise: Calibration sample of noise (vector).
%
% Options ---
% Order: Order of the combined filter (# coefficients - 1) (scalor).
%        (Default = 60.)
% sSpec: EMG amplitude level specifier (same scale as sCal) (vector).
%        (Default specifies 100 filters from just above 0 to 2*sCal.)
%
% Option-Pairs ---
% 'Nfft', NNfft:  FFT length (power of 2) for spectral estimates
(scalor).
% 'MaxGain', MMaxGain: Maximum design gain at any frequency (scalor).
%
% B: Each row contains the moving average coefficients of the
%    complete adaptive whitener corresponding to the EMG amplitude
%    found in the corresponding element of sCalVec.
% sCalVec: Equally spaced vector of EMG amplitudes specifying the
amplitudes
%    for which the filters in B are calibrated.
%
%   EMG Amplitude Estimation Toolbox - Ted Clancy - WPI

%   April 9, 1999
%   Updated 23 January 2004.

%%%%%%%%%%%%%%%%%%%%% Process Command Arguments %%%%%%%%%%%%%%%%%%%%%

% Check number of input/output arguments
ArgsIn  = nargin;
ArgsVar = length(varargin);
if (ArgsIn < 3  |  ArgsIn > 9)
```

```
  error(['Bogus number of arguments (' int2str(ArgsIn) '), expected 3-
9.']);
end

% Look for option-pairs.
NNfft = 256;      % Default FFT length (power of 2).
MMaxGain = 100;   % Default maximum fixed whitening gain.
FlagNfft = 0;     % 0 denotes option not yet used: 1==> used.
FlagMax  = 0;     % 0 denotes option not yet used: 1==> used.
NotDone = 1;
while (ArgsVar >= 2)
  if isstr(varargin{ArgsVar-1})
    switch varargin{ArgsVar-1}
      case 'Nfft'
        if (FlagNfft > 0 ), error('Repeated "Nfft" selection.'); end
        FlagNfft = 1;
        if isempty(varargin{ArgsVar})==0, NNfft = varargin{ArgsVar};
end
        if length(NNfft)>1, error('NNfft not a scalor.'); end
        if (2^nextpow2(NNfft))~=NNfft, error('NNfft not a power of
2.'); end
      case 'MaxGain'
        if (FlagMax > 0 ), error('Repeated "MaxGain" selection.'); end
        FlagMax = 1;
        if isempty(varargin{ArgsVar})==0, MMaxGain = varargin{ArgsVar};
end
        if length(MMaxGain)>1, error('MMaxGain not a scalor.'); end
        if MMaxGain <= 0.0, error('MMaxGain must be non-negative.');
end
      otherwise
        error(['Bogus option "' varargin{ArgsVar-1} '".']);
    end
  else
    break;
  end
  ArgsIn  = ArgsIn  - 2;
  ArgsVar = ArgsVar - 2;
end

% Required arguments and regular optional arguments.
if (ArgsIn < 3  |  ArgsIn > 5)
  error(['Bogus number of non-option-pair arguments (' int2str(ArgsIn)
'), expected 3-5.']);
end

if isstr(SandN), error('First argument "SandN" not numeric.'); end
SizeRes = size(SandN);
if ( length(SizeRes)>2 | (SizeRes(1)~=1 & SizeRes(2)~=1) )
  error('First argument "SandN" not a vector.');
end
if (length(SandN)<2*NNfft), error('First argument length too short.');
end

if isstr(sCal), error('Second argument "sCal" not numeric.'); end
if length(sCal)>1, error('Second argument "sCal" not a scalor.'); end
if sCal<=0.0, error('Second argument "sCal" must be non-negative.');
end

if isstr(Noise), error('Third argument "Noise" not numeric.'); end
SizeRes = size(Noise);
if ( length(SizeRes)>2 | (SizeRes(1)~=1 & SizeRes(2)~=1) )
  error('Third argument "Noise" not a vector.');
end
if (length(Noise)<2*NNfft), error('Third argument length too short.');
end

Order = 60;  % Default.
if ArgsVar > 0
  if isempty(varargin{1})==0, Order = varargin{1}; end
  if isstr(Order), error('Fourth argument "Order" not numeric.'); end
  if length(Order)>1, error('Fourth argument "Order" not a scalor.');
end
  if Order<=0, error('Fourth argument "Order" must be non-negative.');
end
end

sCalVec = 2*sCal/100 : 2*sCal/100 : 2*sCal;  % Default.
if length(sCalVec)~=100, error('Internal "sCalVec" length error.'); end
if ArgsVar > 1  &  isempty(varargin{2}) == 0
  sSpec = varargin{2};
  if isstr(sSpec), error('Fifth argument "sSpec" not numeric.'); end
  SizeRes = size(sSpec);
  if ( length(SizeRes)>2 | (SizeRes(1)~=1 & SizeRes(2)~=1) )
    error('Fifth argument "sSpec" not a vector.');
  end
  if length(sSpec)~=3, error('Fifth argument "sSpec" not length 3.');
end
  if sSpec(1)<=0.0, error('sSpec(1) less than or equal to zero.'); end
  if sSpec(2)<=0.0, error('sSpec(2) less than or equal to zero.'); end
  if sSpec(3)<sSpec(1), error('sSpec(3) less than sSpec(1).'); end
  sCalVec = sSpec(1) : sSpec(2) : sSpec(3);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Compute %%%%%%%%%%%%%%%%%%%%%%%%%

% Compute magnitude response of the fixed whitener.
%   Estimate PSD's of SandN and Noise.  Also get frequency bins.
[Svv, f] = psd(Noise, NNfft, 2, NNfft, round(0.75*NNfft));
Smm_sCal = psd(SandN, NNfft, 2, NNfft, round(0.75*NNfft));
%   Estimate PSD of JUST signal.
Snn = max(Smm_sCal - Svv, eps) / (sCal * sCal);
Snn = ClearEPS(Snn);  % Interpolate over the eps values.
%   Fixed whitener is 1./SQRT of power spectrum of JUST signal.
Htime = min( 1 ./ sqrt(Snn), MMaxGain);  % Assign zero phase.

% Iteratively compute Wiener filter, gain, and FIR filters.
Svv_tilda = Svv .* Htime .* Htime;  % PSD of noise after fixed
whitener.
for i = 1:length(sCalVec)
  Hwiener = sCalVec(i)*sCalVec(i) ./ ( sCalVec(i)*sCalVec(i) +
Svv_tilda );
```

```
  Htw = Htime .* Hwiener;  % Cascade Htime, Hwiener (both real).
  B(i,:) = fir2( Order, f, Htw );  % First pass, no gain scaling.
  AchievedH = abs(freqz(B(i,:), 1, NNfft/2+1));  % Achieved filter
magnitude.
  Sinput = sCalVec(i)*sCalVec(i)*Snn + Svv;  % Input PSD at this EMG
amplitude.
  d = sqrt( mean(Sinput) ./ mean(Sinput.*AchievedH.*AchievedH) );
  B(i,:) = B(i,:) * d;  % Proper gain scaling.
end

return;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ClearEPS %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The minimum value in input Snn is found in all locations where an eps
has
% been inserted.  This function replaces this value with a smooth
% interpolation, so that the ensuing inverse does not "blow up".
%   Originally writen by Punit Prakash, 2004.
function Snn = ClearEPS(Snn)
eps1 = find(Snn == min(Snn));   % find locations where minimum value
occurs in noise
minval = min(Snn);              % store minimum value

if (eps1(1) == 1)
  first = find(Snn >= minval); % if first value in Snn is minval,
replace with first non-minval value in Snn
  Snn(1) = first(1);
end
idx=length(Snn);

if (eps1(end) == length(Snn))   % if last value in Snn is minval
  while (Snn(idx) == minval), idx =  idx-1; end          % find last
"good value"
  num_eps = length(Snn) - idx;
  for p = 1:(num_eps-1)
    Snn(idx+p) = Snn(idx) - (Snn(idx)/num_eps)*p;  % Interpolate
between last good point and 0
  end
end

for z = 1:length(eps1)
  x = eps1(z);    % Get index of location of minval
  a = x-1;        % Get index of point ahead
  b = x+1;        % Get index of point before
  if (a ~= 0 && b~=length(Snn) && b~=length(Snn)+1)
    if((Snn(a) ~= minval) && (Snn(b) ~= minval))
      Snn(x) = mean([Snn(a) Snn(b)]); % interpolate if Snn(a) and
Snn(b) are "good" values
    else
      if (b ~= length(Snn)-1)
        while (a~= 0 && Snn(a) == minval && a~=1), a = a - 1; end  %
find last good value
        while (Snn(b) == minval)
```

```
          if (b == length(Snn) - 1)   % If last value is minval, then
break
            break;
          else
            b = b + 1;                % loop to find next value
          end
        end
        temp = Snn(b) - Snn(a);           % interpolation
        num_eps = b - a;
        for p = 1:(num_eps-1), Snn(a+p) = Snn(a) + (temp/num_eps)*p;
end
      end
    end
  end
end

return;
```

Appendix E: MATLAB Script Node Latency Test

As stated in the main body of the report, it was originally desired to use MATLAB code to implement the gain normalization step of the adaptive whitening procedure. However, results indicate that MATLAB script nodes produce large latencies and, as a result, are not desirable in low latency applications. Figure EA.1 depicts a test that was done to indicate the large latencies associated with MATLAB script nodes. In this test, two real numbers were added together. The amount of time taking to execute the addition operation was then recorded one hundred times at three different Windows priority levels: Normal, High and Real-time. The results of these tests, which are displayed in this Appendix, show that in the best case operation under real-time priority, the latency induced by MATLAB was an average of 8.53 msec.

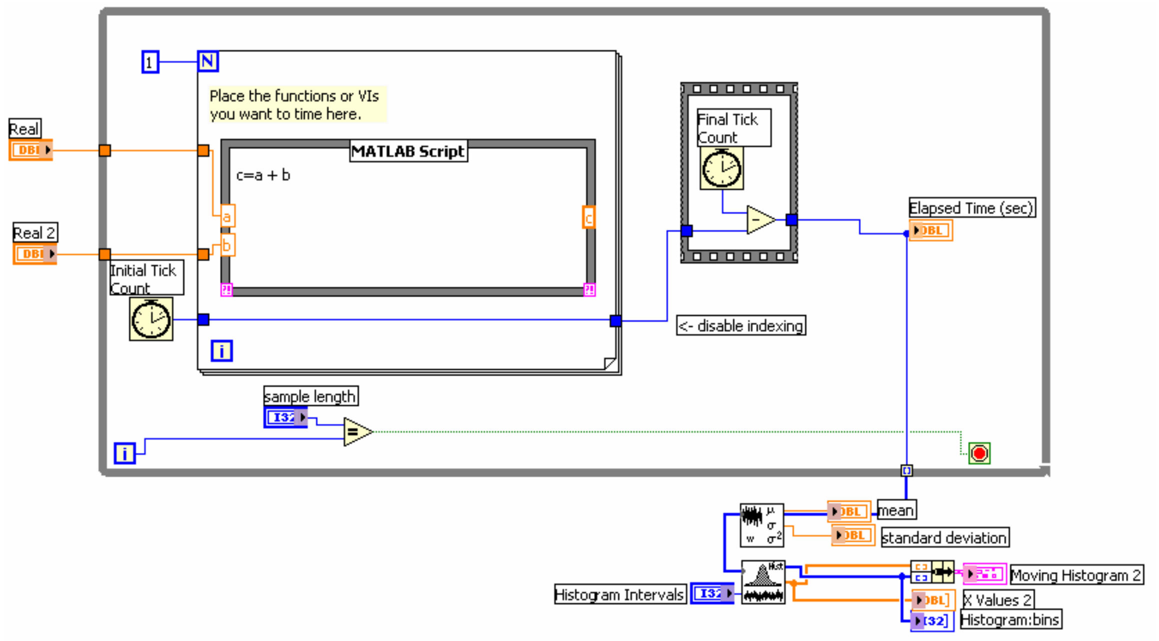
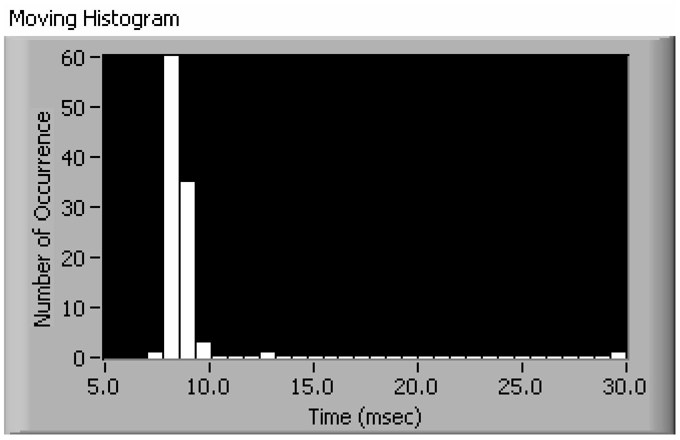


Figure EA.1: VI to Test Latency of MATLAB Script Node Adding Two Real Numbers

Raw Data Collected

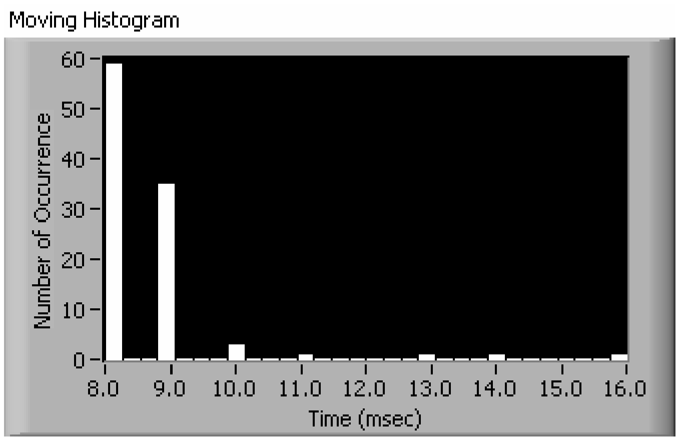
The following histograms display the raw data collected from our experiment. Each test was executed 100 times for three Windows XP priority levels: Normal, High and Real-Time. Histograms displaying the latency recorded for each experiment can be seen below.

Priority = Normal



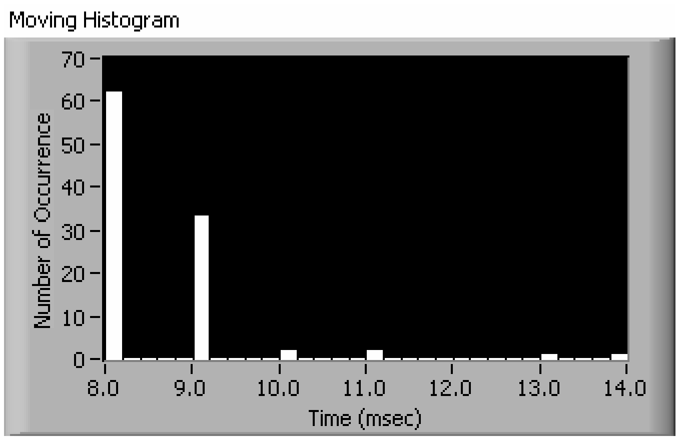
Mean = 8.66 msec
Standard Deviation = 2.26 msec

Priority = High



Mean = 8.62 msec
Standard Deviation =1.19 msec

Priority = Real Time



Mean = 8.53 msec
Standard Deviation = .95 msec

Appendix F: Latency Test Results for EMG Algorithm

In Chapter 5 of the report, information regarding how each EMG amplitude estimation algorithm (each utilizing one the three different LabVIEW analysis methods) was tested to determine which algorithm had the least latency. While summary plots were presented, this Appendix contains all of the raw data collected. Each algorithm implementing a different analysis method was respectively tested at the Normal, High and Real-time Windows XP priorities using 512, 1024, 1536 and 2048 highpass, smoothing and whitening filter coefficients. Details of the parameters set are shown with the histograms below. Furthermore, it should be noted that the data that were collected to test these functions was first obtained from the PCI-6229 and then stored in the buffer. Then, once this data was stored, the testing began.

Raw Data

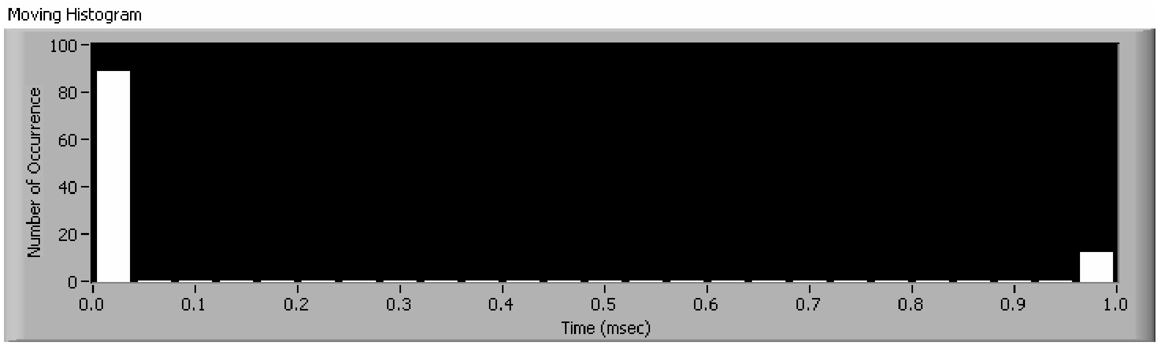
Analysis Method: Point-by-Point
Iteration = 100
Intervals = 25
Channels = 2
MVC = .6
# Smoothing Filter Coefficients = #High-Pass Filter Coefficients = # Whitening Coefficients

---

Sampling Frequency = 500 Hz

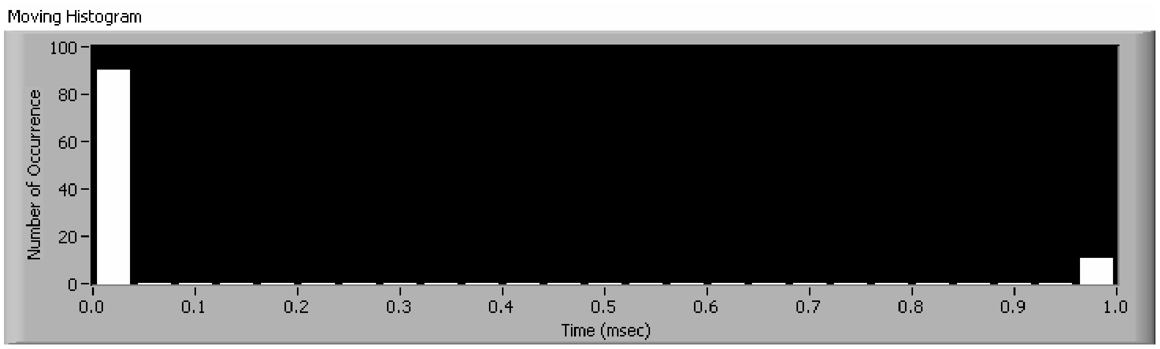
Whitening Filter Coefficients = 512

Priority = Normal



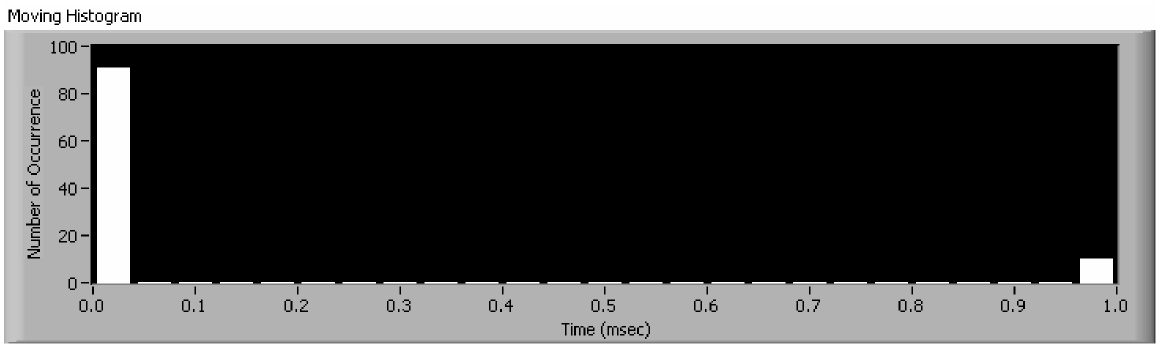
Mean = 0.12 (msec)
Standard Deviation = 0.33 (msec)

Priority = High



Mean = 0.11 (msec)
Standard Deviation = 0.31 (msec)

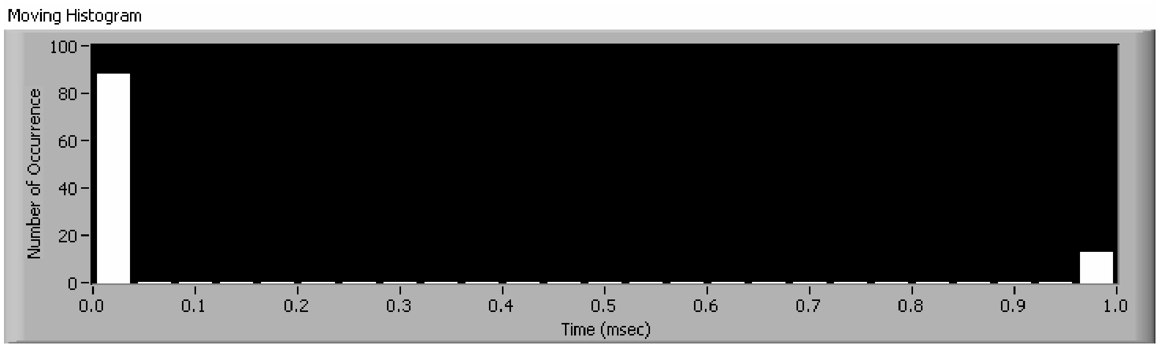
Priority = Real Time



Mean = 0.10 (msec)
Standard Deviation = 0.30 (msec)

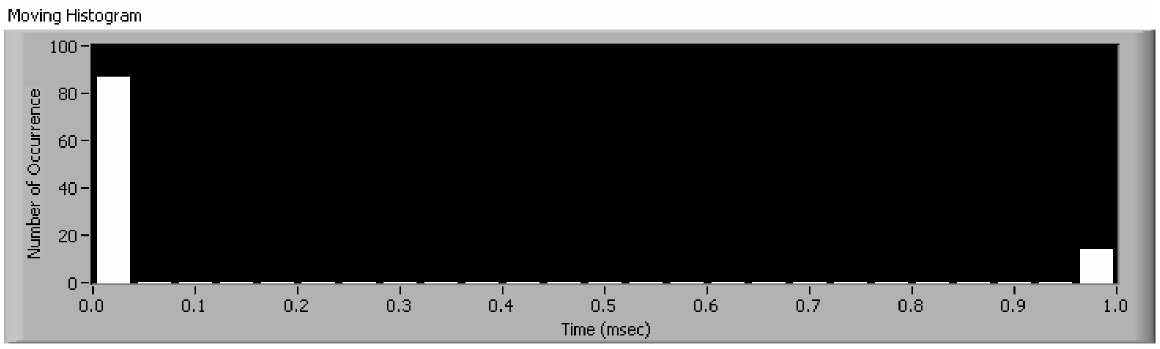
Whitening Filter Coefficients = 1024

Priority = Normal



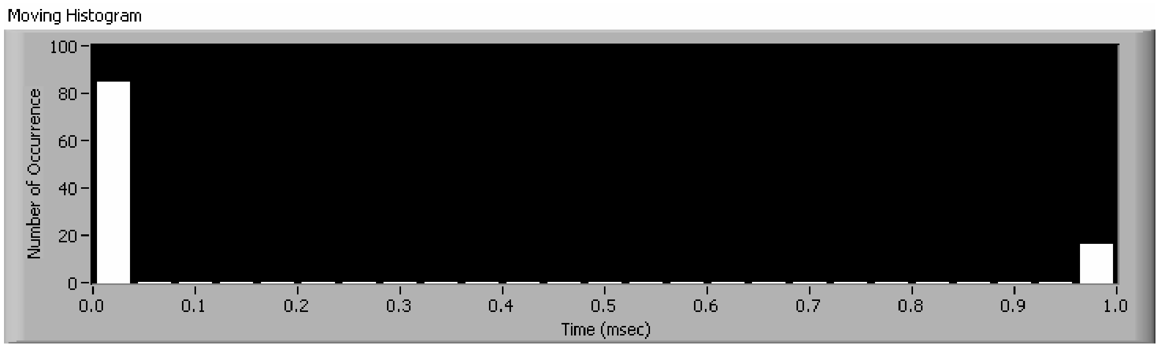
Mean = 0.13 (msec)
Standard Deviation = 0.34 (msec)

Priority = High



Mean = 0.14 (msec)
Standard Deviation = 0.35 (msec)

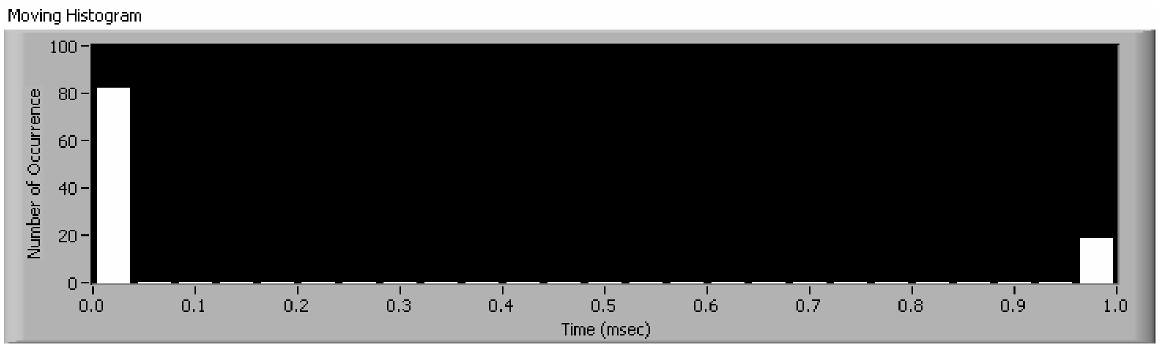
Priority = Real-time



Mean = 0.16 (msec)
Standard Deviation = 0.37 (msec)

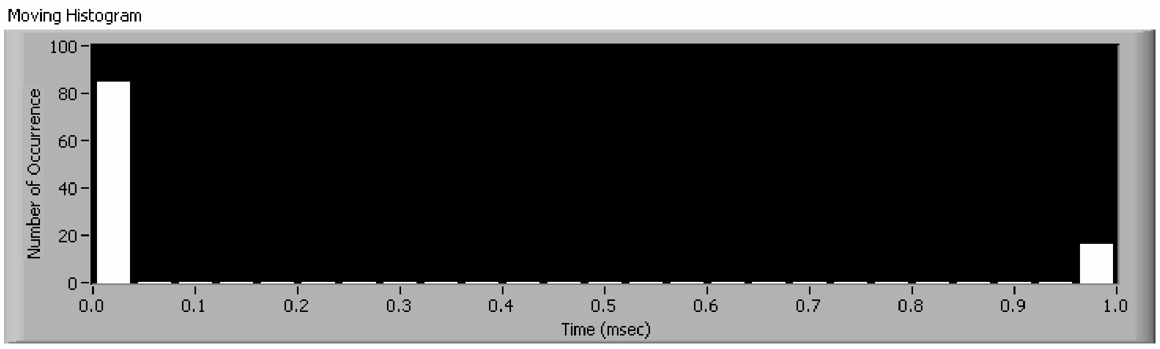
Whitening Filter Coefficients = 1536

Priority = Normal



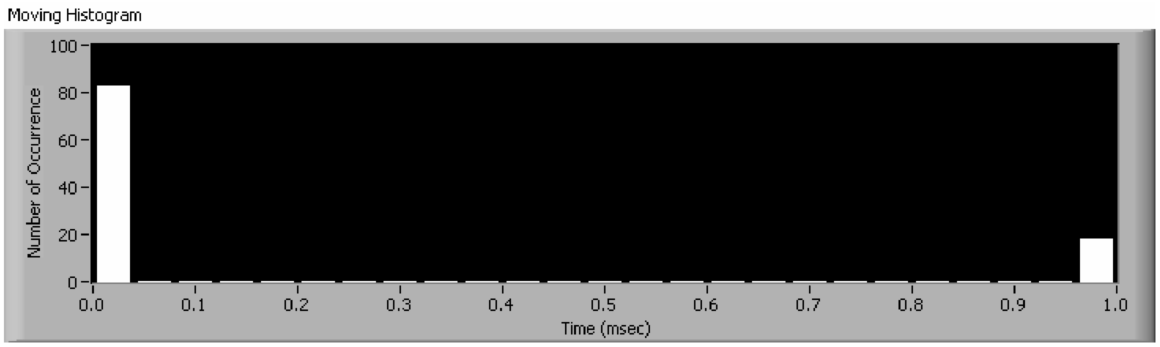
Mean = 0.19 (msec)
Standard Deviation = 0.39 (msec)

Priority = High



Mean = 0.16 (msec)
Standard Deviation = 0.37 (msec)

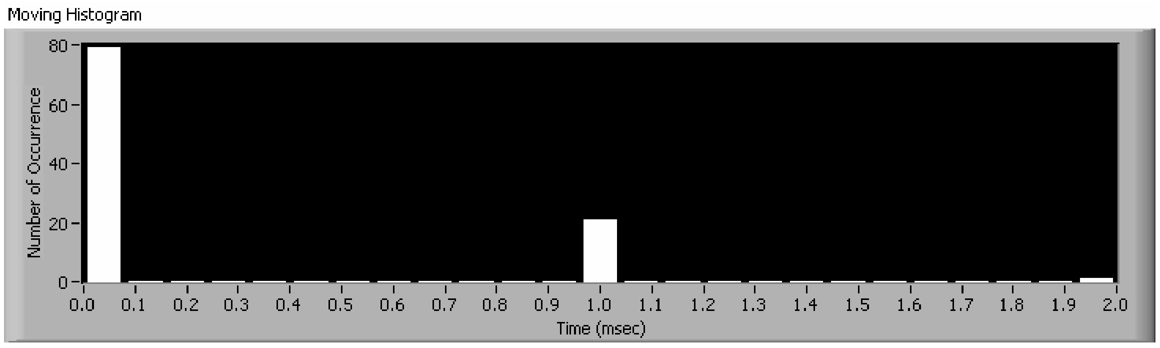
Priority = Real-time



Mean = 0.18 (msec)
Std Dev = 0.38 (msec)

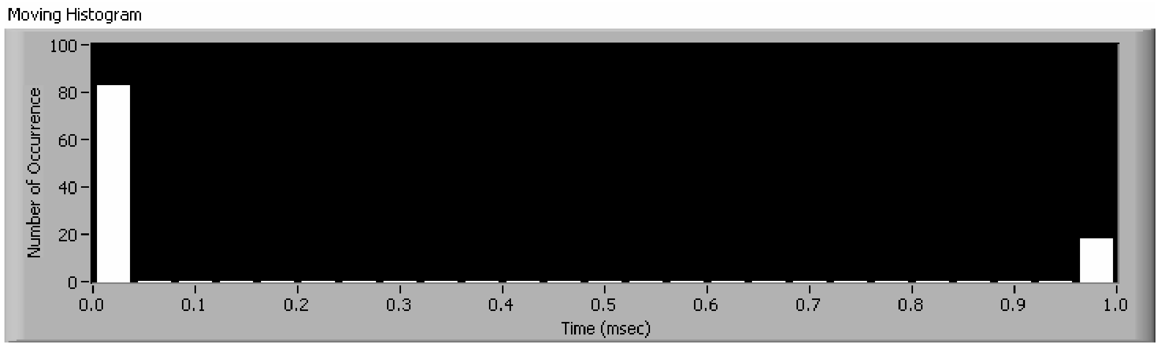
Whitening Filter Coefficients = 2048

Priority = Normal



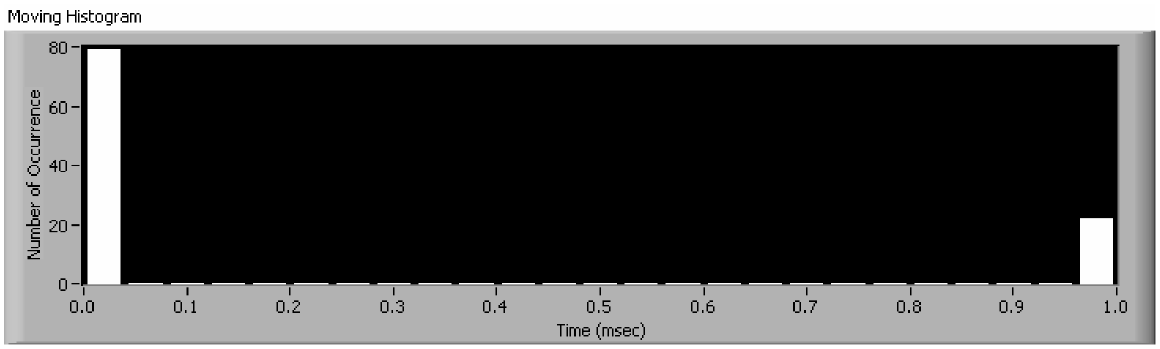
Mean = 0.23 (msec)
Standard Deviation = 0.44 (msec)

Priority = High



Mean = 0.18 (msec)
Standard Deviation = 0.38 (msec)

Priority = Real-time



Mean = 0.22 (msec)
Standard Deviation = 0.41 (msec)

Iteration = 100
Intervals = 25
Channels = 2
Array Based Method

# Smoothing Filter Coefficients = #High-Pass Filter Coefficients = # Whitening Coefficients

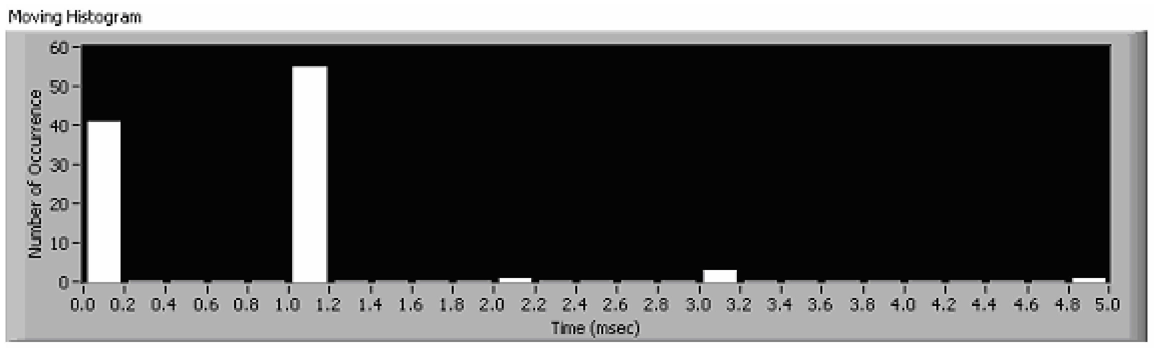
----

Sampling Frequency = 500 Hz

Number of Samples = 1

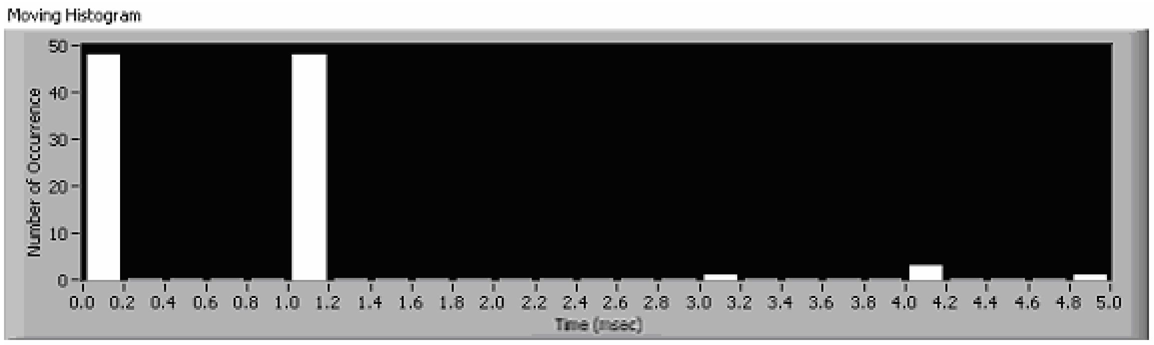
Whitening Filter Coefficients = 512

Priority = Normal



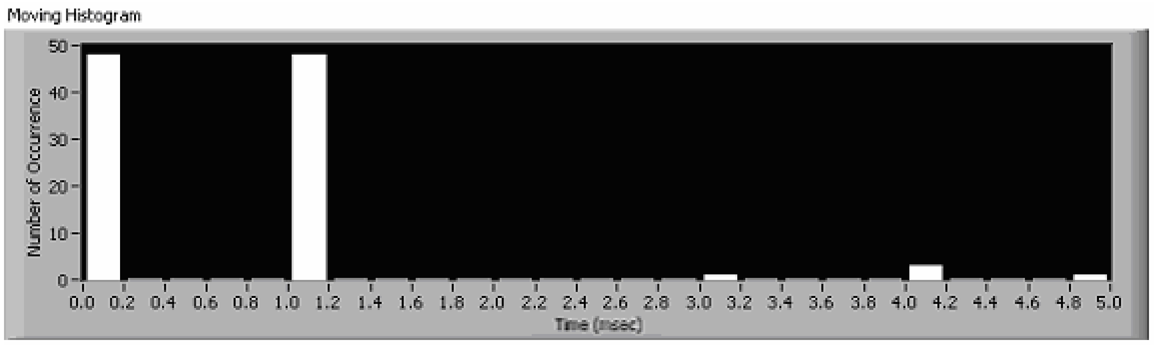
Mean = .70 msec
Standard Deviation = .78 msec

Priority = High



Mean =.61 msec
Standard Deviation = .79 msec

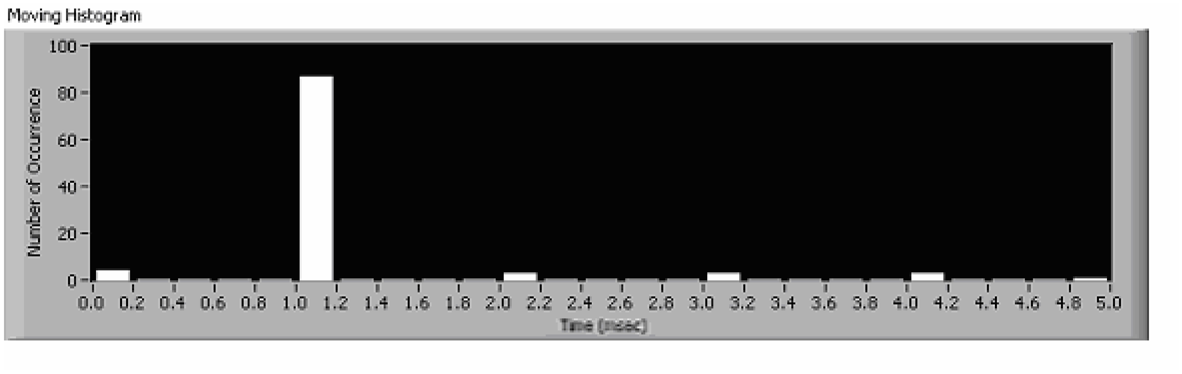
Priority = Real Time



Mean = .67 msec
Standard Deviation = .92 msec

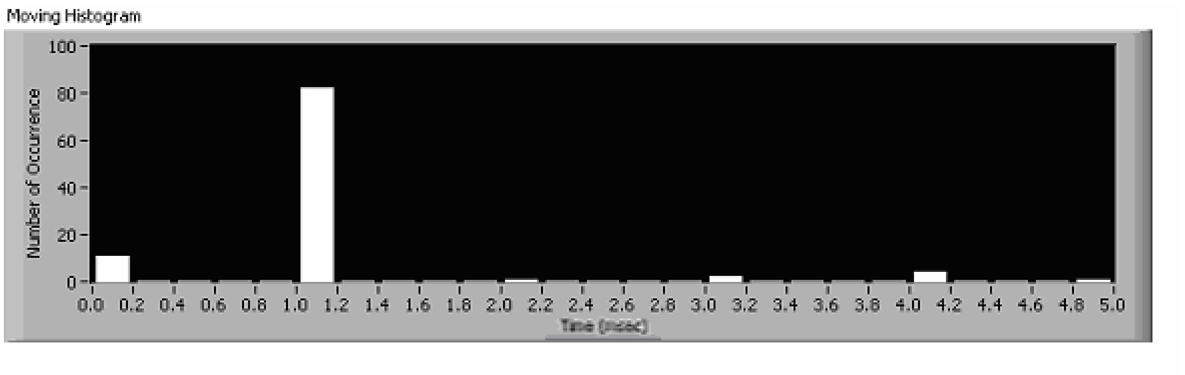
Whitening Filter Coefficients = 1024

Priority = Normal



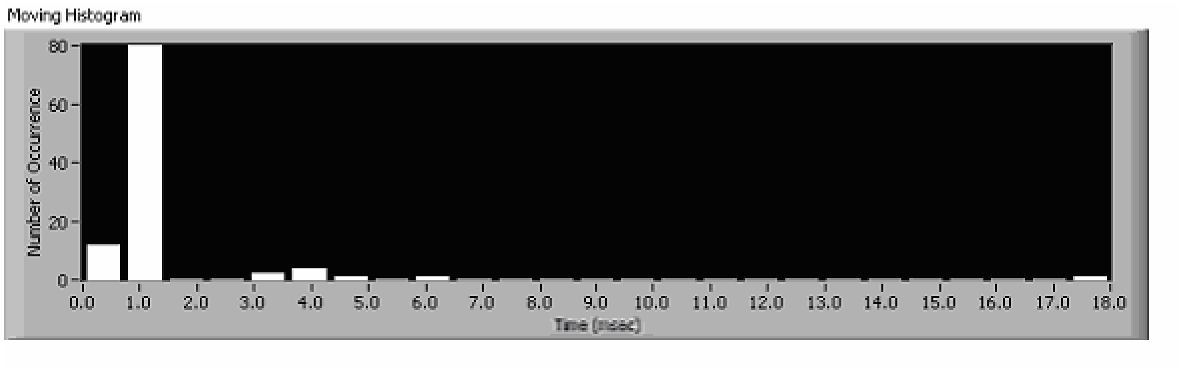
Mean = 1.18 msec
Standard Deviation = .77 msec

Priority = High



Mean =1.10 msec
Standard Deviation = .84 msec

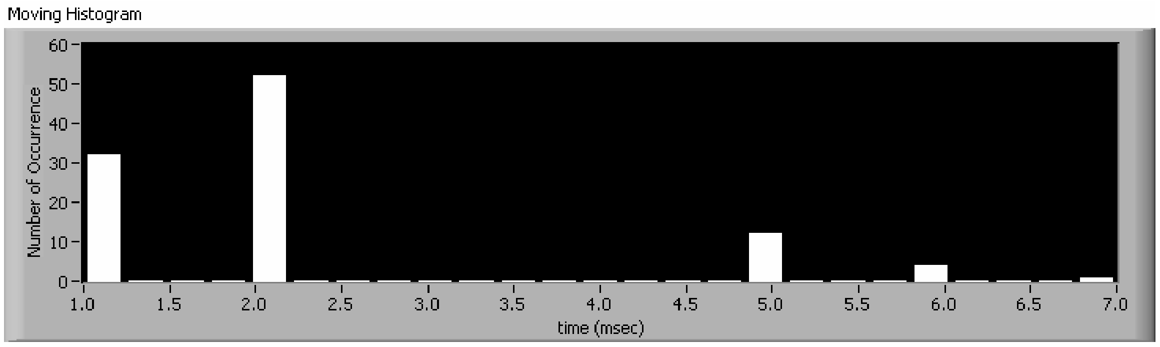
Priority = Real-Time



Mean = 1.30 msec
Standard Deviation = 1.94 msec

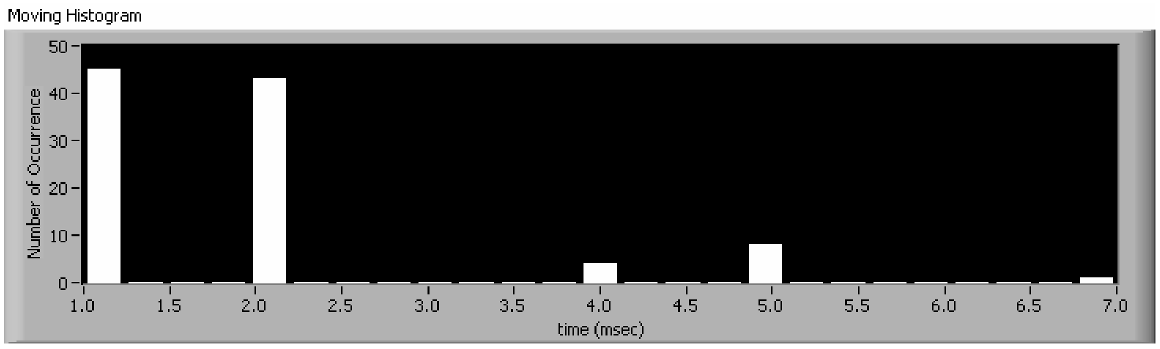
Whitening Filter Coefficients = 1536

Priority = Normal



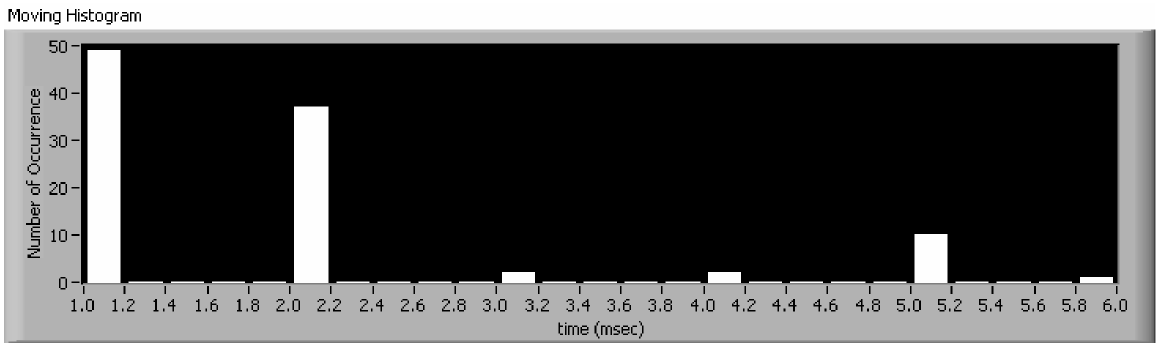
Mean = 2.25 msec
Std Dev = 1.49 msec

Priority = High



Mean = 1.92 msec
Standard Deviation = 1.25 msec

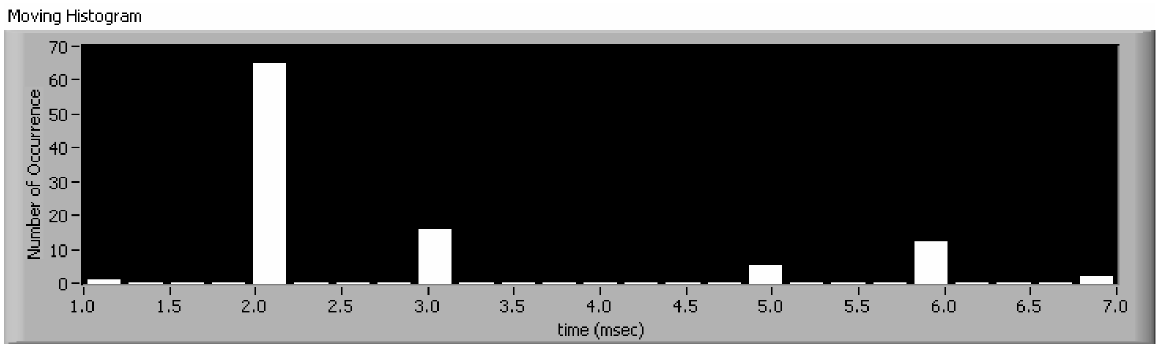
Priority = Real-Time



Mean = 1.91 msec
Standard Deviation = 1.28 msec

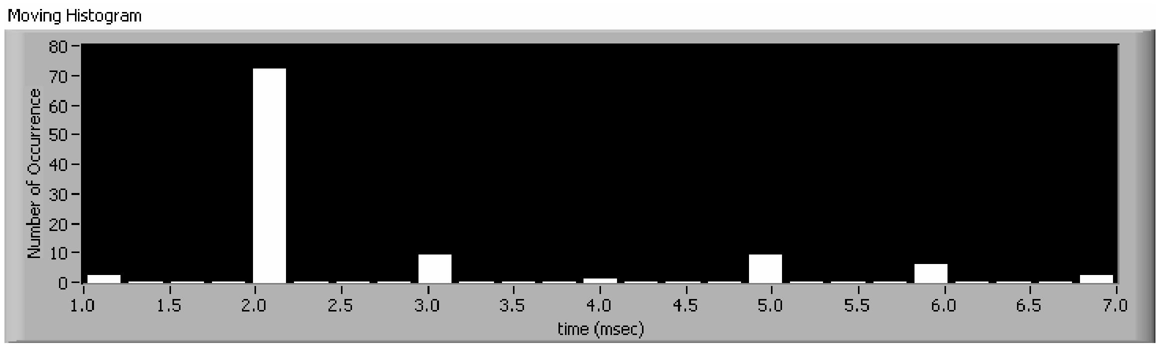
Whitening Filter Coefficients = 2048

Priority = Normal



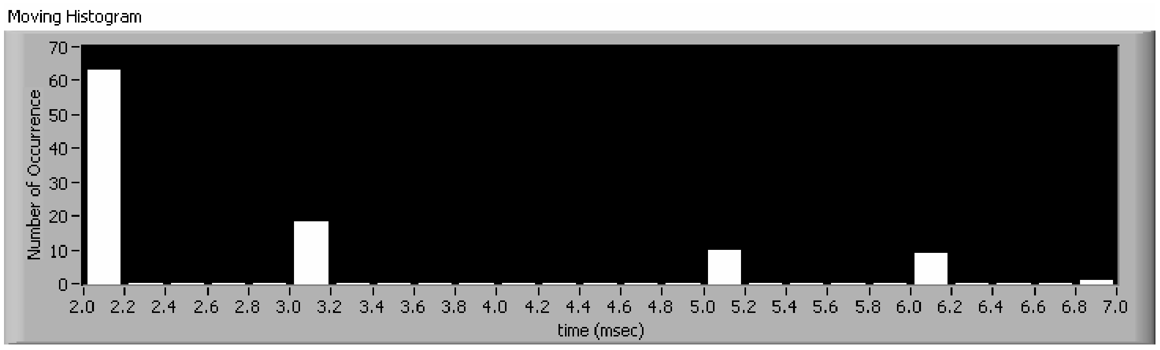
Mean = 2.87
Standard Deviation = 1.51

Priority = High



Mean = 2.69 msec
Standard Deviation = 1.39 msec

Priority = Real-time

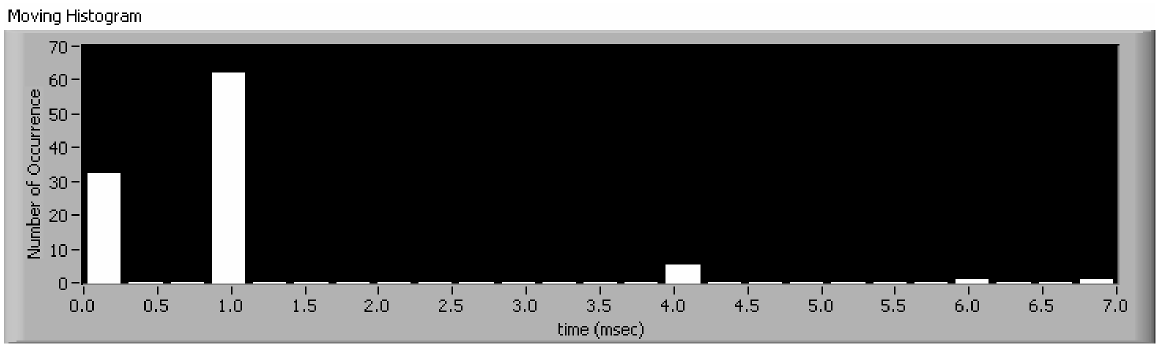


Mean = 2.88 msec
Standard Deviation = 1.41 msec

Sampling Frequency = 500 Hz
Number of Samples = 10

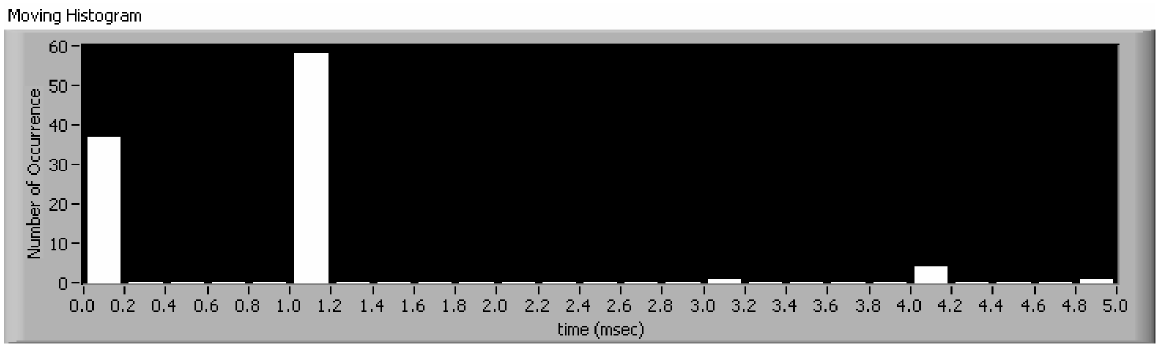
Whitening Filter Coefficients = 512

Priority = Normal



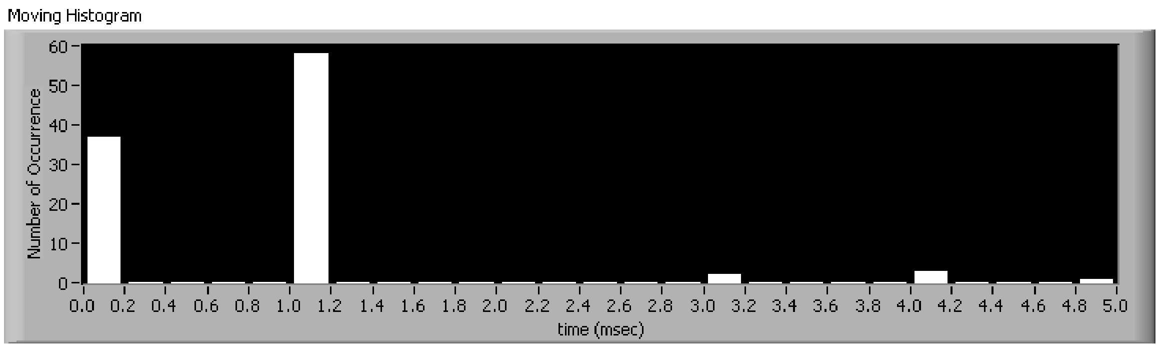
Mean = 0.94 msec
Standard Deviation = 1.17 msec

Priority = High



Mean = 0.81 msec
Standard Deviation = 0.95 msec

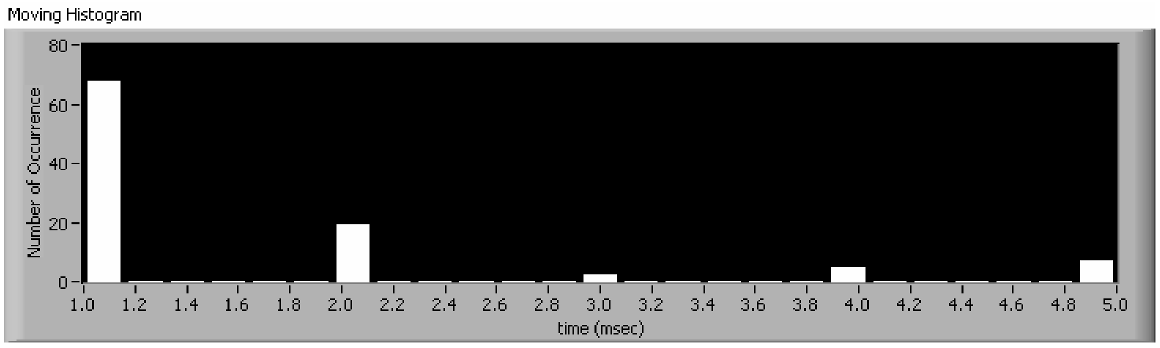
Priority = Real-Time



Mean = 0.80 msec
Standard Deviation = 0.92 msec

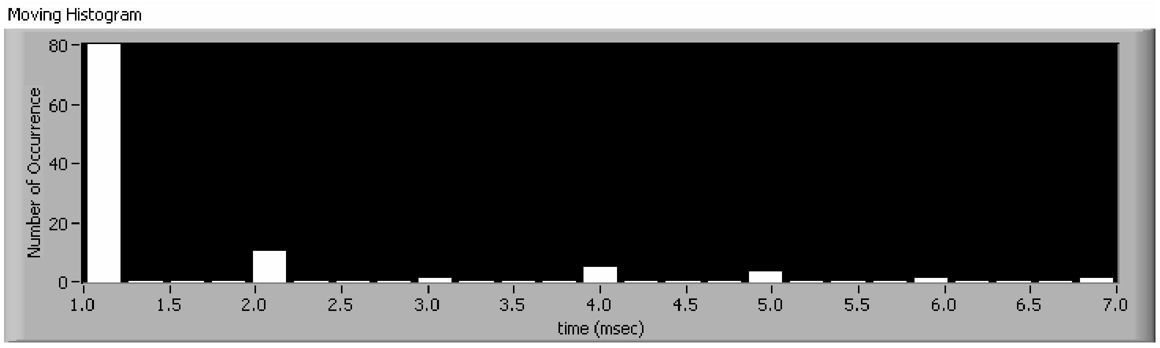
Whitening Filter Coefficients = 1024

Priority = Normal



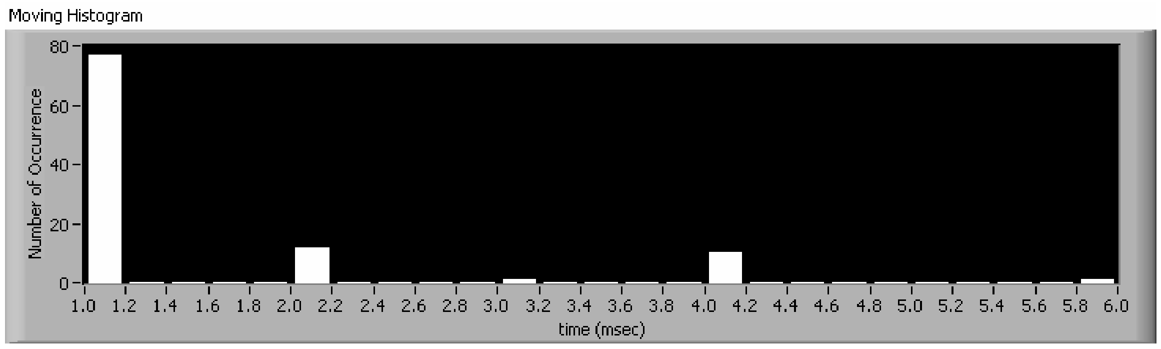
Mean = 1.65 msec
Standard Deviation = 1.19 msec

Priority = High



Mean = 1.50 msec
Standard Deviation = 1.20 msec

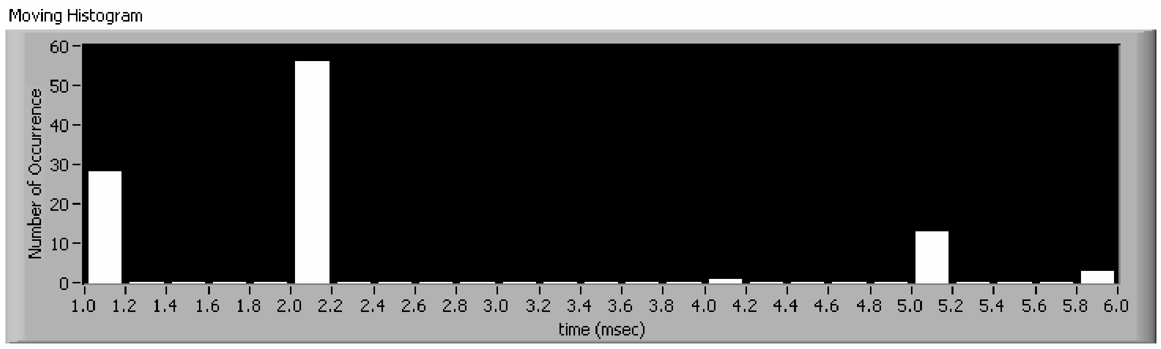
Priority = Real-time



Mean = 1.48 msec
Standard Deviation = 1.04 msec

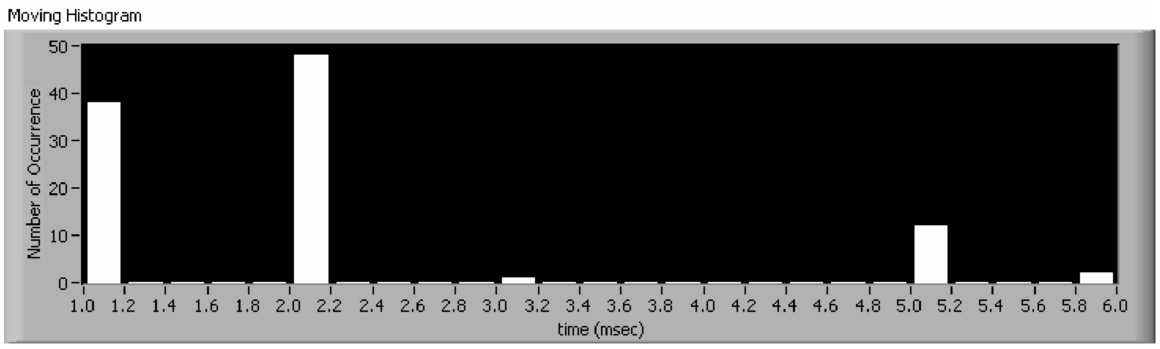
Whitening Filter Coefficients = 1536

Priority = Normal



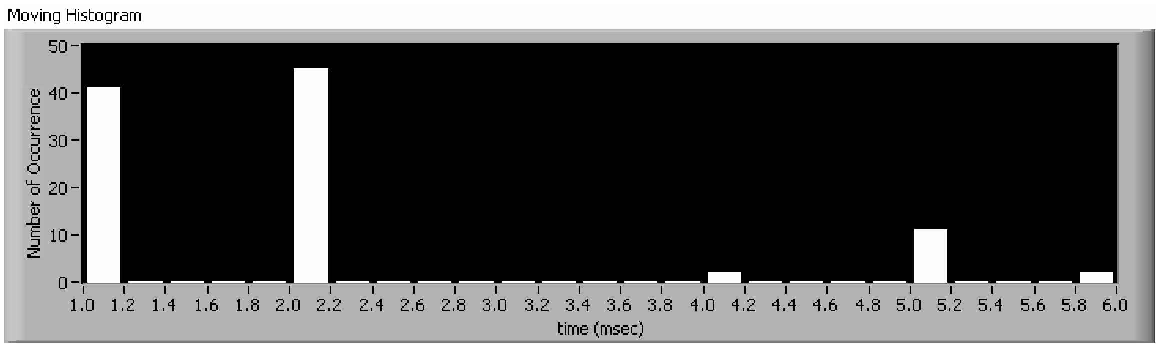
Mean = 2.25 msec
Std Dev = 1.38 msec

Priority = High



Mean = 2.07 msec
Standard Deviation = 1.34 msec

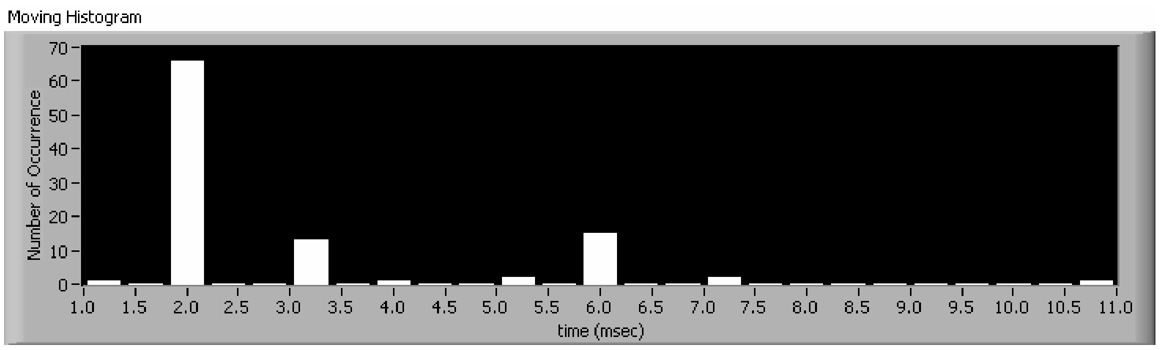
Priority = Real Time



Mean = 2.04 msec
Standard Deviation = 1.34 msec

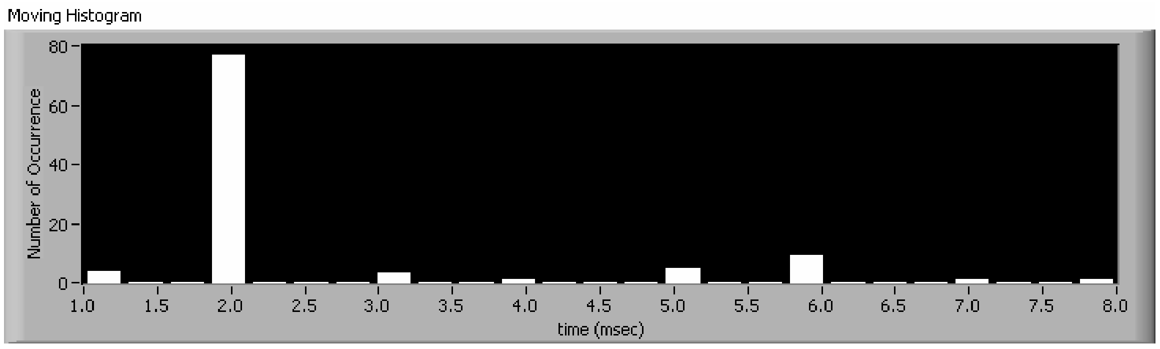
Whitening Filter Coefficients = 2048

Priority = Normal



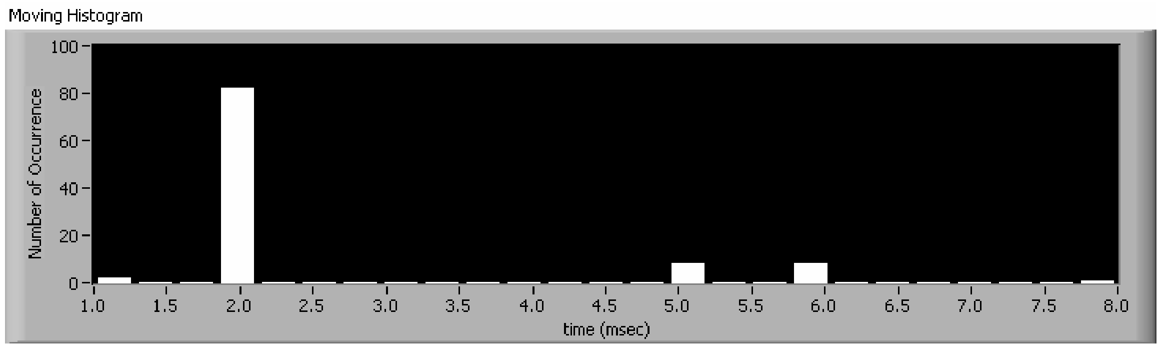
Mean = 2.96 msec
Standard Deviation = 1.76 msec

Priority = High



Mean = 2.62 msec
Standard Deviation = 1.49 msec

Priority = Real Time



Mean = 2.59 msec
Standard Deviation = 1.42 msec

Iteration = 100
Intervals = 25
Channels = 2
Express Method

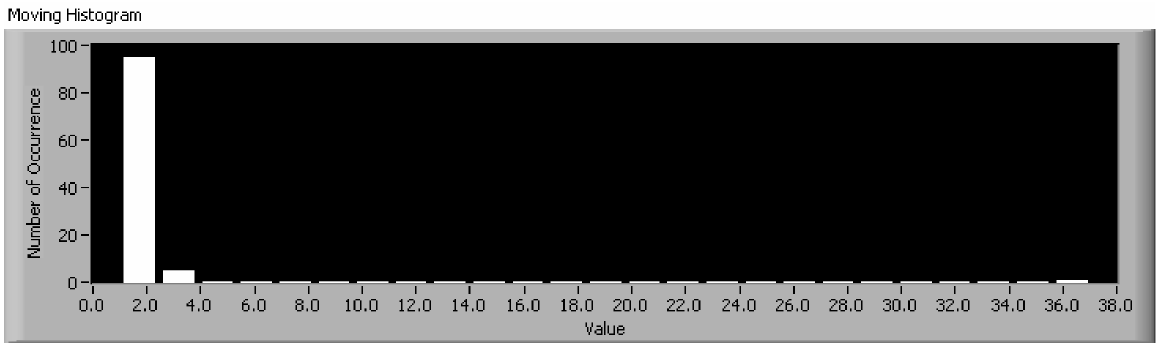
# Smoothing Filter Coefficients = #High-Pass Filter Coefficients = # Whitening Coefficients

----

Sampling Frequency = 500 Hz
Number of Samples = 1

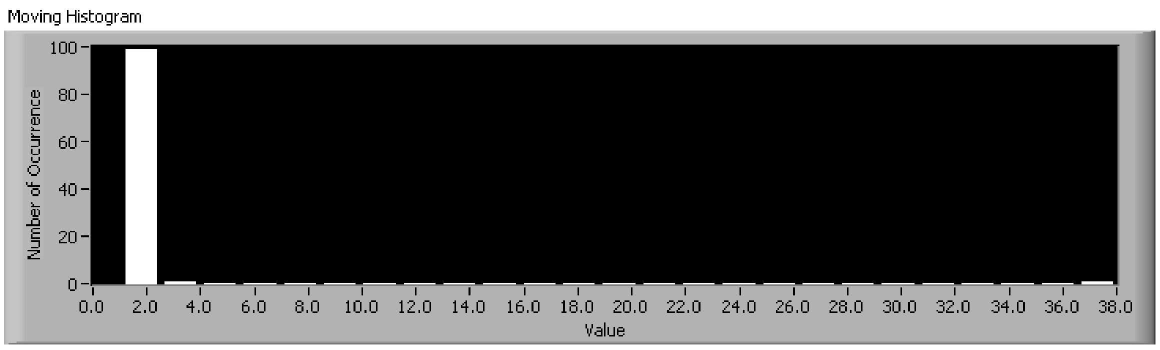
Whitening Filter Coefficients = 512

Priority = Normal



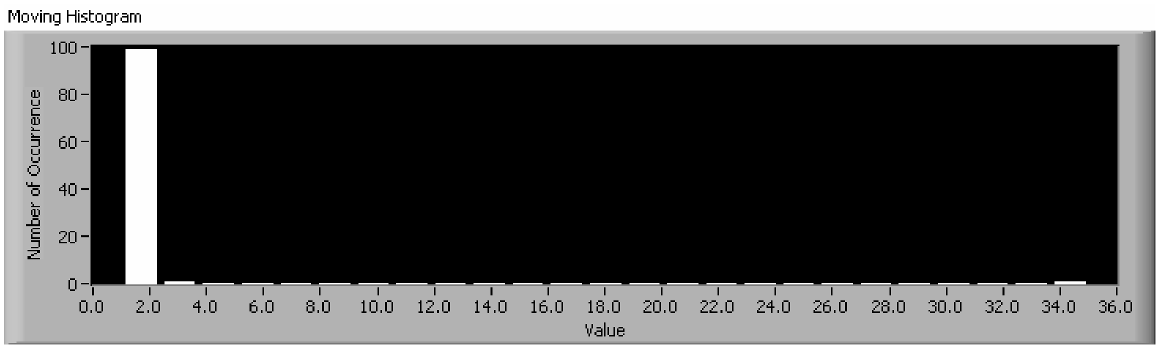
Mean = 2.36 msec
Standard Deviation = 3.49 msec

Priority = High



Mean = 2.18 msec
Standard Deviation = 3.62 msec

Priority = Real-Time



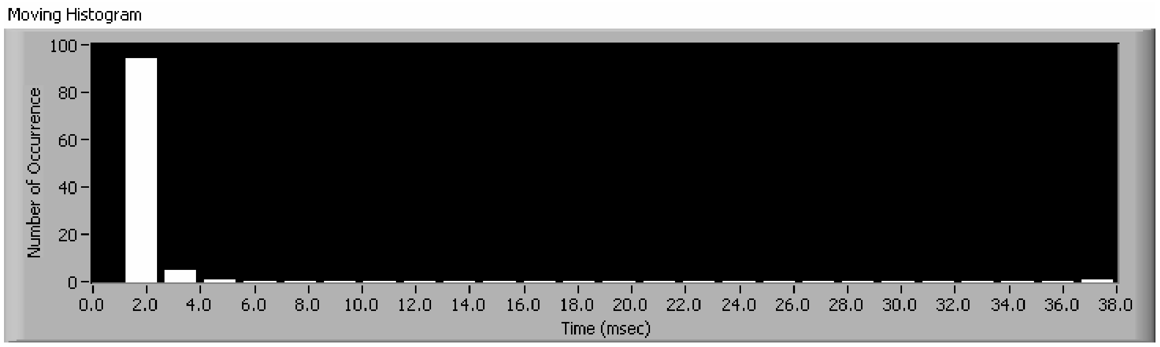
Mean = 2.13 msec
Standard Deviation = 3.33 msec

<u>*NOTE: Can not use more then 512 Taps when using Express VIs</u>

Sampling Frequency = 500 Hz
Number of Samples = 10

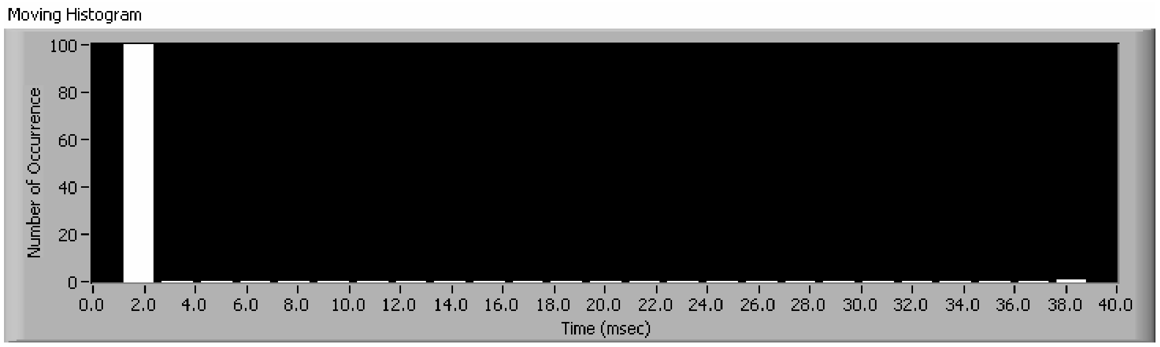
Whitening Filter Coefficients = 512

Priority = Normal



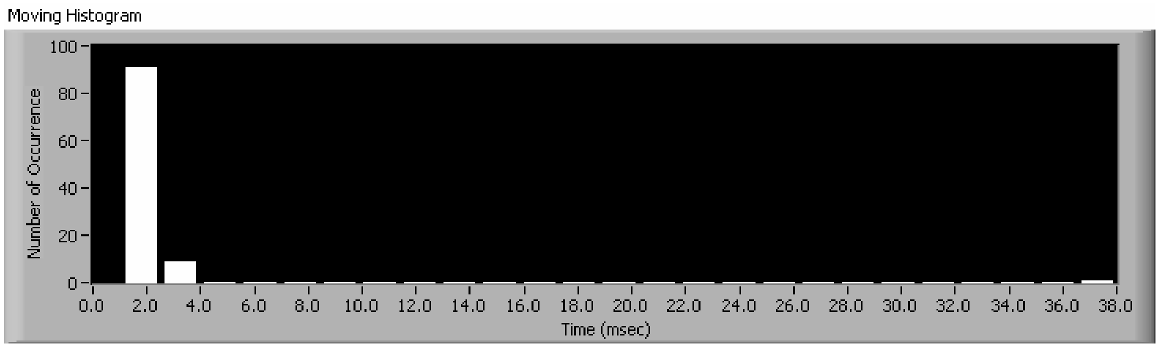
Mean = 2.39 msec
Std Dev = 3.60 msec

Priority = High



Mean = 2.71 msec
Standard Deviation = 3.72 msec

Priority = Real-Time



Mean = 2.40 msec
Standard Deviation = 3.60 msec

NOTE: Can not use more then 512 Taps when using Express VIs

Appendix G: Testing to Evaluate LabVIEW Real-Time Performance

In Chapter 6 of the report, information regarding how each EMG amplitude estimation algorithm (each utilizing one the three different LabVIEW analysis methods) was tested to determine which algorithm had the least latency. While summary plots were presented, this Appendix contains all of the collected raw data. Each test was conducted at the Normal, High and Real-time priorities. Details of the parameters set are shown with the histograms below. Furthermore, it should be noted that the simultaneous input and output was testing with the PCI-6229 using the single-point hardware-timed transfer method.

Raw Data

Iteration = 100
Intervals = 25
Channels = 1
Computation = None

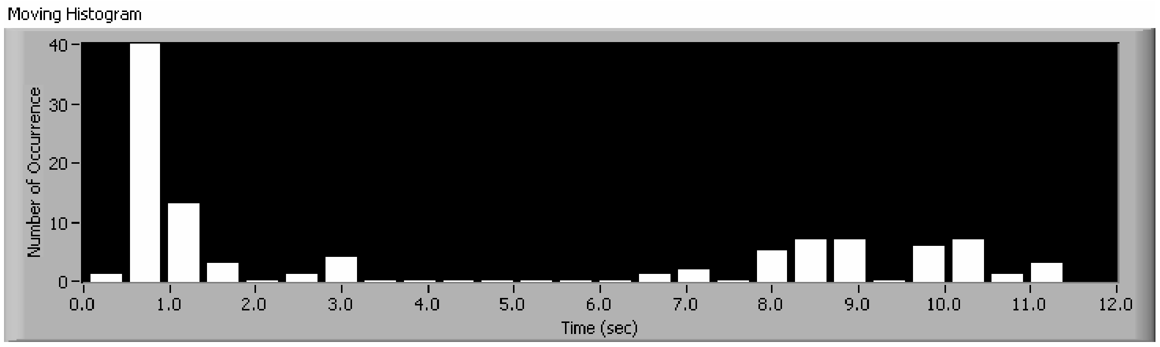
Sampling Frequency = 200 Hz

Priority = Normal

17 hours without a failure

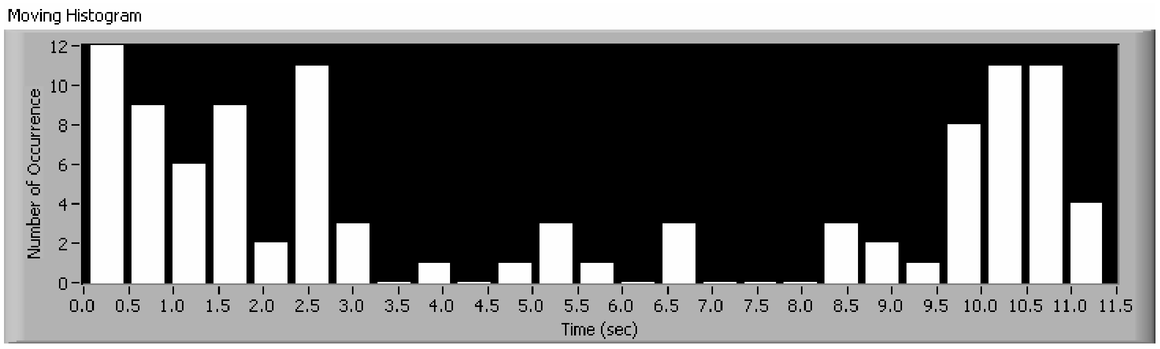
Sampling Frequency = 500 Hz

Priority = Normal



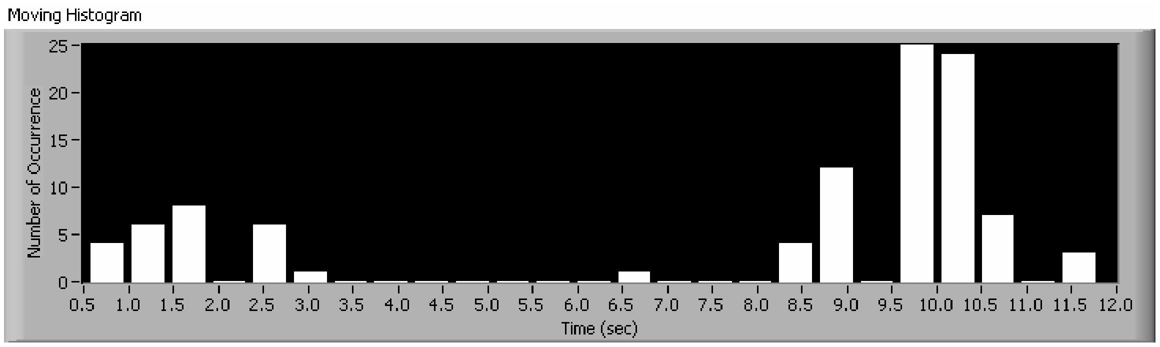
Mean = 4.09 (sec)
Standard Deviation = 4.11 (sec)

Priority = High



Mean = 5.21 (sec)
Standard Deviation = 4.26 (sec)

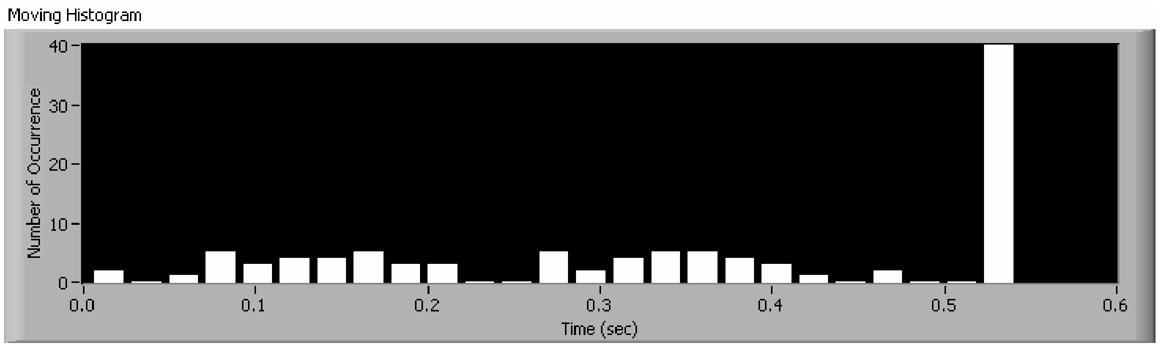
Priority = Real-Time



Mean = 7.74 (sec)
Standard Deviation = 3.62 (sec)

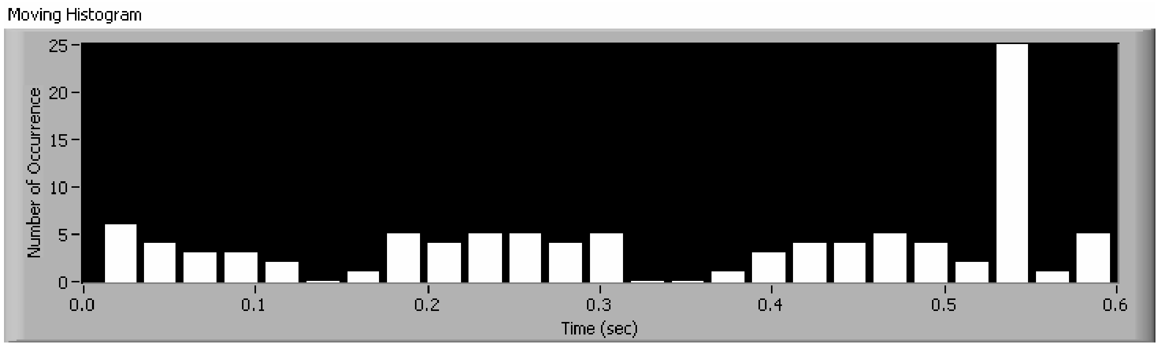
Sampling Frequency = 1000 Hz

Priority = Normal



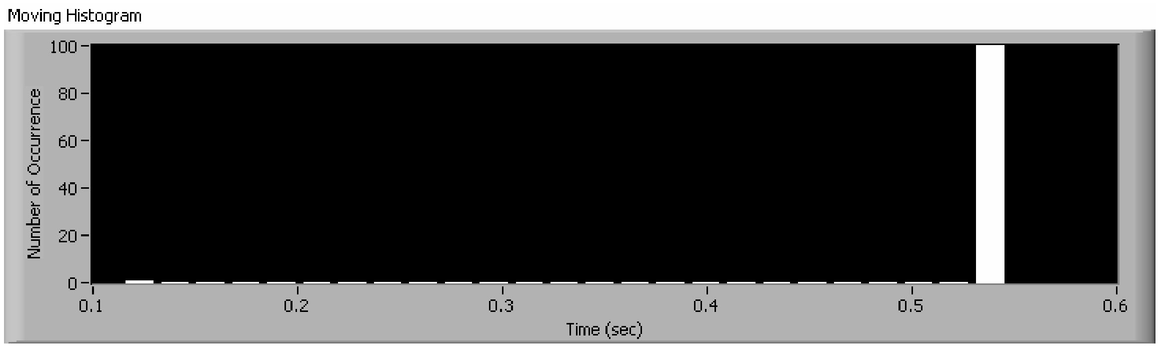
Mean = .36 (sec)
Standard Deviation = 0.18 (sec)

Priority = High



Mean = 0.35 (sec)
Standard Deviation = 0.19 (sec)

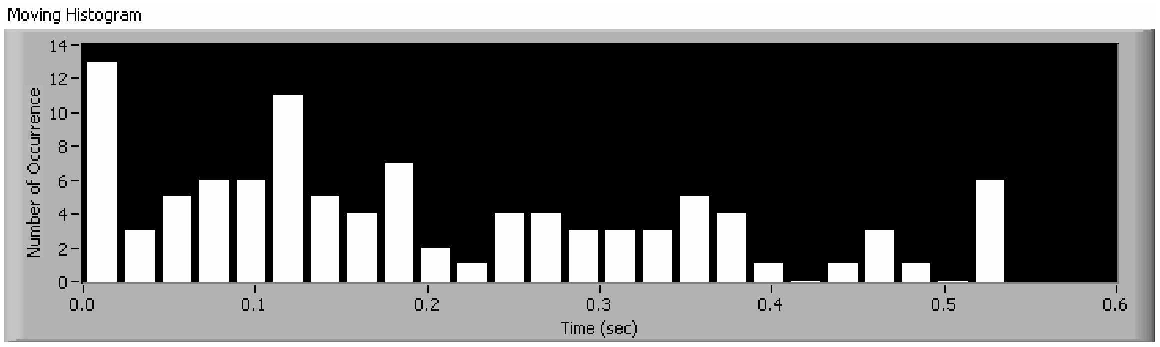
Priority = Real Time



Mean = 0.54 (sec)
Standard Deviation = 0.04 (sec)

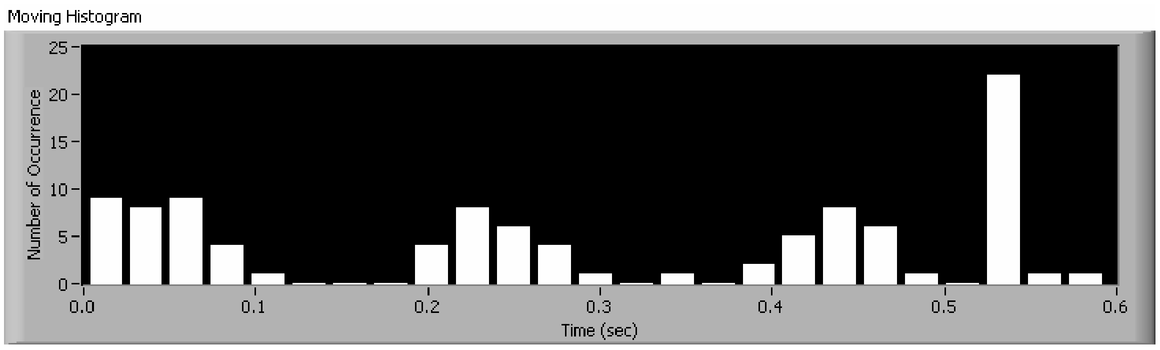
Sampling Frequency = 2000 Hz

Priority = Normal



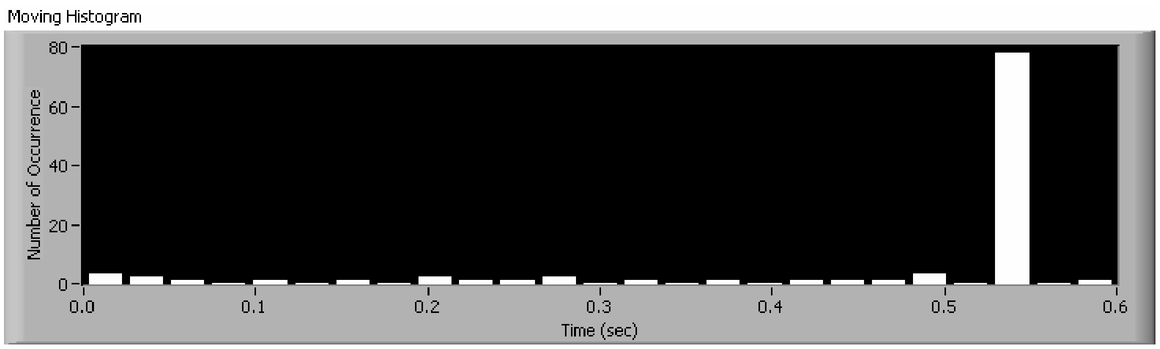
Mean = 0.20 (sec)
Standard Deviation = 0.15 (sec)

Priority = High



Mean = 0.30 (sec)
Standard Deviation = 0.20 (sec)

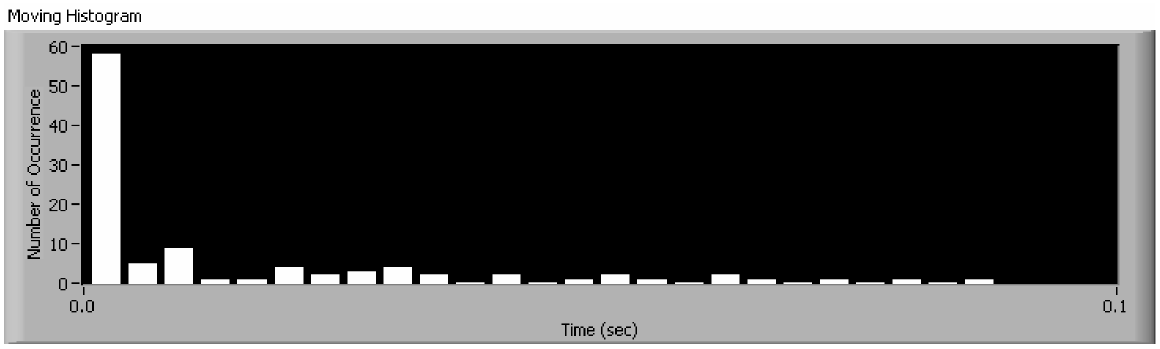
Priority = Real Time



Mean = 0.48 (sec)
Standard Deviation = 0.15 (sec)

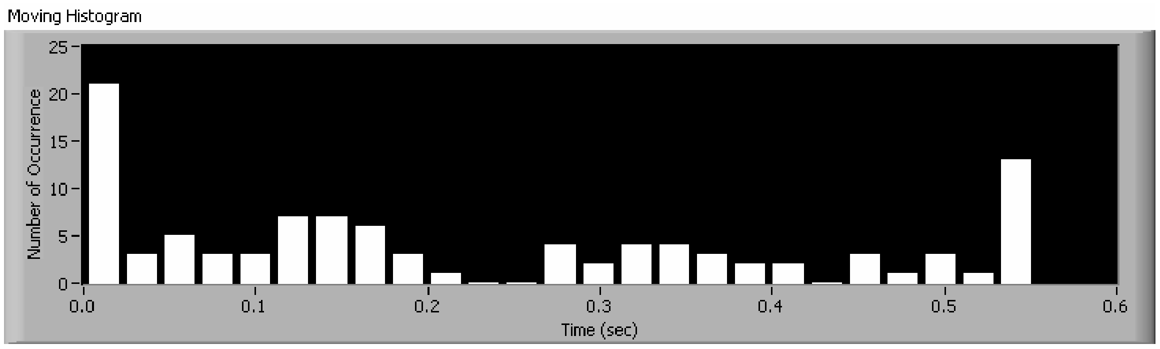
Sampling Frequency = 2500 Hz

Priority = Normal



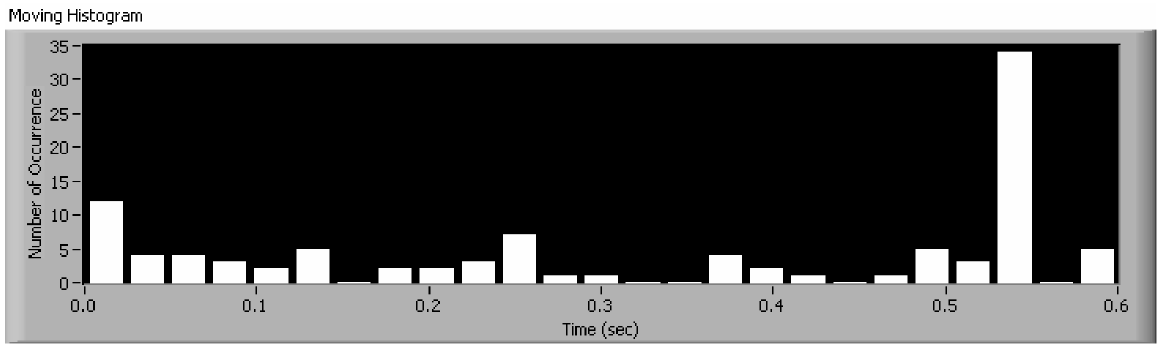
Mean = 0.01 (sec)
Standard Deviation = 0.02 (sec)

Priority = High



Mean = 0.22 sec
Standard Deviation = 0.19 sec

Priority = Real Time

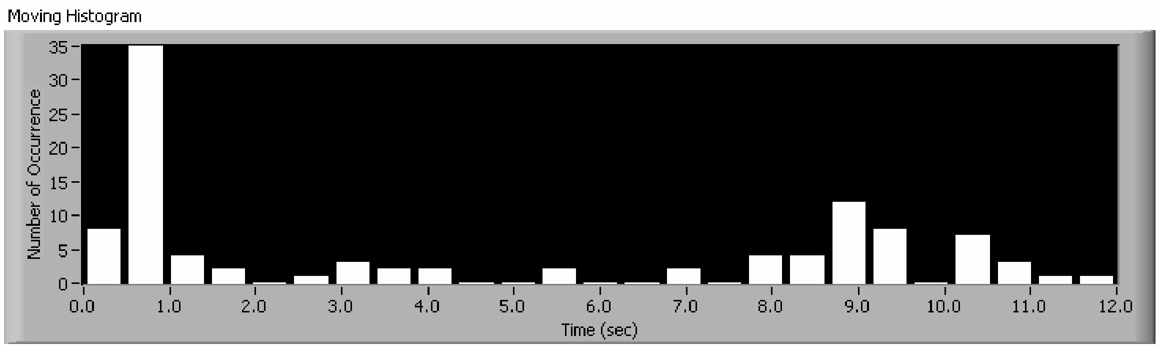


Mean = 0.34 sec
Standard Deviation = 0.22 sec

Iteration = 100
Intervals = 25
Channels = 2
Computation = None

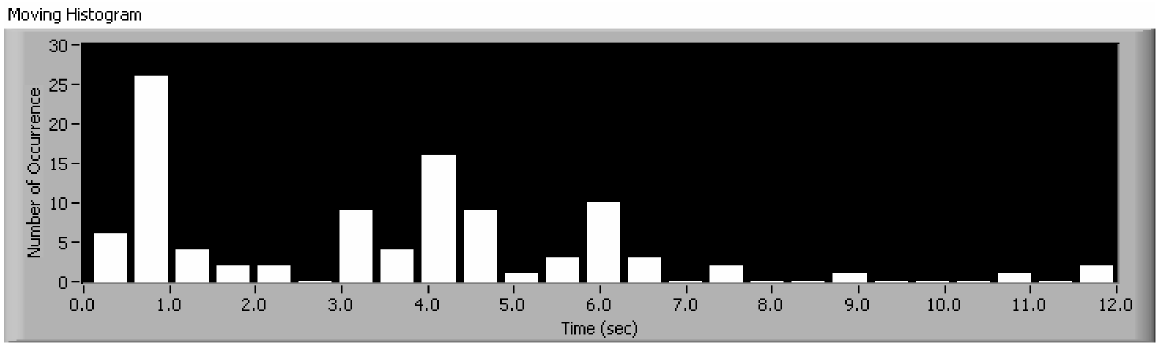
Sampling Frequency = 500 Hz

Priority = Normal



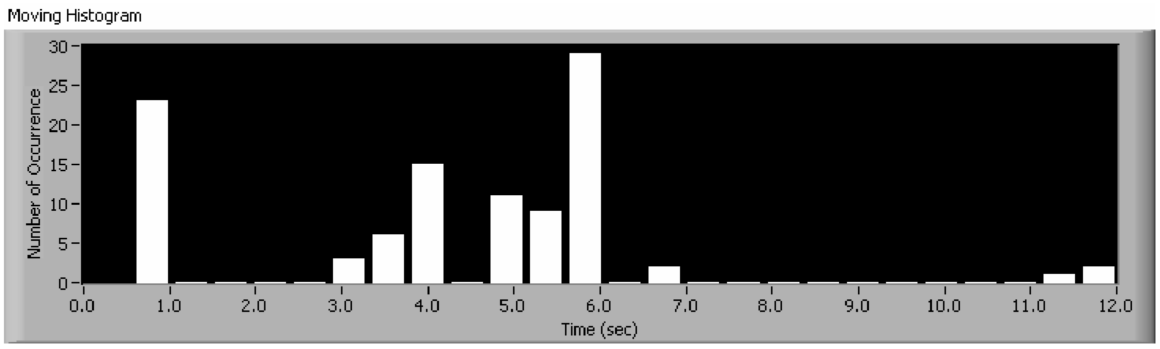
Mean = 4.54 (sec)
Standard Deviation = 4.22 (sec)

Priority = High



Mean = 3.37 (sec)
Standard Deviation = 2.64 (sec)

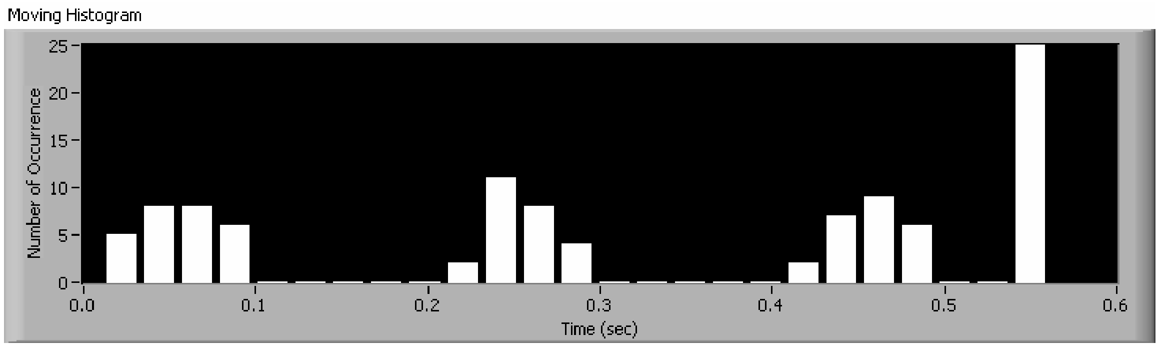
Priority = Real Time



Mean = 4.24 (sec)
Standard Deviation = 2.45 (sec)

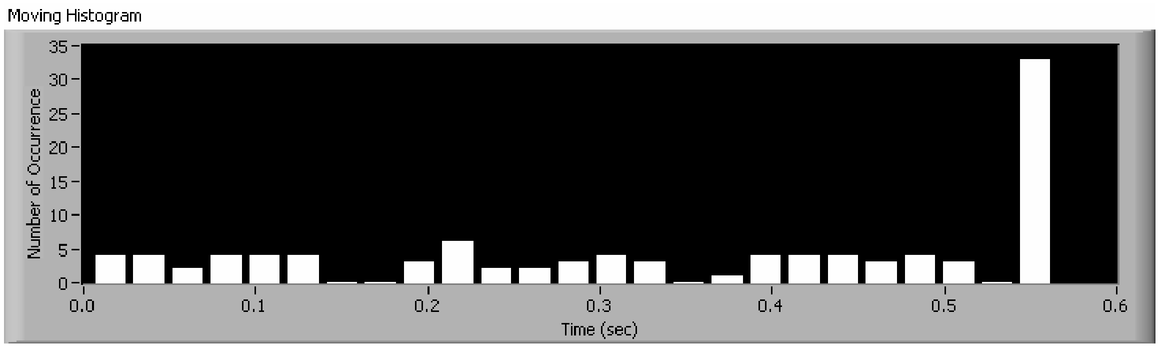
<u>Sampling Frequency = 1000 Hz</u>

Priority = Normal



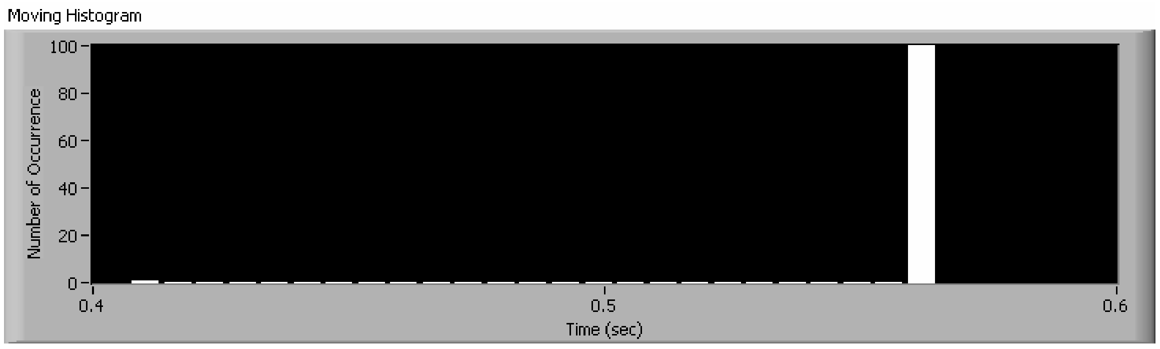
Mean = 0.32 (sec)
Standard Deviation = 0.20 (sec)

Priority = High



Mean = 0.36 (sec)
Standard Deviation = 0.19 (sec)

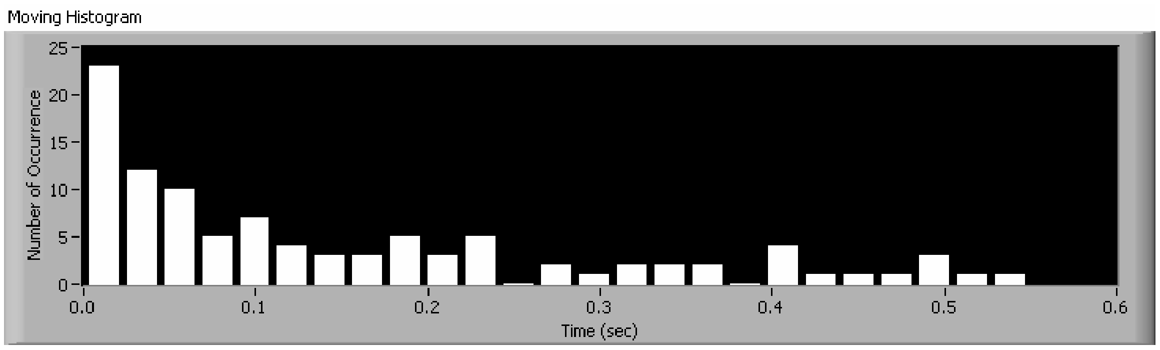
Priority = Real Time



Mean = .56 (sec)
Standard Deviation = 0.02 (sec)

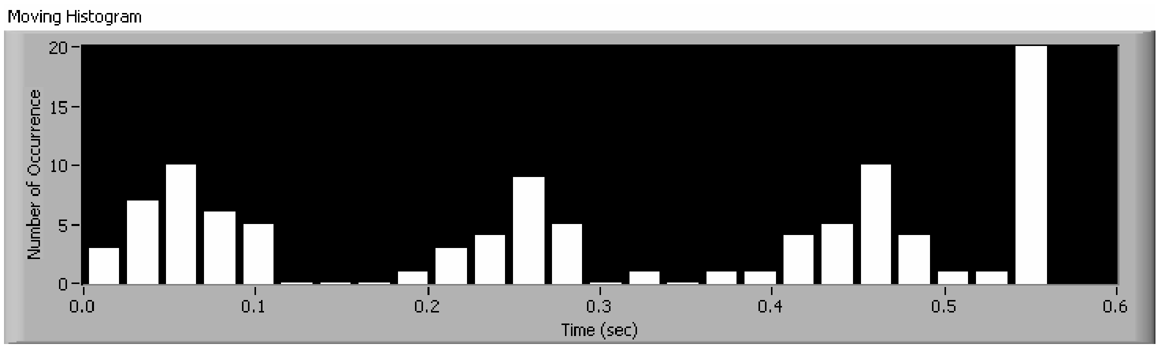
Sampling Frequency = 2000 Hz

Priority = Normal



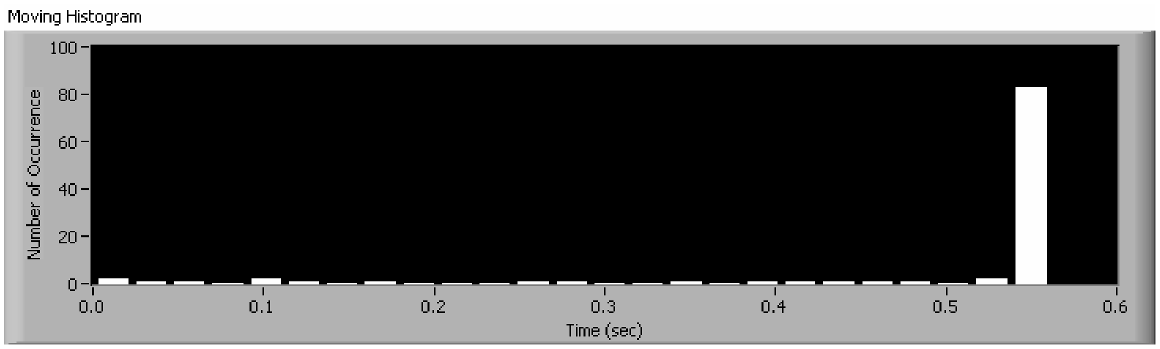
Mean = 0.15 (sec)
Standard Deviation = 0.15 (sec)

Priority = High



Mean = 0.31 (sec)
Standard Deviation = 0.20 (sec)

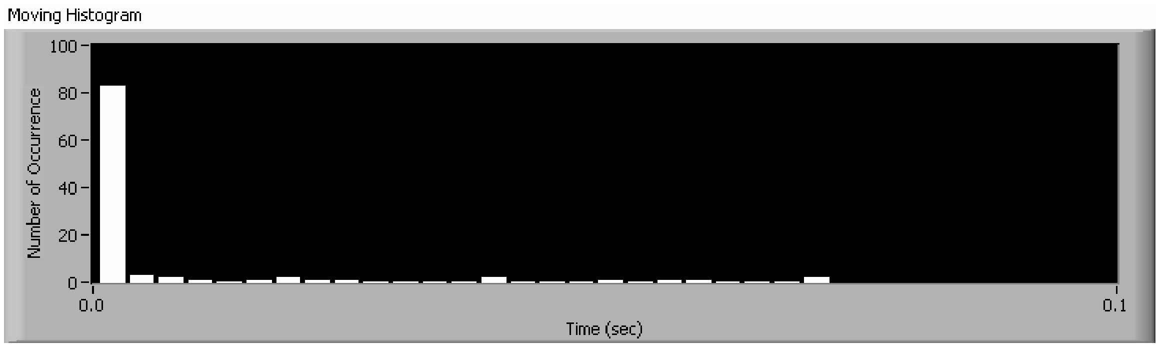
Priority = Real Time



Mean = 0.51 (sec)
Standard Deviation = 0.14 (sec)

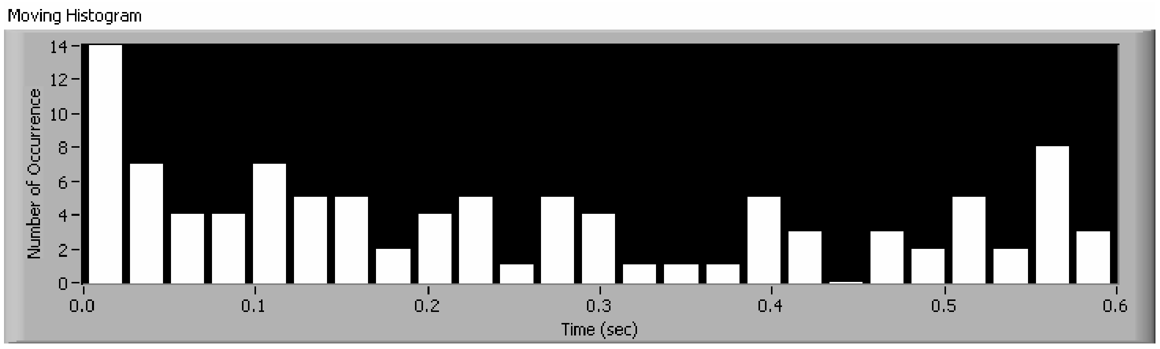
Sampling Frequency = 2500

Priority = Normal



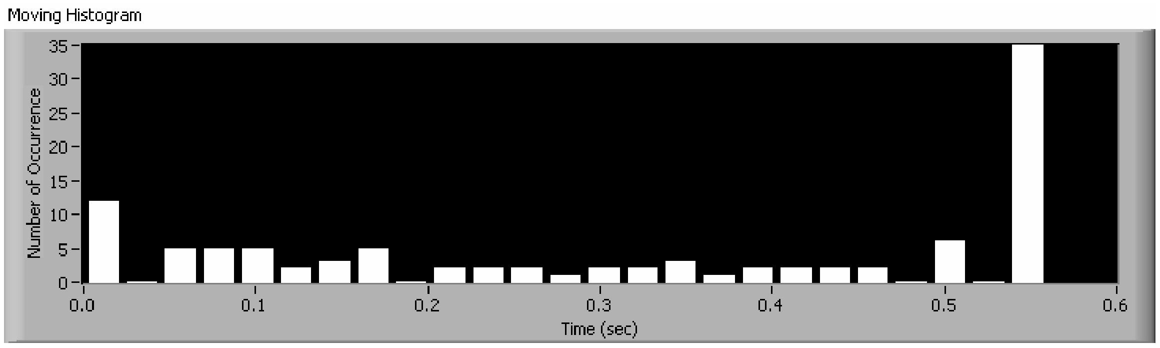
Mean = .01 sec
Standard Deviation = .01 sec

Priority = High



Mean = 0.25 sec
Standard Deviation = 0.20 sec

Priority = Real Time

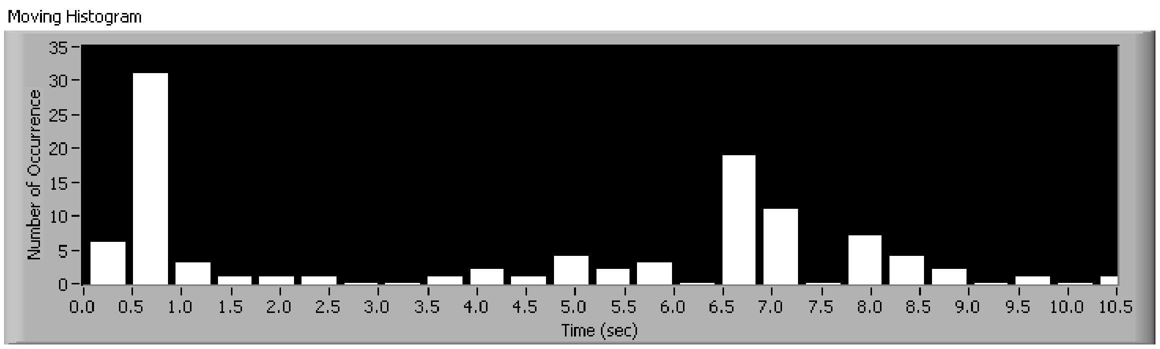


Mean = 0.33 sec
Std Dev = 0.22 sec

Iteration = 100
Intervals = 25
Channels = 4
Computation = None

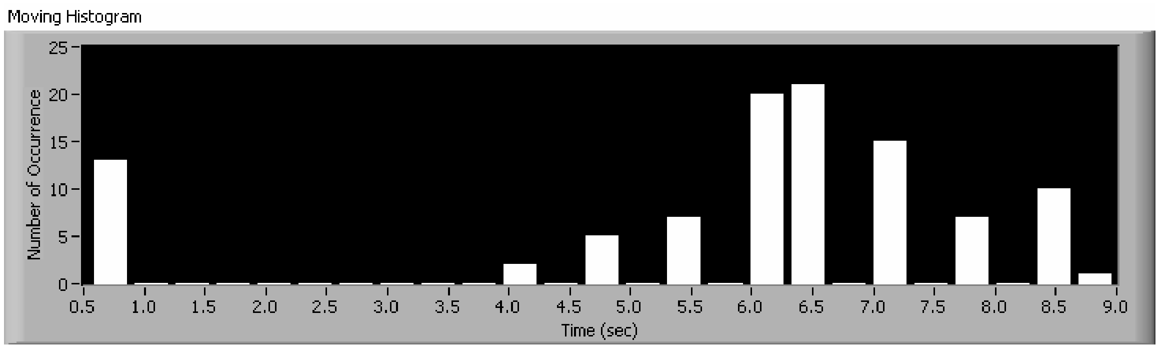
Sampling Frequency = 500 Hz

Priority = High



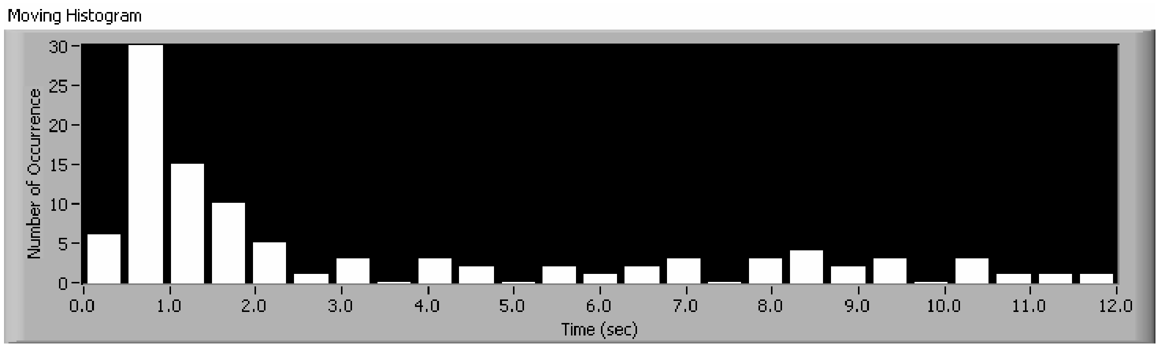
Mean = 4.19 (sec)
Standard Deviation = 3.23 (sec)

Priority = Real Time



Mean = 5.84 (sec)
Standard Deviation = 2.27 (sec)

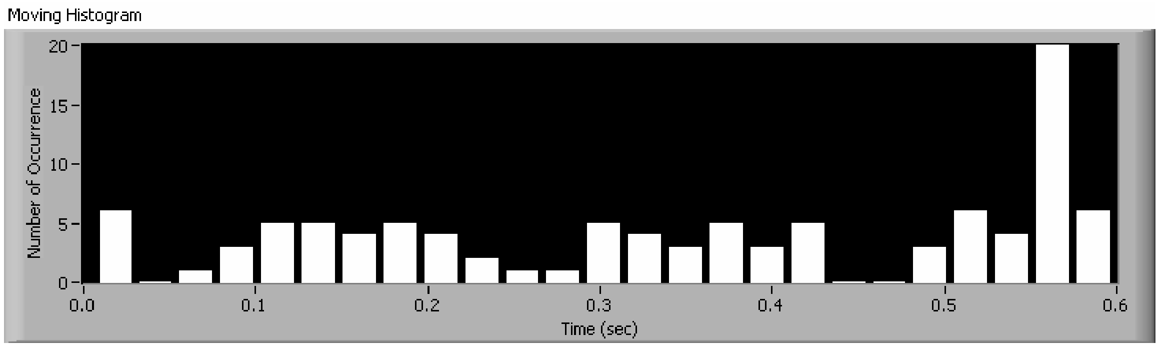
Priority = Normal



Mean = 3.15 (sec)
Standard Deviation = 3.40 (sec)

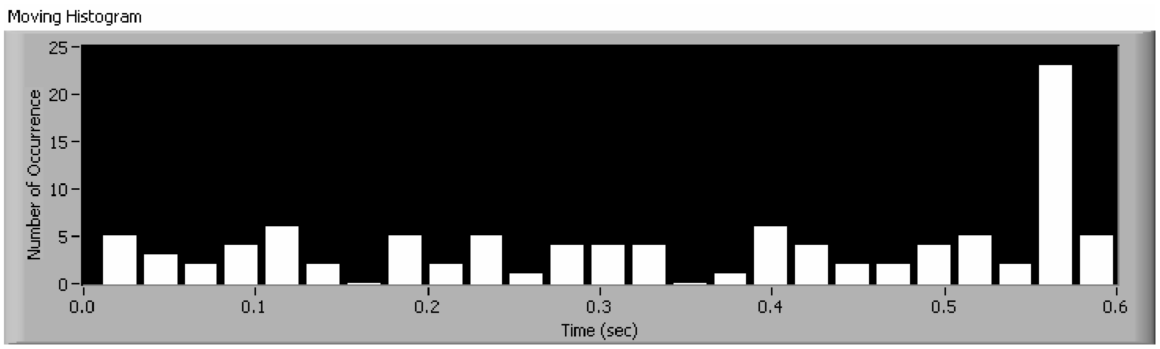
Sampling Frequency = 1000 Hz

Priority = Normal



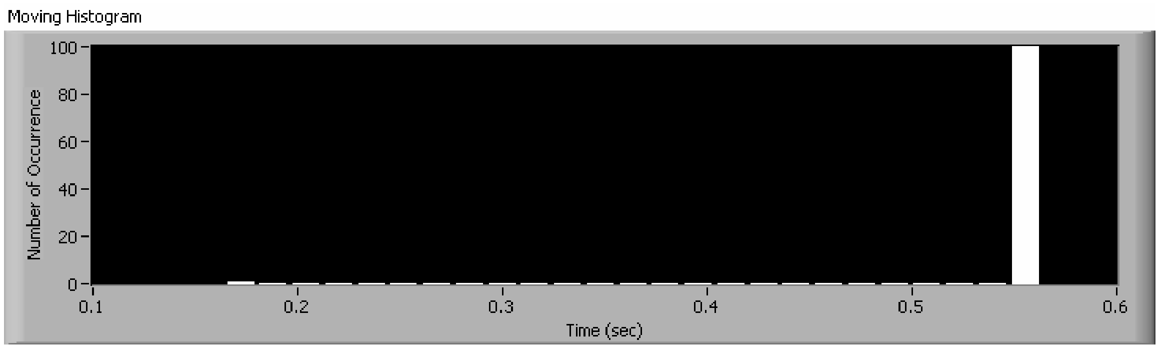
Mean = 0.35 (sec)
Standard Deviation = 0.19 (sec)

Priority = High



Mean = 0.36 (sec)
Standard Deviation = 0.19 (sec)

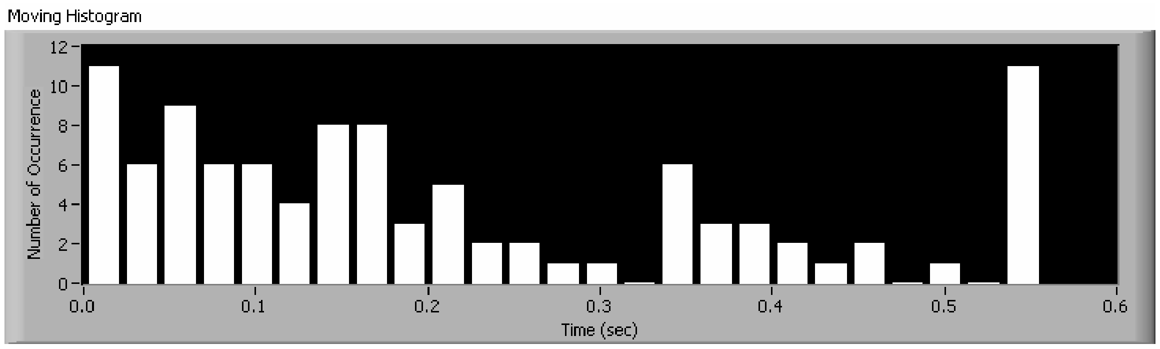
Priority = Real Time



Mean = 0.56 (sec)
Standard Deviation = 0.04 (sec)

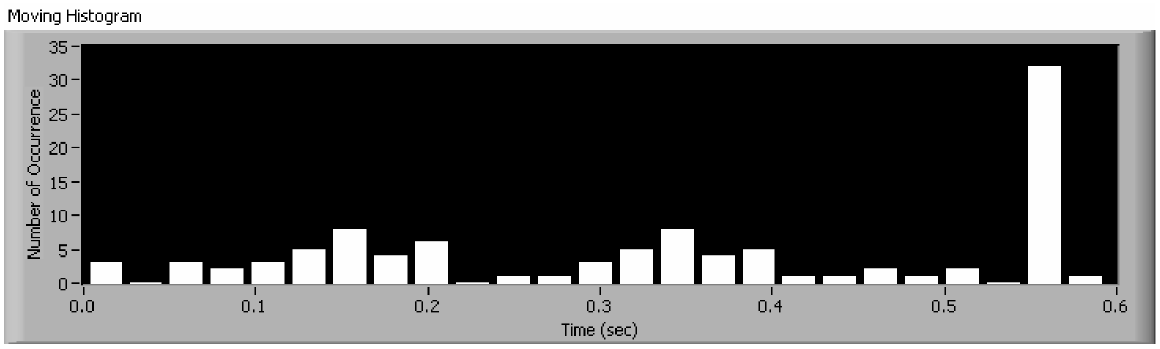
Sampling Frequency = 2000 Hz

Priority = Normal



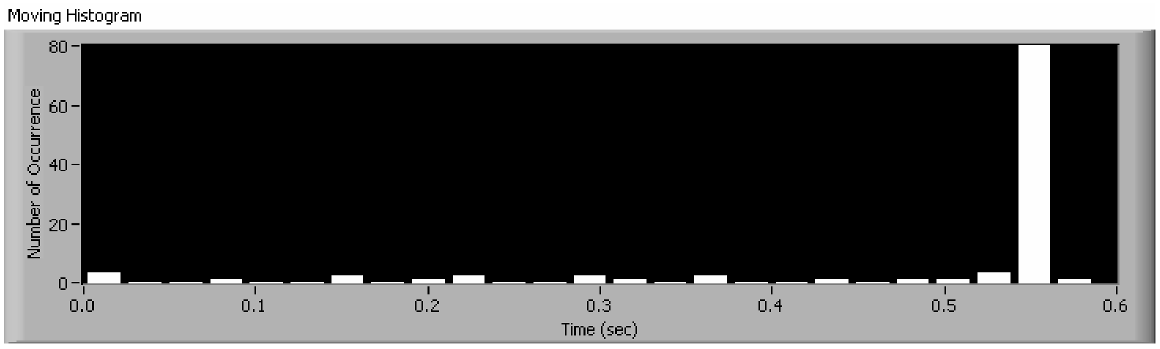
Mean = 0.21
Standard Deviation = 0.17

Priority = High



Mean = 0.35
Standard Deviation = 0.18

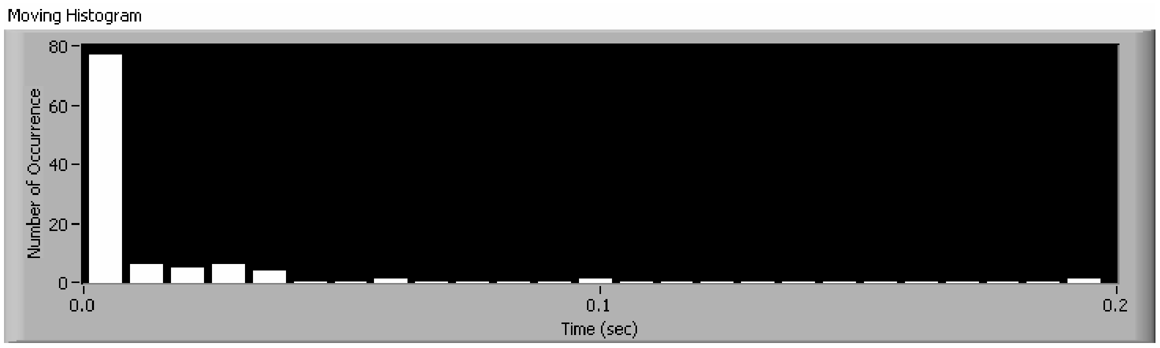
Priority = Real Time



Mean = 0.50 (sec)
Standard Deviation = 0.14 (sec)

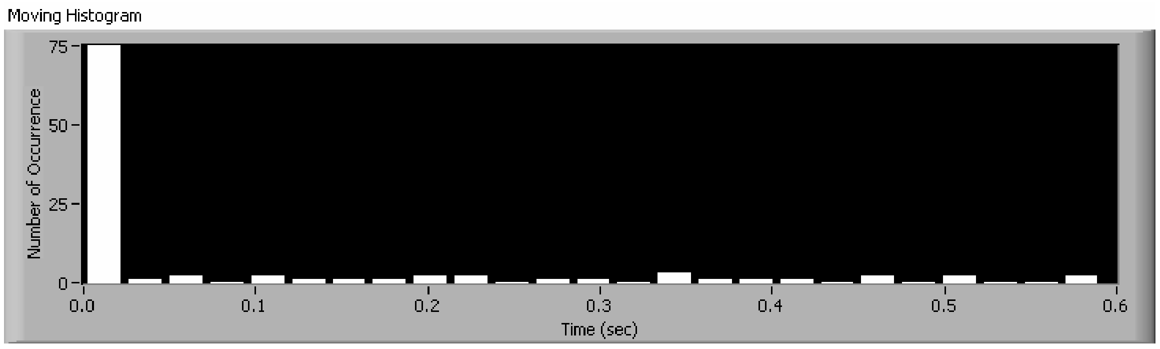
Sampling Frequency = 2500 Hz

Priority = Normal



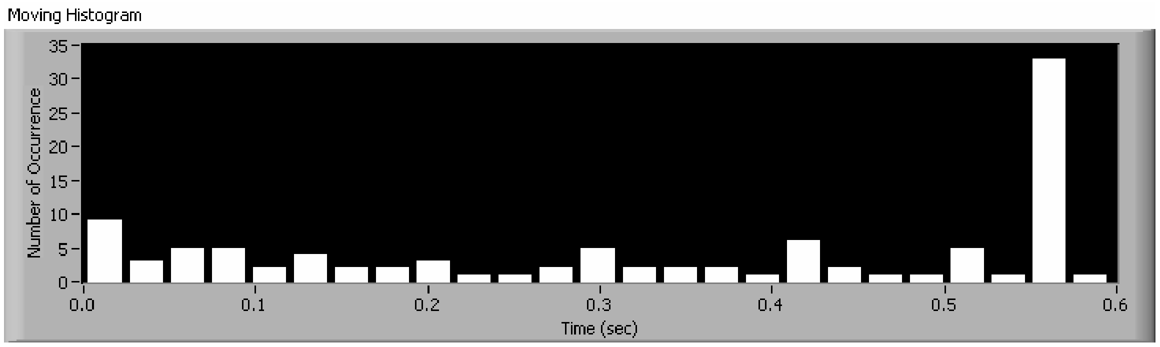
Mean = 0.01 sec
Std Dev = 0.02 sec

Priority = High



Mean = 0.34
Standard Deviation = 0.21

Priority = Real Time

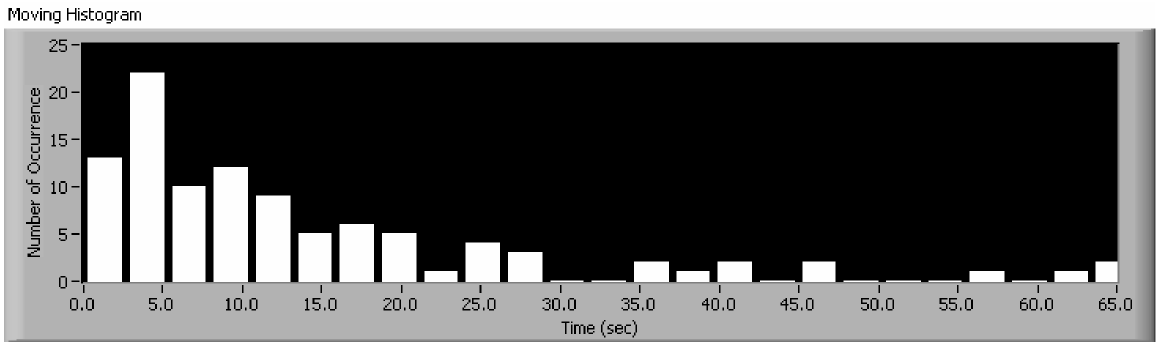


Mean = 0.34
Std Dev = 0.21

Iteration = 100
Intervals = 25
Channels = 1
Computation = 1 FIR LP Windowed Filter (no window), cutoff frequency = 0.125

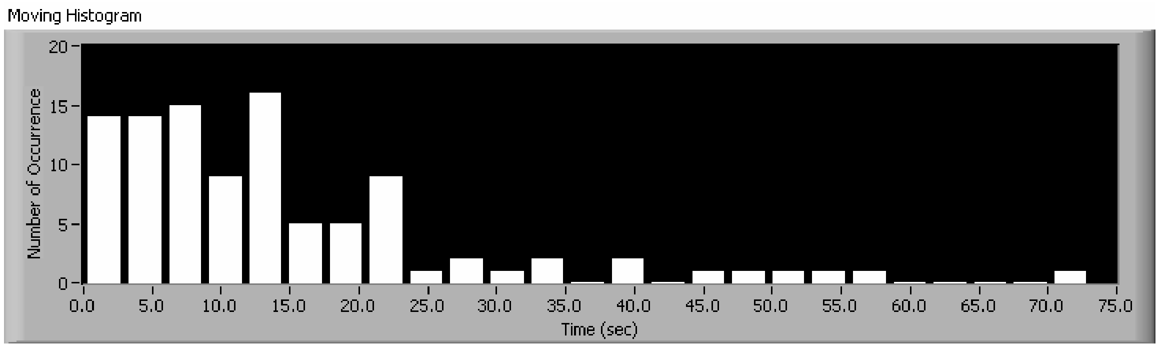
Sampling Frequency = 200 Hz

# Taps = 500
Priority = Normal



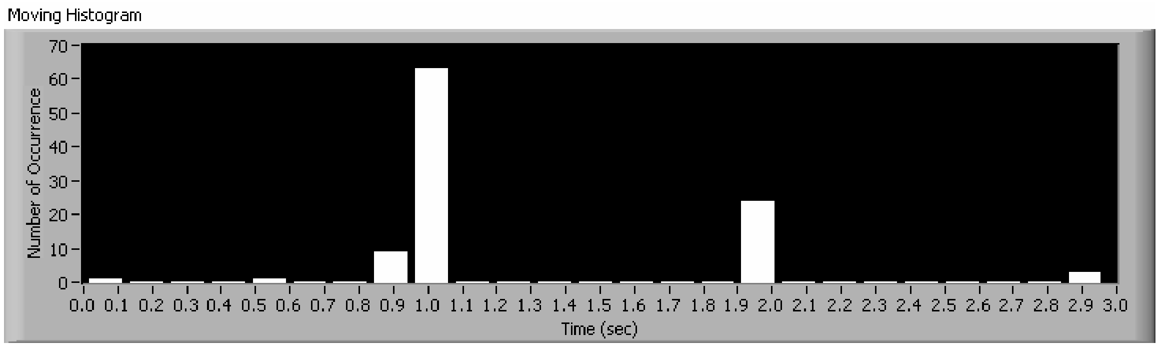
Mean = 13.70 sec
Standard Deviation = 14.59 sec

# Taps = 5000
Priority = Normal



Mean = 14.01
Standard Deviation = 13.77

# Taps = 50000
Priority = Normal



Mean = 1.24 sec
Standard Deviation = .54 sec

Appendix H: Detailed Description of EMG Amplitude Estimation VIs

In section 5.1 of the report, a high level overview of the three different EMG amplitude estimation VIs, which each utilize a different LabVIEW function type, was presented. In this section, a detailed description displaying how each routine was developed in LabVIEW will be presented.

Array-Based Implementation

The first implementation of the EMG amplitude estimation algorithm is the standard array based-algorithm. The block diagram of this VI can be seen in Figure EJ.1, while the LabVIEW block diagram can be seen in Figure EJ.2.

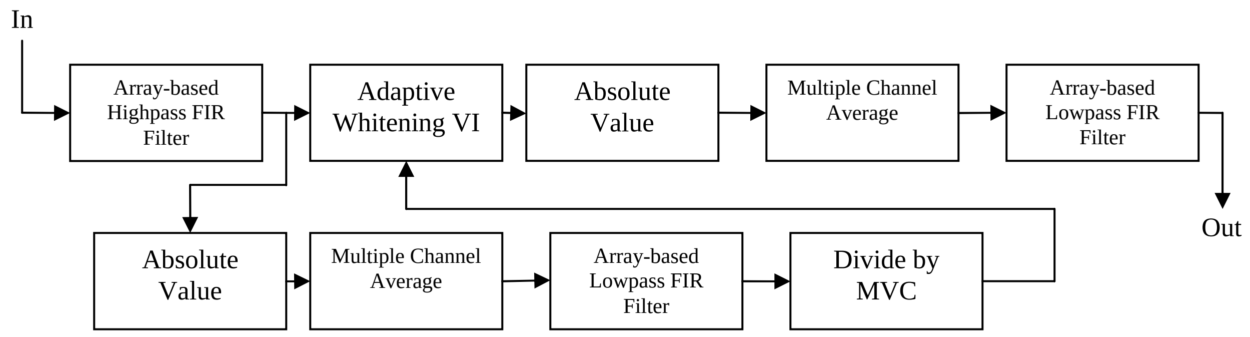


Figure EJ.1: Block Diagram of Array-Based VI

The input of this function is a multidimensional array of data. DAQmx functions, which will be used to acquire data for use with this VI, hold data from multiple channels in a multidimensional array. Each row in the multidimensional array contains data from each individual channel. As figures EJ.2 depict, the input signals are first placed through an array-based highpass FIR filter inside of a for-loop. The user is able to enter the parameters for this filter on the front panel. The parameters that the user is able to configure include the number of taps used and the type of window, if any, that the filter should implement. It should be noted that the for-loop is utilized here (and throughout this VI) to allow each row in the multidimensional input array, and thus all of the data from each channel, to be processed.

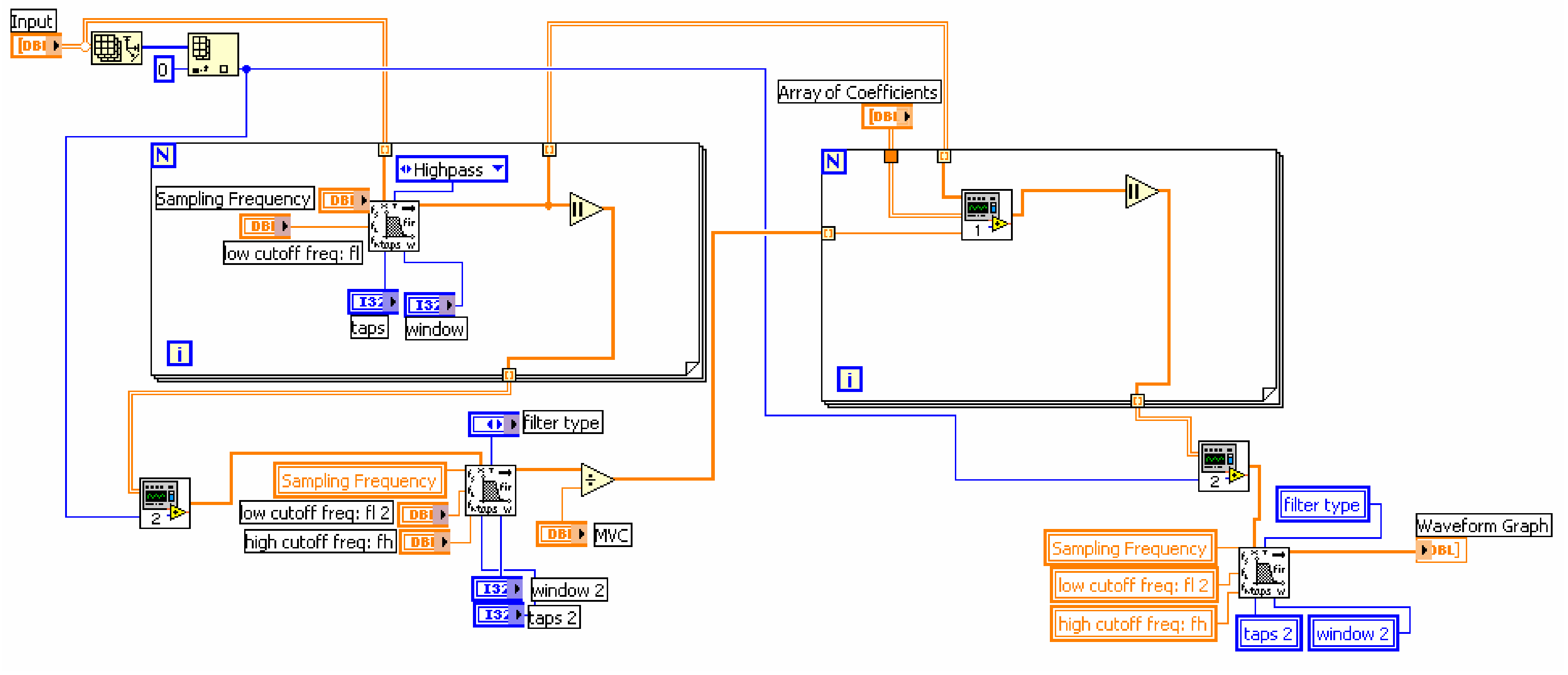


Figure EJ.2: EMG Array Based Amplitude Estimation VI

After the output of the highpass filter is absolute valued, data is passed to a module labeled with the number 2. This module is used to average the data from multiple channels and the implementation of this module can be seen in Figure EJ.3. This average module takes a two dimensional waveform, sums each element in a particular column together and then divides the entire array by the divisor to create a one-dimensional array result. It should be noted that the LabVIEW summation function sums each element in a row. As a result, in order to have each element in a column summed, the input array must first be transposed. Consequently, this module is used to average these data after it has been absolute valued.

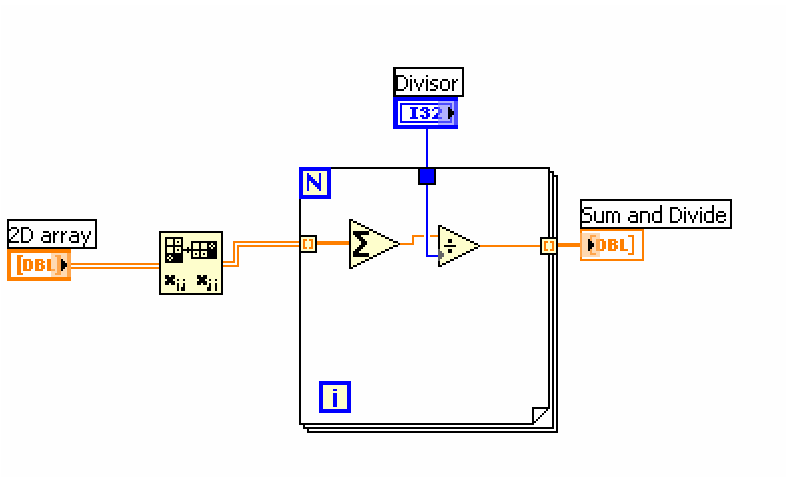


Figure EJ.3: Module that Averages Data from Multiple Channels

These averaged data are next placed through a smoothing filter. This smoothing filter was chosen to be implemented as an FIR windowed filter. The user is able to select on the front panel the type of filter, number of taps and window utilized to implement the smoothing filter.

Next, the smoothed data are normalized by the MVC and placed in the adaptive whitening VI, which is labeled with a 1 on the block diagram. Figure EJ.4 depicts the block diagram of this adaptive whitening filter module. The function takes as input first the array of data to be whitened. It then takes a multidimensional array of filter coefficients and a scaled amplitude value. Based on the scaled amplitude value entered, one of the vectors of filter coefficients is selected. A C code interface node is used to select the vector of filter coefficients. The code used to implement this CIN can be viewed in Appendix B.

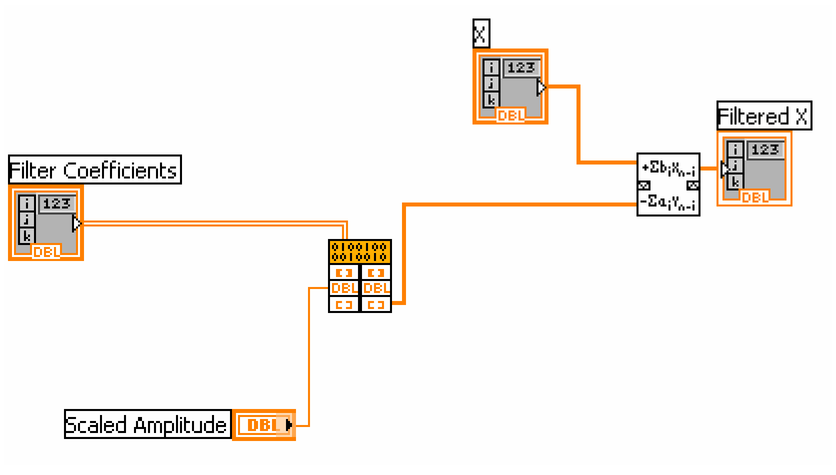
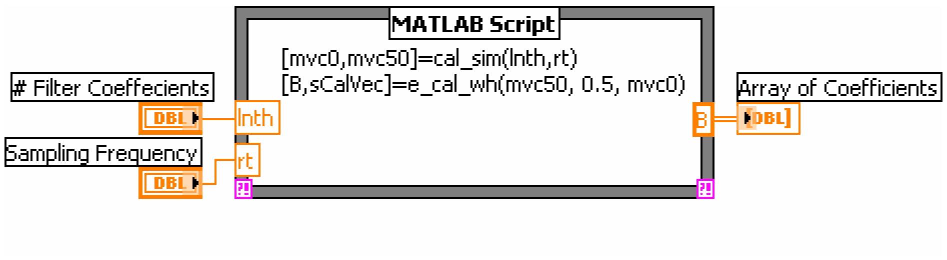


Figure EJ.4: Array-Based Adaptive Whitening VI

Next, with the appropriate array of filter coefficients selected, the LabVIEW FIR filter is utilized. This function implements the FIR filter specified by the vector of coefficients entered. Consequently, with the filter implemented, this function can be utilized in the EMG amplitude estimation VI. The array of filter coefficients is implemented by a module that utilizes a MATLAB script. Figure EJ.5 depicts the block diagram of the module used to construct the array of whitening coefficients. In this module two MATLAB functions are utilized, cal_sim and e_cal_wh, to calculate this matrix of filter coefficients. The function cal_sim is a function created by the authors to create a vector containing a calibration sample of the EMG signal acquired during a constant-force, constant-posture, nonfatiguing contraction and a vector containing a calibration sample of the noise signal. The function e_cal_wh, which actually designs the adaptive whitening filter coefficients, was previously created by the advisor of this project. These MATLAB functions can be seen in Appendix D.



```
[mvc0,mvc50]=cal_sim(lnth,rt)
[B,sCalVec]=e_cal_wh(mvc50, 0.5, mvc0)
```

Figure EJ.5: MATLAB Script Used to Construct Multidimensional Array of Whitening Filter Coefficients

The remainder of the EMG algorithm follows the block diagram located in the beginning of the chapter. First, the whitened signal is absolute valued. Next, the data is averaged again, using the previously described module. Finally, the averaged signal is placed through the previously defined smoothing filter. The output of this final smoothing filter gives the EMG amplitude estimate.

Express VI Implementation

Next, a block diagram version of the same EMG amplitude estimation algorithm described in section 5.2 implemented with Express VIs can be seen in Figure EJ.6. Figure EJ.7 displays the LabVIEW block diagram of the routine.

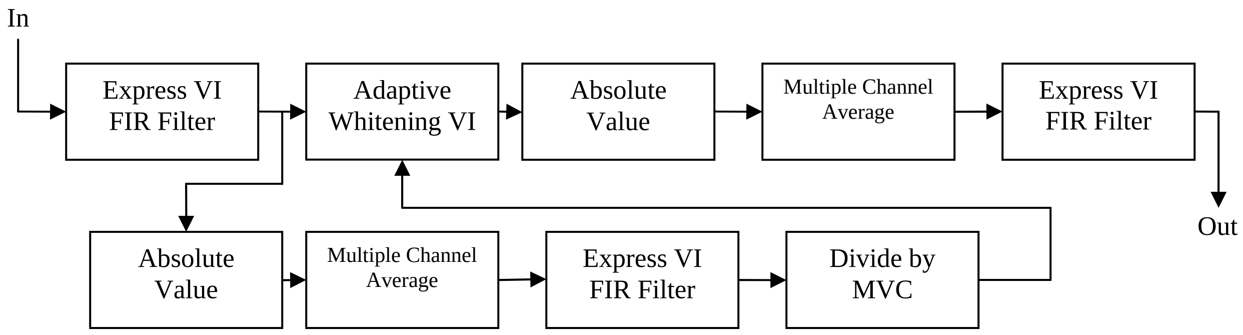


Figure EJ.6: Block Diagram of Array-Based VI

This routine again utilizes a high pass filter and smoothing filter. However, instead of using array based filters, Express Filter VIs are utilized. The parameters of these filters are configured by clicking on the filter blocks in the block diagram and entering the desired parameters in the graphical palette.

Another notable difference in this implementation is that this VI takes a time-stamped waveform containing the analog input from various channels as input. A for-loop can not be utilized to obtain time-stamped data from each channel. Consequently, in order to separate these data from each channel so that they can be combined, the LabVIEW waveform parser function must be utilized. In the version of the amplitude estimation function depicted in Figure EJ.7, a parser that only separates the waveform into two-channels is utilized. However, by simply clicking on the parser on the block diagram, the user is able to separate more channels if desired.

Also, the same adaptive whitening filter that was detailed in the previous section is utilized in this example. However, this previously detailed filter takes an array of data, in the form of a double data type, as input. Hence, in order to use this module with time-stamped data, a LabVIEW function that converts a time-stamped waveform to a standard multidimensional array of double values is utilized.

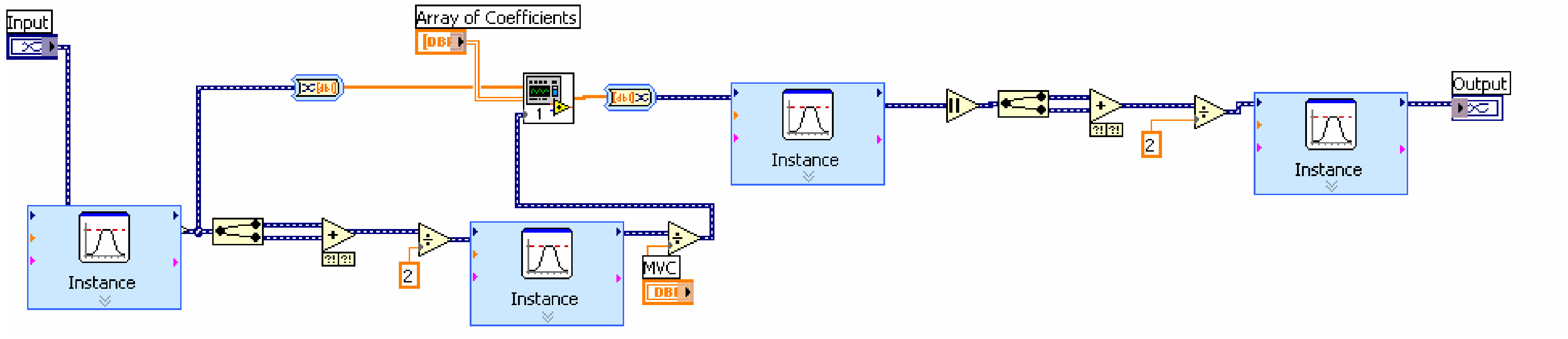


Figure EJ.7: Express VI EMG Amplitude Estimation Algorithm

Point-by-Point Implementation

The final version of the EMG amplitude estimation VI utilizes point-by-point analysis. A block diagram of the stages of this routine can be seen in figure EJ.8. Again, this routine is similar to the array-based version depicted in Figure EJ.1.

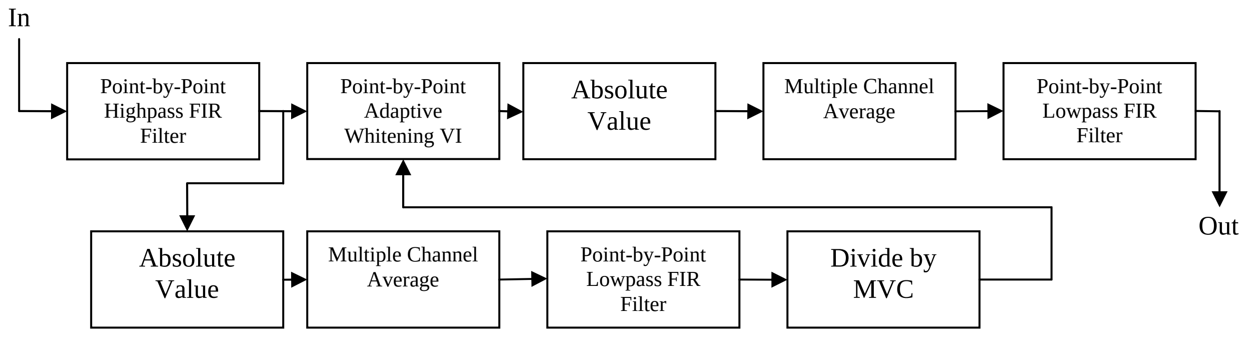

In
Point-by-Point Highpass FIR Filter
Point-by-Point Adaptive Whitening VI
Absolute Value
Multiple Channel Average
Point-by-Point Lowpass FIR Filter
Out
Absolute Value
Multiple Channel Average
Point-by-Point Lowpass FIR Filter
Divide by MVC

Figure EJ.8: Block Diagram of Point-by-Point VI

However, the main difference in this algorithm is that a one dimensional array is taken as input. Each piece of datum in this array comes from a unique analog input channel. The algorithm is then the standard algorithm described in section 5.2, however point-by-point FIR filters are utilized instead of array-based or Express VI filters.

Moreover, the adaptive whitening filter is also modified to become compatible with point-by-point analysis. While the method to calculate the matrix of possible whitening filter coefficients is the same as described in the beginning of this Appendix, the adaptive whitening function is slightly modified. This slightly modified adaptive whitening module can be seen in Figure EJ.9. This module uses the same CIN as previously described to select the appropriate array of filter coefficients based on the scaled amplitude entered. However, this function uses the point-by-point FIR filter to implement the desired filter specified by the array of coefficients instead of the array based function. Figure EJ.10 depicts the entire EMG amplitude estimation routine developed using point-by-point analysis.

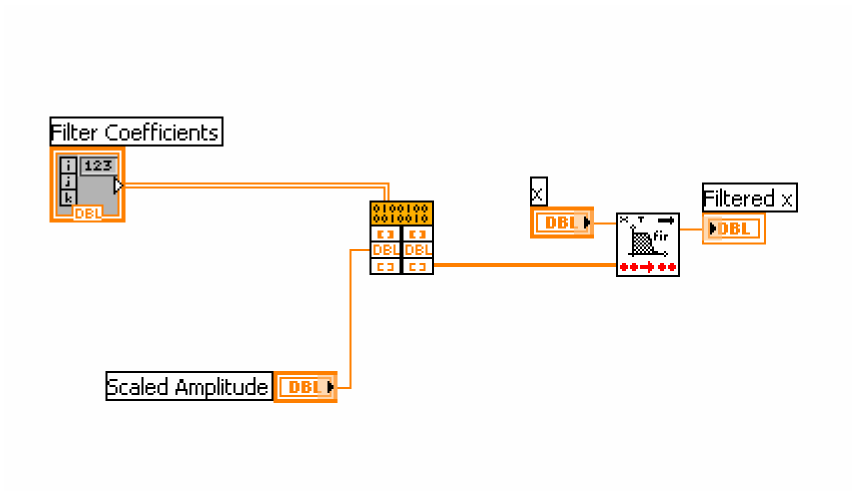


Figure EJ.9: Point-by-Point Adaptive Whitening Filter

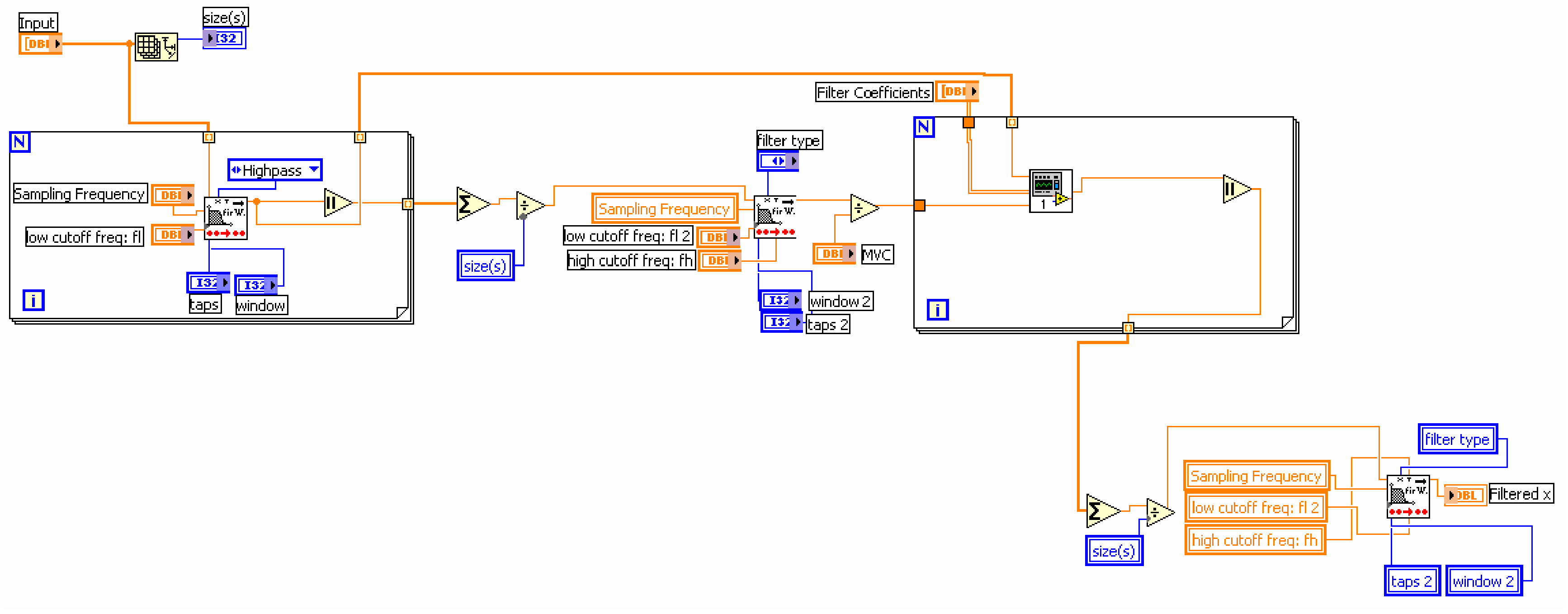


Figure EJ.10: Point-by-Point EMG Amplitude Estimation Algorithm