



March 11, 2005

Mr. Lakshman One  
School of Engineering Science  
Simon Fraser University  
Burnaby, British Columbia  
V5A 1S6

*Re: ENSC 440: Design Specification for a P300 Spelling Device for the Completely Disabled*

Dear Mr. One:

Please find the attached document, *Design Specification for a P300 Spelling Device for the Completely Disabled*, briefly outlining our chosen design. As you know, we are currently in the process of implementing an operational prototype of a brain computer interface that uses the P300 brain response to allow completely disabled persons, such as those suffering from late-stage *Amyotrophic Lateral Sclerosis*, to communicate.

In the attached document we give a brief system overview and we outline the system's design.

ThinQ Innovation is comprised of five senior-level engineering students, each with unique strengths to complement the group and a common trait in determination. We are Jack Cha, Jae-Seok Jeon, Brian John, Min Seo and Jyh-Liang Yeh. ThinQ Innovation looks forward to consulting with you on any concerns you may have. Should you have any questions, please feel free to contact us at [ensc440-project@sfu.ca](mailto:ensc440-project@sfu.ca).

Best Regards,

A handwritten signature in black ink that reads "Brian John". The signature is written in a cursive, flowing style.

Brian John  
Project Lead

Enclosure: *Design Specification for a P300 Spelling Device for the Completely Disabled*



## Design Specification for a **P300 Spelling Device for the Completely Disabled**

**Project Team Members:** Jack, Cha  
Jaeseok, Jeon  
Brian Henry, John  
Min, Seo  
Jyh-Liang, Yeh

**Contact:** [ensc440-project@sfu.ca](mailto:ensc440-project@sfu.ca)

**Submitted to:** Mr. Lakshman One – ENSC 440  
Mr. Mike Sjoerdsma – ENSC 305

**Issued date:** March 11, 2005

**Revision:** 1.0



## Executive Summary

ThinQ Innovation is currently developing a *P300 Brain Computer Interface* (BCI) spelling device prototype intended to provide a communication medium for completely disabled persons, such as those suffering from late-stage *Amyotrophic Lateral Sclerosis* (ALS). Ultimately, our system may allow late-stage ALS patients to lead a more fulfilling life and may also help researchers and medical doctors to gain further insight in the patient's physiological and psychological state.

The overall development of our system consists of 3 phases. After the first phase, we will have a prototype with essential functionalities. Once we are satisfied with the performance of our prototype, we will move into phase 2 to further improve the performance towards a production system. In the third phase, we will develop our system to include features necessary for commercial deployment. Phase 1 completion is scheduled for April, 2005 and will have the following main features:

1. Offline P300 detection accuracy of 99% in fewer than 10 trials using 10 data channels.
2. Online data collection and detection accuracy of 99% in fewer than 40 trials using 10 data channels.
3. Enhanced word selection using word prediction feature that will increase speed. Word selection will be available after the first target letter is decoded.

From a design point of view, our system can be broken down into four main modules: data acquisition module, output and control module, P300 detection module and word prediction module. As well, our P300 spelling device can operate in both real-time (online) and non-real-time (offline) modes. The remainder of this document further details our design for P300 Spelling Device.



## Table of Contents

---

<b>EXECUTIVE SUMMARY .....</b>	<b>II</b>
<b>1. INTRODUCTION.....</b>	<b>1</b>
1.1. SCOPE .....	2
1.2. INTENDED AUDIENCE.....	2
1.3. LIST OF ACRONYMS .....	2
<b>2. SYSTEM OVERVIEW.....</b>	<b>3</b>
<b>3. DATA ACQUISITION MODULE.....</b>	<b>6</b>
3.1. DATA COLLECTION MODULE HARDWARE .....	6
3.2. DATA COLLECTION MODULE SOFTWARE .....	8
3.2.1. <i>Data Collection Module Software—Online System.....</i>	<i>9</i>
3.2.2. <i>Data Collection Module Software—Offline System.....</i>	<i>10</i>
<b>4. CONTROL AND OUTPUT MODULE .....</b>	<b>11</b>
4.1. CONTROL AND OUTPUT MODULE HARDWARE.....	11
4.2. CONTROL AND OUTPUT MODULE SOFTWARE .....	12
<b>5. P300 DETECTION MODULE .....</b>	<b>15</b>
5.1 SUPPORT VECTOR MACHINE ALGORITHM .....	15
<b>6. WORD PREDICTION MODULE .....</b>	<b>19</b>
<b>7. TESTPLAN .....</b>	<b>20</b>
7.1. DATA COLLECTION MODULE TESTING .....	20
7.2. CONTROL AND OUTPUT MODULE TESTING .....	21
7.3. P300 DETECTION MODULE TESTING.....	22
7.4. WORD PREDICTION MODULE TESTING .....	22
<b>8. CONCLUSION .....</b>	<b>23</b>
<b>9. REFERENCES.....</b>	<b>24</b>

## List of Figures

---

Figure 2.1: System Overview .....	3
Figure 2.2: (Left) Display matrix currently flashing the second row, (right) first column .....	4
Figure 2.3: High-level system design overview. Arrows indicate direction of data flow.....	5
Figure 3.1: Data Acquisition Module Flow .....	6
Figure 3.1.1: DAQ Hardware Block Diagram .....	7
Figure 3.2.1.1: Data Overlapping in P300 Response .....	9
Figure 3.2.1.2: Data Flow in Online Data Collection Module .....	10
Figure 4.1.1: Row or column intensification and de-intensification for display output .....	12
Figure 4.2: Pattern for intensification and de-intensification timing.....	13
Figure 4.3: Pattern for data acquisition timing.....	13
Figure 4.4: Functional and signal dataflow flowchart for control and output modules .....	14
Figure 5.1.1: Optimal Hyperplane using SVM algorithm.....	15
Figure 5.1.2: Training Data Set .....	16
Figure 5.1.3: Variables after Training.....	17
Figure 5.1.4: Variables after Testing.....	18
Figure 6.1: A database showing top 20 common used words.....	19
Figure 7.1.1: Graphing GUI for displaying Testing Result.....	20
Figure 7.2: Output status messages for control timing testing.....	21

## List of Tables

---

Table 4.1.1: Table of timer API functions.....	12
Table 5.1.1: Variable Description .....	17

## 1. Introduction

The P300 Spelling Device is a communication system for completely disabled persons. The device uses visual stimuli in the form of flashing alphanumeric characters to invoke a response from the user's brain to determine the intended character and display on a screen for others to see. The system can be divided into four design modules: data acquisition module, output and control module, P300 detection module and word prediction module.

The development of our system consists of 3 phases. Our immediate goal is to develop a phase 1 prototype system to achieve fast (within 10 trials) and accurate (better than 99%) spelling using offline data. Because of uncertainties in regards to the performance of the available amplifier, we aim to utilize this system in a real-time implementation that can produce similar, but not identical, results as our offline system. Hence, we have broken phase 1 into two stages. In the first stage, we will analyze offline data in order to ensure that the detection of known results is accurate. The second stage will involve equipping the offline system with data acquisition capabilities and running it in a real-time fashion.

Once a prototype is developed and if we are sufficiently satisfied with its performance, we will move into phase 2 prototype development to improve the characteristics in preparation for a production system. Finally, once we have sufficiently improved our prototype such that it is suitable to enter the production phase (phase 3), we will further enhance the spelling device to prepare it for commercial deployment. Phase 1 is slated for completion in April, 2005.

## **1.1. Scope**

This document outlines the design of our P300 Spelling Device. The remainder gives sufficient design detail so that ThinQ Innovation will be able to use it as a reference document during implementation.

For a more general overview of the system, including financing and marketability, refer to our project proposal, *Proposal for a P300 Spelling Device for the Completely Disabled* [1].

For functional specifications, please refer to *Functional Specification for a P300 Spelling Device for the Completely Disabled* [2].

## **1.2. Intended Audience**

This document is not intended for external release and does not represent a finalized design. Until the design is finalized this document is intended as a reference document for ThinQ Innovation during implementation. The intended audience includes members of the design team, management and marketing persons who may need more detailed information of the system beyond the functional specifications.

## **1.3. List of Acronyms**

<b>ALS</b>	Amyotrophic Lateral Sclerosis
<b>API</b>	Advance Programming Interface
<b>BCI</b>	Brain Computer Interface
<b>CPU</b>	Central Processing Unit
<b>CRT</b>	Cathode Ray Tube
<b>DAQ</b>	Data Acquisition System
<b>EEG</b>	Electroencephalograph
<b>GUI</b>	Graphic User Interface
<b>LCD</b>	Liquid Crystal Display
<b>MND</b>	Motor Neuron Disease
<b>PC</b>	Personal Computer
<b>PCI</b>	Peripheral Component Interconnect
<b>SVM</b>	Support Vector Machine

## 2. System Overview

This section outlines the basic configuration of our P300 Spelling Device. It is intended to give the reader a better understanding of the overall system operation.

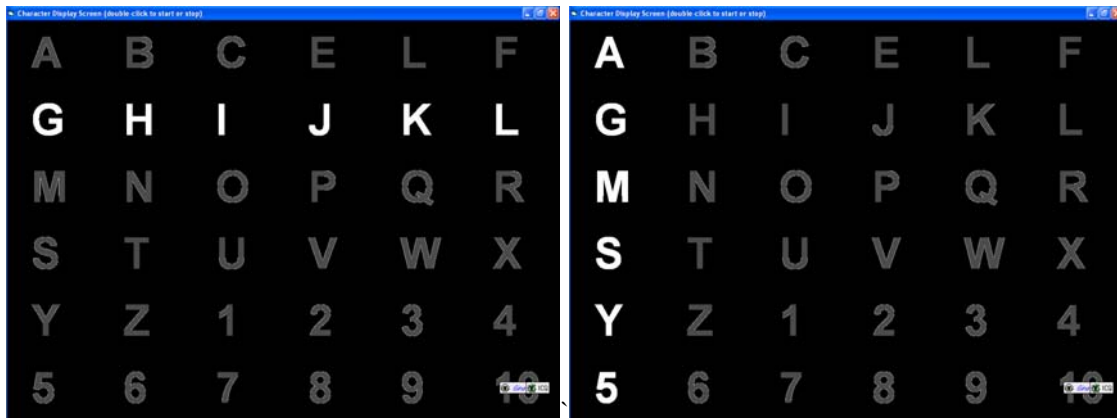
In a working P300 Spelling Device, shown below in figure 2.1, a monitor is placed in front of a user who is fitted with a head cap containing probes to detect brain activities. The probes, each one of which represents a “channel”, are connected to a biomedical amplifier, which then feeds the signals into a processor via an analogue to digital converter. During operation, we ask the user to focus on a target letter and count the number of times the letter flashes. During this time, rows and columns on the monitor will flash in random fashion, separated by approximately 300ms. When the target letter flashes, the user’s scalp will develop a P300 response. By keeping track of when particular rows and columns flash, accompanied with the response that the user gave after such flashes, our detection software will be able to determine the target letter. That is, a target letter will occur at the intersection of the row and column that gave positive P300 responses.



**Figure 2.1: System Overview**

Our system utilizes the fact that under certain circumstances, rare events elicit P300 potential. Flashing of target letters on the monitor evokes these rare events. A sample monitor output, taken at different times, is shown in figure 2.2. In the left case, the row G-H-I-J-K-L is highlighted, and a P300 response would be evoked on the user’s scalp if he or she had either of G, H, J, K, L, or I as the chosen target. All rows and columns are flashed in random sequence. On the right of Figure 2.2, the column A-G-M-S-Y-5 is highlighted. If, for this trial, a P300 was detected by our system directly after the flashing of highlighted row and column in figure 2.2, we could deduce that the user was trying to communicate the letter ‘G’.





**Figure 2.2: (Left) Display matrix currently flashing the second row, (right) first column**

The time required between flashing rows or columns, denoted as a single “epoch”, lasts approximately 300ms. For the display matrix depicted above, it would require a total of 12 epochs (6 rows + 6 columns) to constitute as a single trial. The P300 will appear approximately 300ms after the intensification of a target letter and is characterized by a prolonged positive change in scalp potential. The system’s task then reduces to the detection of which row and which column that has elicited a P300 on a given trial.

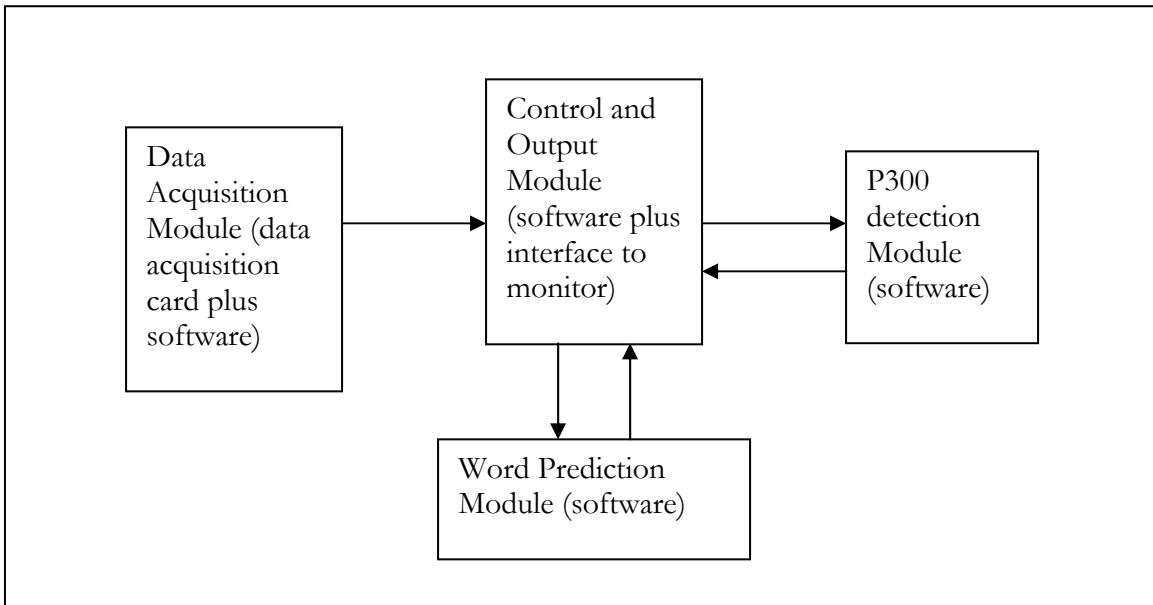
Because the P300 is substantially smaller than the continuous EEG activities on the scalp, it is very difficult to detect it on a single-trial basis. Thus, the detection of the P300 within a noisy EEG environment will require a series of trials averaged together to obtain more accurate detection. The distinguishing feature of any BCI is the ability to communicate efficiently and therefore it is clearly desirable to minimize the number of trials in the P300 Spelling Device.

After successful detection of a target letter, the user’s monitor will be updated to reflect the choice of letter. To increase the communication rate of our system, we intend to incorporate a word prediction function feature. It involves developing a database of commonly used words, ranked based on the frequency of usage. When our system is able to make a guess as to the target word, it will display it on the screen as a possible user input.

Based on rank and previously chosen letters, the word prediction feature will output probable word choices. Thus, the six most common words matching a partially typed word will be displayed on an additional row of the matrix screen.

From a design point of view, our system can be broken down into four categorical modules: data acquisition module, output and control module, P300 detection module and word prediction module. As shown in figure 2.3, data flows from the data acquisition module to the control and output module, which passes it to the P300 detection module. The P300 module then gives back to the control and output module a score indicating the possibility

of P300 absence/presence, so that the control and output module may display the chosen character on the display. Correctly decoded words are also passed to the word prediction module so that the control and output module may also display possible word choices to the monitor.

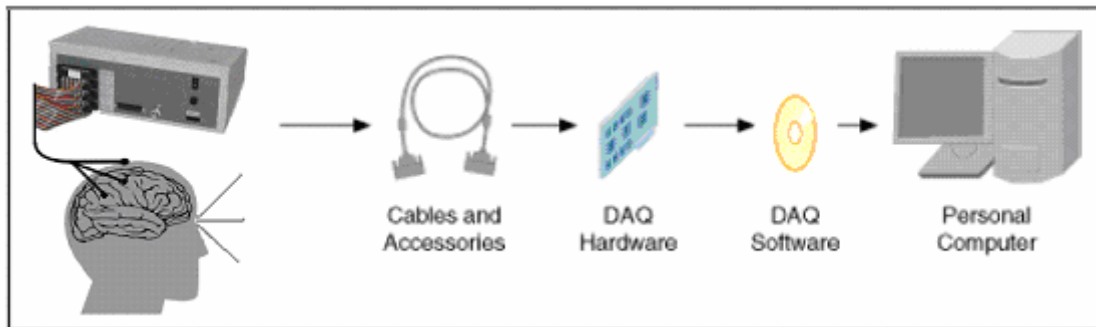


**Figure 2.3: High-level system design overview. Arrows indicate direction of data flow.**

An important function of our P300 Spelling Device is to be able to operate in offline mode as well as in online mode. Offline mode allows the processing of pre-collected data and is necessary for both design development and further optimization. We anticipate that we will continue to operate our P300 Spelling Device in offline mode even after the completion of prototype development thereby making it necessary to integrate offline mode into our design. Hence, where online and offline mode differ, we have included separate sections outlining their design. Subsequent sections of this document will describe the design of each of these modules and briefly illustrate the corresponding test plans.

### 3. Data Acquisition Module

We use NI-6220 *Data Acquisition* (DAQ) system by National Instruments for analogue to digital conversion. Figure 3.1 illustrates how the data is acquired from user's brain and reaches the processor (Personal Computer in figure 3.1).



**Figure 3.1: Data Acquisition Module Flow**

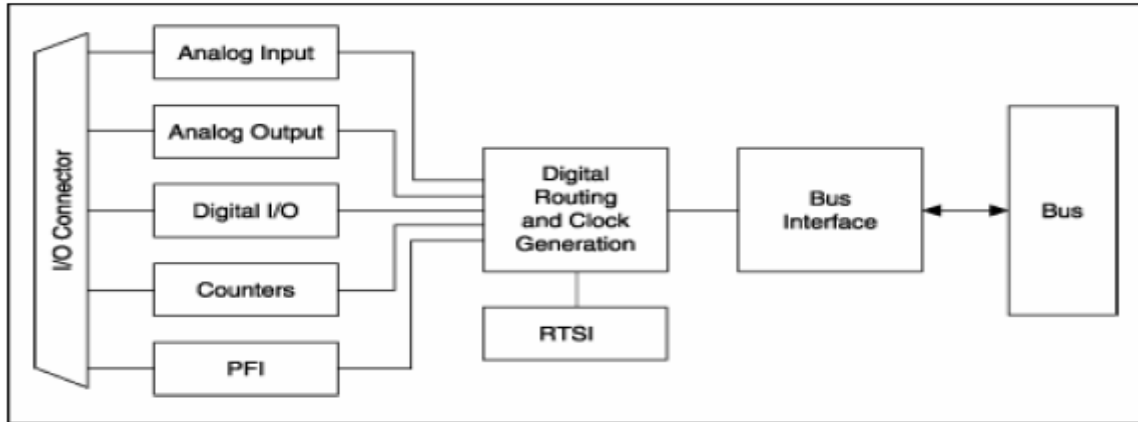
The data acquisition module consists of an amplifier, a personal computer, DAQ software/hardware and cables that connect to and from NI-6220.

For both offline and online operation, the data collection module is responsible for data partition and data filtration for processing, this is what we call pre-processing. The aforementioned pre-processing is done through the DAQ software. Online operation, in particular, requires the DAQ hardware to collect and condition real-time signals that come from the user's scalp.

The following sections outline the design of our data collection module hardware and software.

#### **3.1. Data Collection Module Hardware**

The DAQ hardware is responsible for A/D conversion of the input signals to fit input signal profile. Figure 3.1.1 shows the DAQ hardware block diagram.



**Figure 3.1.1: DAQ Hardware Block Diagram**

The DAQ hardware has maximum sample rate of 250 kilo-samples per second and features *Common Mode Rejection Ratio* (CMRR) of 95dB in the range between DC and 60Hz. Since our P300 Spelling Device requires sampling rate of 240 samples per second and CMRR greater than 80dB in the range between 0.5Hz and 15Hz, both characteristics satisfy the system requirements. Furthermore, due to the inherent noise accompanied by EEG signals, it is desirable to have input impedance of greater than 100M $\Omega$ , which is well below the DAQ input impedance of 10,000M $\Omega$ .

The following figure 3.1.2 shows the pin-out of the DAQ hardware connector

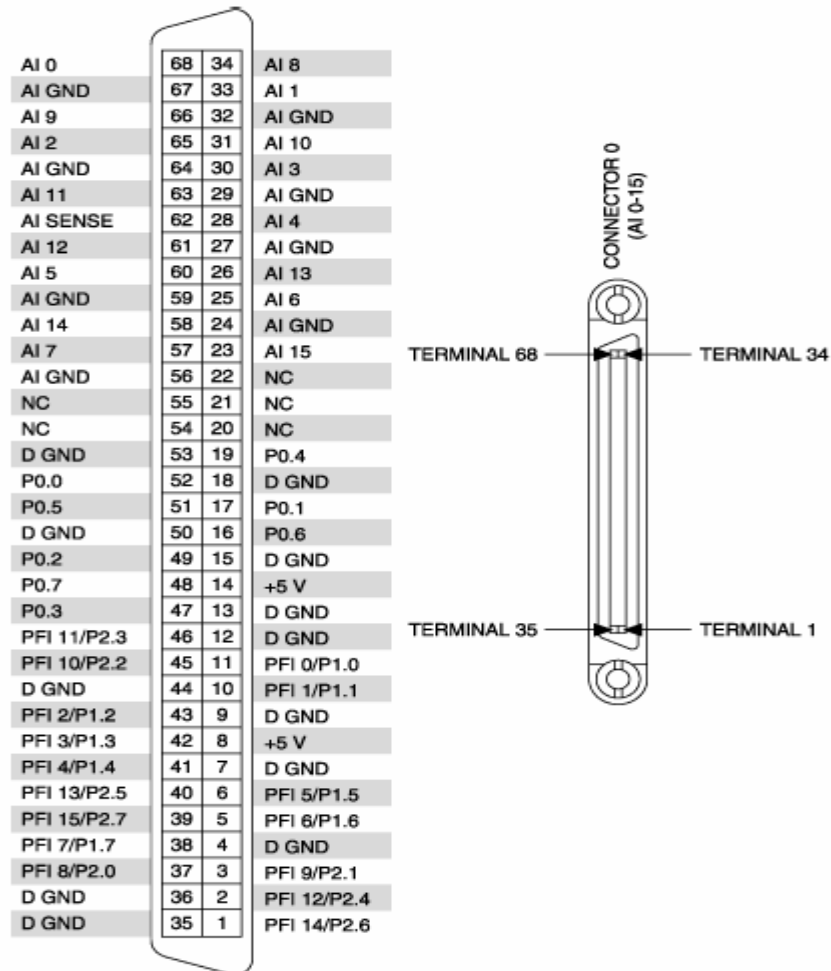


Figure 3.1.2: NI-6220 Data Acquisition System Pin-out

Our P300 Spelling Device, which will be using 10 channels, requires 11 I/O pins. Specifically, analog input channels from zero to nine (AI <0...9>) and analog input sense (AI SENSE) need to be connected to a *shielded cable box* (NI SCB-100) since each channel requires one analog input voltage channel for the single-ended measurements. In *non-referenced single-ended* (NRSE) mode, AI SENSE acts as the reference for each of AI <0...9> signal because the DAQ hardware measures the voltage of an analog input signal (AI <0...9>) with respect to AI SENSE.

### 3.2. Data Collection Module Software

There are two types of data that our P300 Spelling Device can process, i.e. pre-collected data from saved into a *.mat* file format (offline) or real-time data acquired from the user's scalp (online). Hence our data collection software must be able to accommodate both the online

and offline operation. Subsequent sections describe the online and offline software operation, including what they do and how they interface to the control module.

### 3.2.1. Data Collection Module Software—Online System

The online implementation of data collection module software interface with NI-6220 DAQ system via a driver called NI-DAQmx which provides NI-DAQmx library functions that can be used to develop instrumentation amplifications, data acquisitions and control applications. It carries out three essential tasks: initialization (*init()* function), conversion of input data into meaningful data block for specified channels (*Trigger()* function) and finally making these data available to the rest of the software (*getNextDataEpoch(Channel Number)* function).

Implementation of aforementioned task is heavily dependent on the overlap of subsequent P300 responses. Hence the data blocks would share data with another stimulus. The overlap is shown in figure 3.2.1.1.

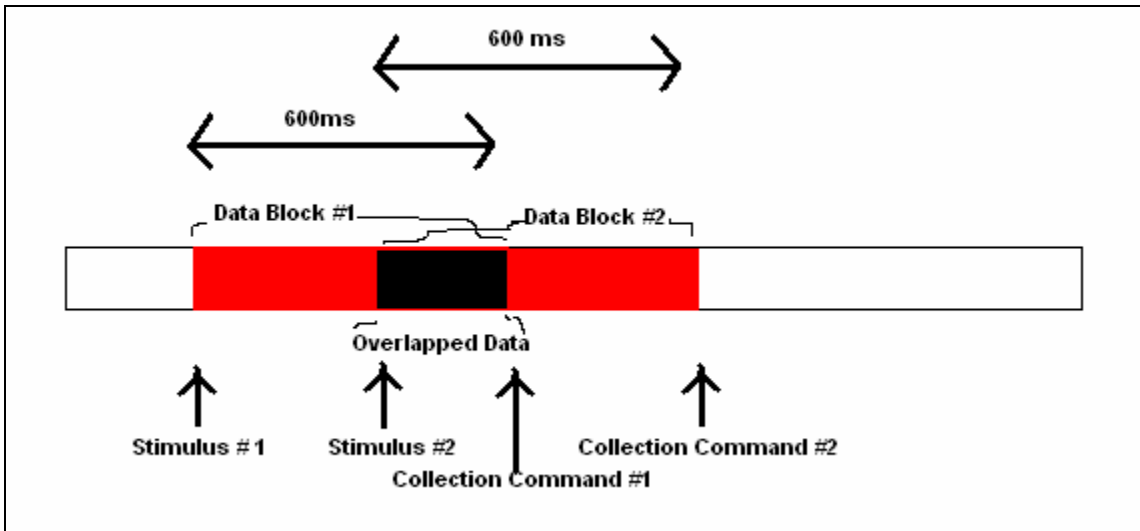


Figure 3.2.1.1: Data Overlapping in P300 Response

The online software module collects the most recent data collected in the previous 600ms. However, because of the overlapping in P300 response time, we must avoid deleting data in the buffer. Hence, we keep the data in the buffer so that specifying the collection mode to group the data by the channel number can access data blocks for each of the 10 channels. Figure 3.2.1.2 illustrates this concept.

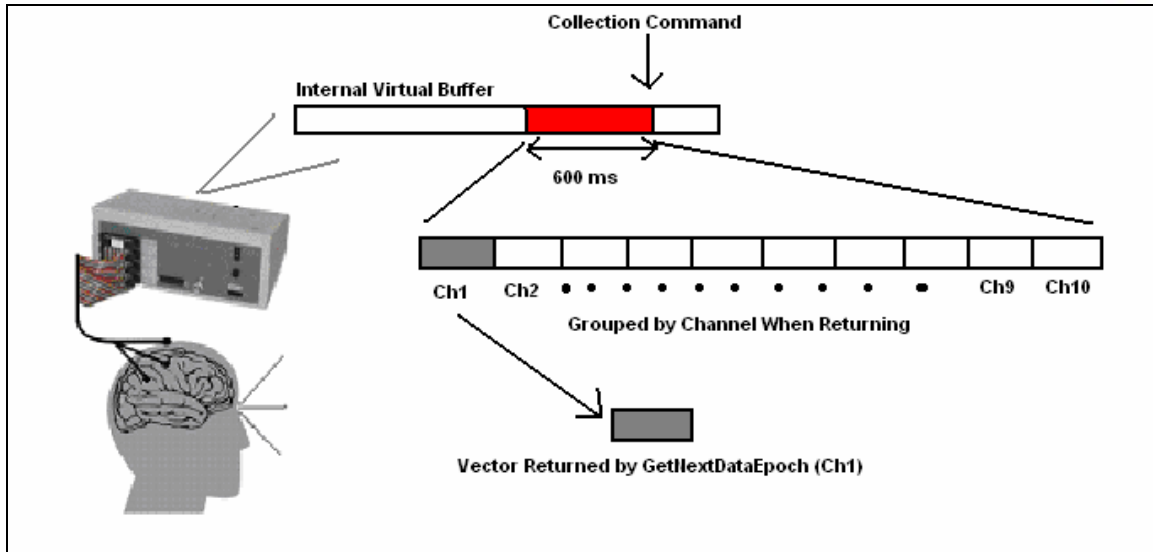


Figure 3.2.1.2: Data Flow in Online Data Collection Module

### 3.2.2. Data Collection Module Software—Offline System

In contrast to the online operation, the offline operation uses pre-collected data stored in *.mat* files. In the testing of our offline system, we used to the data available from BCI Competition 2003 [3]. Data collection module is responsible for extracting appropriate data and returning them to the control module when requested. Implementation is done entirely in MatLab because of its efficiency in dealing with matrix-based computations.

Access to the offline data collection block is achieved in two steps: 1) *init()* and 2) *getNextDataEpoch()*. *init()* initializes the data collection block to set appropriate parameters. *getNextDataEpoch()* returns the data epochs in the sequence of recorded values as well as the coordinates of the letter that was flashed.

## 4. Control and Output Module

The control and output module acts as the main control of the P300 spelling device. In particular, it coordinates flow of data between the data acquisition module and the P300 detection module. Also, it controls the timing of row or column flashing on the main display monitor.

### 4.1. *Control and Output Module Hardware*

The control and output module controls the flashing of letters on the output monitor. The monitor we will use is a generic 19-inch CRT monitor. In order to control the flashes, we require hardware timers that are provided with the Pentium 4 CPU. These hardware timers are essential to our software display controller and data acquisition module, which are all timing dependent.

There are different hardware timers that differentiate themselves based on their resolution, interrupt priority, and accessibility. Some of these timers also depend on the make of the CPU such as between an Intel or AMD manufacturer [4]. Since our functional specification states that our system is designed for a Pentium-based PC, we will discuss hardware timers applicative only to Intel Pentium processors.

The first class of hardware timers is a 32-bit, 1 ms maximum resolution counter that wraps around in 49.7 days. This timer is readable as well as accessible such that an event can be instantiated to evoke and interrupt [4]. The interrupt occurs when the counter matches a user-defined delay and is set at the highest possible Windows system priority. The accuracy of these timers is fairly accurate since a kernel call to set these timers will create a separate thread independent of our parent program [4]. Once the timer elapses and interrupts, the timer thread executes a callback function, which will be the address of one of our timer functions located in our program. Furthermore, the timer events can be set as one-shot or periodic firing. These kernel calls are located the Windows Multimedia library (winmm.dll). The library was originally created for critical graphic applications [4]. Thus, our program utilizes these high-priority timers for flashing letters generated by the display output module.

The second class of timers used in our system is of higher resolution. More specifically, this is a 64-bit counter with an approximate  $0.8\mu\text{s}$  resolution [4]. However, the high resolution trades off on the interrupt and event capabilities. These timers can only be read but cannot be instantiated. Furthermore, these timers are only available in the Windows kernel32 library (kernel32.dll). Hence, we only use these Kernel calls to profile our software to accurately time the execution overhead for certain sections of our source code. The purpose is to pinpoint computational bottlenecks located in our software. As a result, we can further optimize our code or explore alternate computation algorithms.



The following table lists the API functions available in their respective Windows library, which are used in our software to read or instantiate an event on the hardware timers [5].

**Table 4.1.1: Table of timer API functions**  
**Windows' Multimedia Library (winmm.dll)**

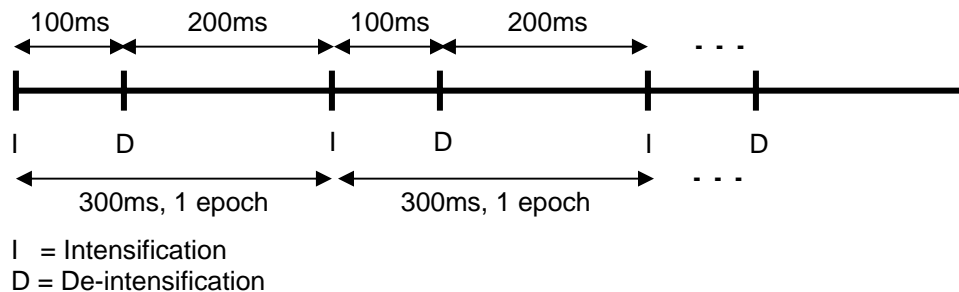
<i>API Function</i>	<i>Parameters</i>	<i>API Description</i>
timeGetTime	None	To read the timer counter value
timeSetEvent	(delay, counter increment resolution, address of callback function, timer ID, event firing flags)	To instantiate a timer event thread based on the delay and event firing type
timeKillEvent	(timer ID)	To properly kill the timer thread

**Windows' Kernel32 Library (kernel32.dll)**

<i>API Function</i>	<i>Parameters</i>	<i>API Description</i>
QueryPerformanceCounter	(Counter value)	Reads the high-resolution counter
QueryPerformanceFrequency	(Counter frequency)	Checks the counter frequency as it is CPU-make dependent

## 4.2. Control and Output Module Software

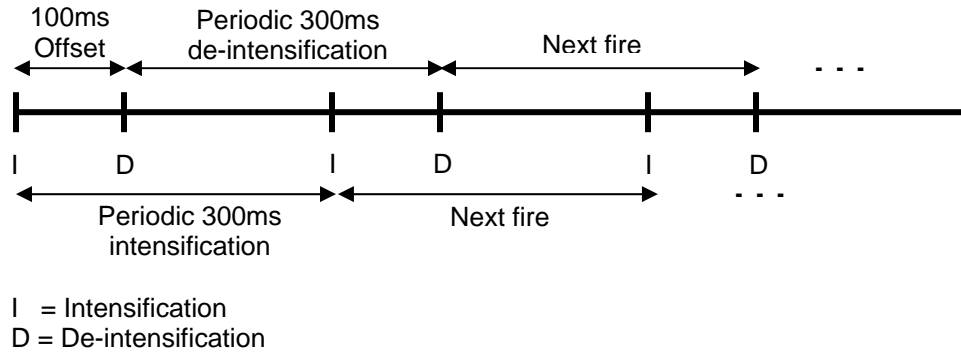
The display output of our design requires intensification and de-intensification of certain row or column of characters. Initially, we set the intensification and de-intensification period to 100 ms and 200 ms respectively. Thus, the total duration between consecutive flashes is 300 ms. This complete duration is considered as one epoch. It should be noted that these timing figures are not hard-coded but rather flexible numbers that can be set by the developer. But, based on the characteristics of the P300 response and pass researches, we chose these standard figures as initial settings for our system [6]. Figure 4.1.1 depicts the timeline of our display output.



**Figure 4.1.1: Row or column intensification and de-intensification for display output**

There are several approaches to implement a timing controller that confirm to the timing diagram shown above. The first solution is to instantiate two timers, one for 100ms and the

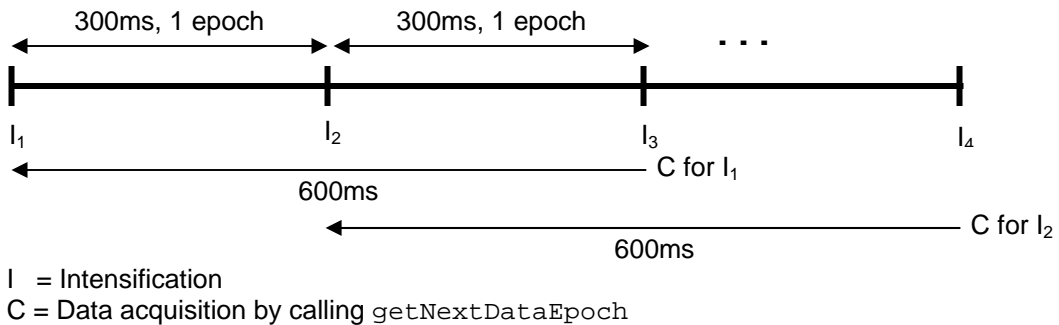
other for 200 ms. The disadvantage is that these timers can only be one-shot events instantiated and killed consecutively. As a result, this method has excessive API overhead that increases CPU usage and decrease dataflow output. Thus, our design resorts to another method which requires 3 timers, but 2 are set to periodic-event firing. Figure 4.2.1 is another representation of figure 4.1.1 depicted above but with a timing pattern that allows us to create periodic timers.



**Figure 4.2: Pattern for intensification and de-intensification timing**

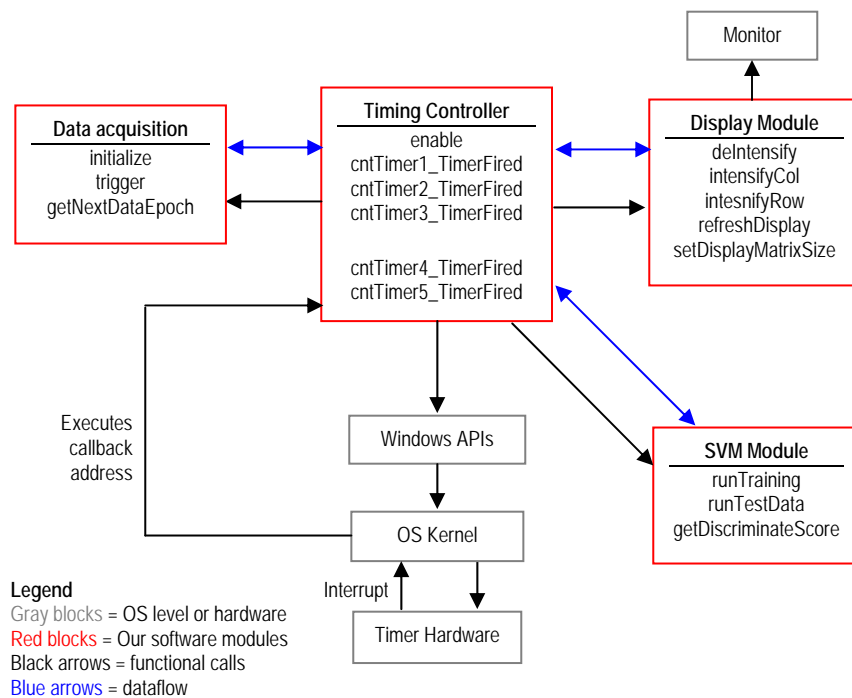
Based on the new figure, one timer has the task of intensifying a certain row or column and one timer has the task of de-intensifying the already intensified row or column. The third timer is a one-shot event to setup the first 100ms de-intensification offset. Once the third timer elapses, the two remaining timers will continue firing without excessive API overhead. The timers will be killed only at the end of user training or spelling.

More must be said about the data acquisition timers. Signal data from the DAQ A/D card must be collected 600 ms after a flash of certain row or column. A positive P300 response to that flash will have a positive signal within the 600 ms of data. Thus, the controller will require 2 more timers to accomplish this. The logic of 2 timers is the same as for the display timers. One is set as periodic and the other is set as one-shot for the initial offset. This manner again reduces API overhead. The following timeline illustrates the data acquisition trigger by calling the getNextDataEpoch function from the data acquisition module.



**Figure 4.3: Pattern for data acquisition timing**

The core module for coordination is the timing controller of our program. It coordinates the timing aspects of our system such as flashing the display, intermittently triggering the DAQ to collect data, and communicating with Windows APIs. Hence, the timing controller module is comprised of all timers mentioned above. The timing controller also facilitates dataflow of signal data, which will be passed to the SVM module for further processing. The following illustrates the coordination flowchart of the controller system and its associates. The black arrows indicate functional calls and the blue arrows indicate signal dataflow between two modules. Note that these modules can also be referred to as objects from a programming standpoint.



**Figure 4.4: Functional and signal dataflow flowchart for control and output modules**

The functional calls indicate normal method calls without transmitting signal data or SVM results. On the other hand, dataflow involves signal transmissions such as collecting raw data and passing off to other modules for processing. Within each module is a listing of descriptive object methods that are called by other modules to perform certain tasks. For example, when the de-intensification timer has fired and the appropriate callback function within the controller has been called, the controller will then call the `deintensify` method in the display module to clear the row or column currently intensified. Similarly, when one of the data acquisition timers has elapsed, it will call the `trigger` method and `getNextDataEpoch` method to prompt the DAQ to collect data from the A/D card and return the signal data back to the controller respectively. Another example is when the controller instantiates its timer by calling API methods which ultimately accesses the kernel and hardware timers. When, the hardware generates an interrupt, it will fire and jump to the callback address within our program for interrupt processing.

## 5. P300 Detection Module

The P300 detection module is responsible for detecting the P300 signal acquired from the user's brain. Because of the inherent noise associated with brain signals, it is difficult to isolate the P300 response. The advantage of our P300 detection module is that, it uses a machine-learning algorithm called *Support Vector Machine* (SVM), which allows a customized training profile for each individual. As such, P300 detection module works in two modes: 1) training mode and 2) testing mode.

### 5.1 Support Vector Machine Algorithm

The SVM algorithm is a machine-learning algorithm used for binary classification of the presence or the absence of P300 response. In order to accurately classify the input signals, it first needs to be trained with a large set of training data (sample data streams) with corresponding class labels (labels which indicate whether a P300 is present or not present). Using these two variables, the SVM algorithm constructs a hyperplane described by the weight vector  $\mathbf{w}$  and the bias term  $b$  as illustrated in Figure 5.1.1 [7]. The sign of this projection gives the prediction, positive indicating the presence of a P300 and negative the absence.

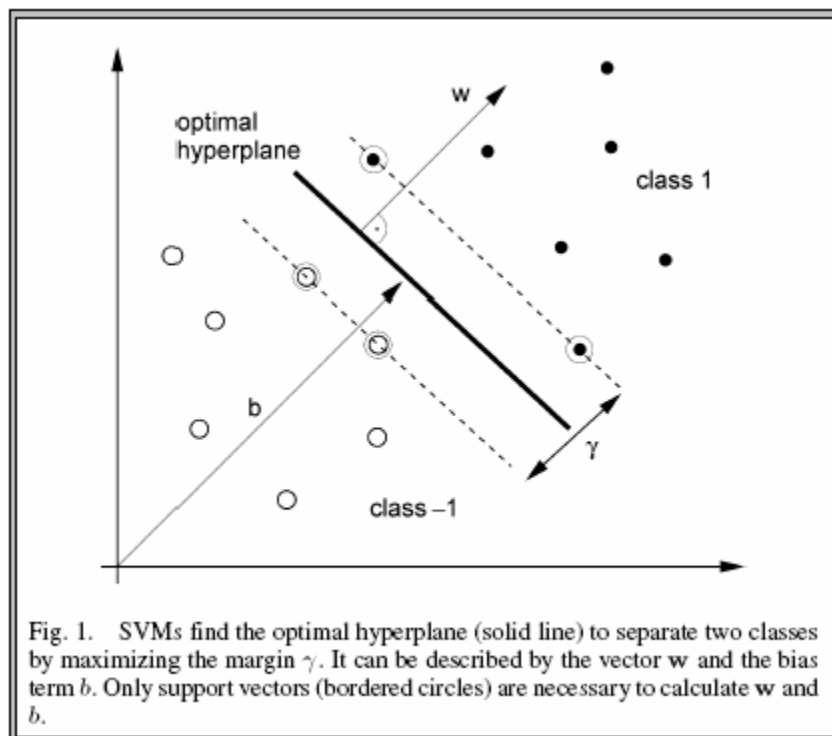
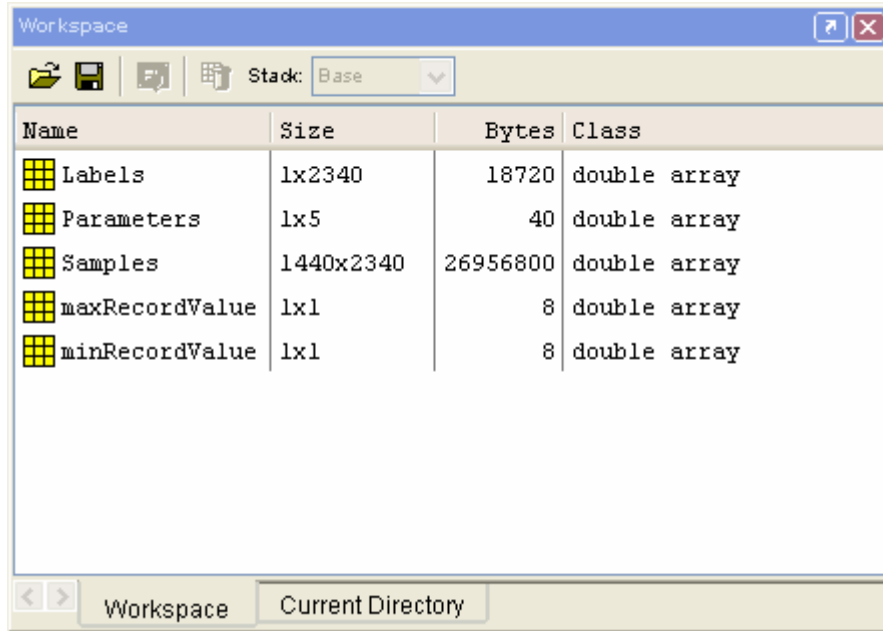


Figure 5.1.1: Optimal Hyperplane using SVM algorithm

Both the training data and class labels are in the form of *.mat* format used in MatLab and are acquired through the controller module. Figure 5.1.2 shows screen capture of the workspace containing training data sets.



Name	Size	Bytes	Class
Labels	1x2340	18720	double array
Parameters	1x5	40	double array
Samples	1440x2340	26956800	double array
maxRecordValue	1x1	8	double array
minRecordValue	1x1	8	double array

**Figure 5.1.2: Training Data Set**

Referring to figure 5.1.2, columns (2340 of them) of the *Samples* and *Labels* indicate the number of training data vectors and corresponding class labels (1 for P300 presence and -1 for P300 absence). The row (1440) of the *Samples* refers to the number of attributes in 10-channel implementation; each channel contributing 144 attributes. Once these two variables are put into MatLab workspace, *run\_train* script is executed which accesses the SVM library [8]. *run\_train* uses *Parameters* variable which defines options to be used in *mex* library including statistical distribution (Gaussian), classification type (linear), regularization parameter (20.007) and variance (27.359)<sup>1</sup>. It also searches for absolute maximum and minimum values in the training data in order to normalize the entire set thereby normalizing the statistical distribution and filter output between 0.5 Hz and 15 Hz.

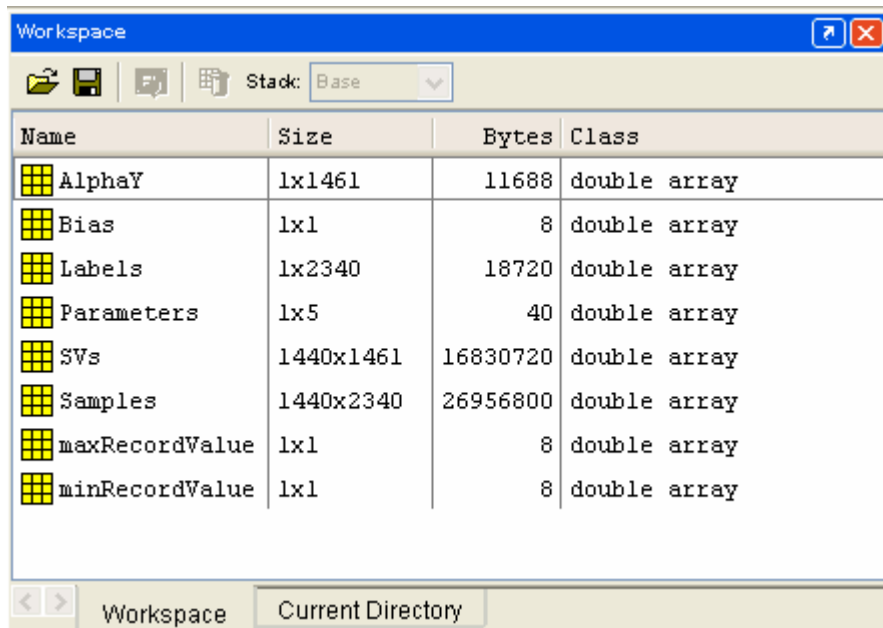
---

<sup>1</sup> Note that the values used for Regularization Parameter and the Variance are excerpt from Kaper et al. [7]. Values subject to change upon further investigation.

Variables	Description
Labels	Class labels corresponding to the data vectors (1 label by 2340 samples)
Parameters	Defines options to be used in <i>mex</i> library
Samples	Data vectors (144 attributes by 2340 samples)
minRecordValue	Absolute maximum value for normalization
maxRecordValue	Absolute minimum value for normalization

**Table 5.1.1: Variable Description**

Once the normalization is complete, *run\_train* calls the *mex:SVMTrain* function in the *mex* library within which the SVM parameters *AlphaY*, *SVs* and *Bias* are returned as shown in Figure 5.1.3.

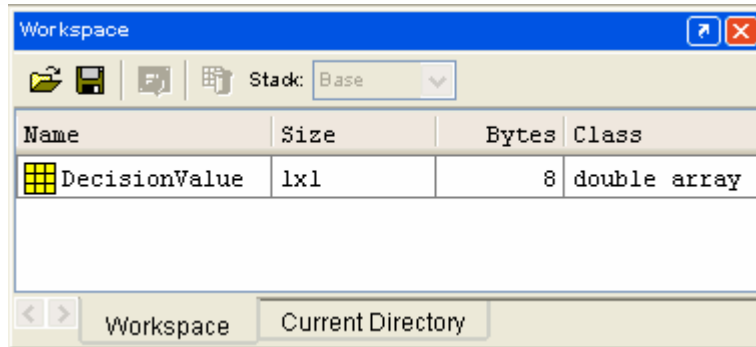


Name	Size	Bytes	Class
AlphaY	1x1461	11688	double array
Bias	1x1	8	double array
Labels	1x2340	18720	double array
Parameters	1x5	40	double array
SVs	1440x1461	16830720	double array
Samples	1440x2340	26956800	double array
maxRecordValue	1x1	8	double array
minRecordValue	1x1	8	double array

**Figure 5.1.3: Variables after Training**

The variables *AlphaY*, *SVs*, *Bias*, *maxRecordValue*, *minRecordValue* are saved as a model to be used in the test mode.

In test mode, which can be done in either offline or online scheme, the controller module places data vectors with unknown class labels and executes *run\_test* script. The *run\_test* script implementation follows the *run\_train* script very closely except for the fact that it calls *mex:SVMTest* instead of *mex:SVMTrain*. *mex:SVMTest* returns number of variables, one of which, is *DecisionValue*. *DecisionValue* is the variable, which represents the score for particular data vector(s) that were processed in the test mode. It is this variable that determines the presence or absence of P300 response for the corresponding data vector(s). Figure 5.1.4 shows the workspace after testing one data vector from 10-channel implementation.



**Figure 5.1.4: Variables after Testing**

Note that both *run\_train* and *run\_test* script interfaces with controller module via workspace only. All necessary input data must be present before executing either script.

## 6. Word Prediction Module

The word prediction module receives chosen letters from the control module and it returns a list of probable target words that the patient is trying to spell. When the word prediction module receives a letter, it appends the letter to the current letters in the word and searches through SQL database to find the most probable words that the patient is trying to spell. The SQL database is a list of the 1000 most commonly used words and is sorted according to rank. A sample of the 1000 most commonly used words is shown below in figure 6.1.

	Rank	Word
▶	1	the
	2	of
	3	to
	4	and
	5	a
	6	in
	7	is
	8	it
	9	you
	10	that
	11	he
	12	was
	13	for
	14	on
	15	are
	16	with
	17	as
	18	I
	19	his
	20	they

Figure 6.1: A database showing top 20 common used words



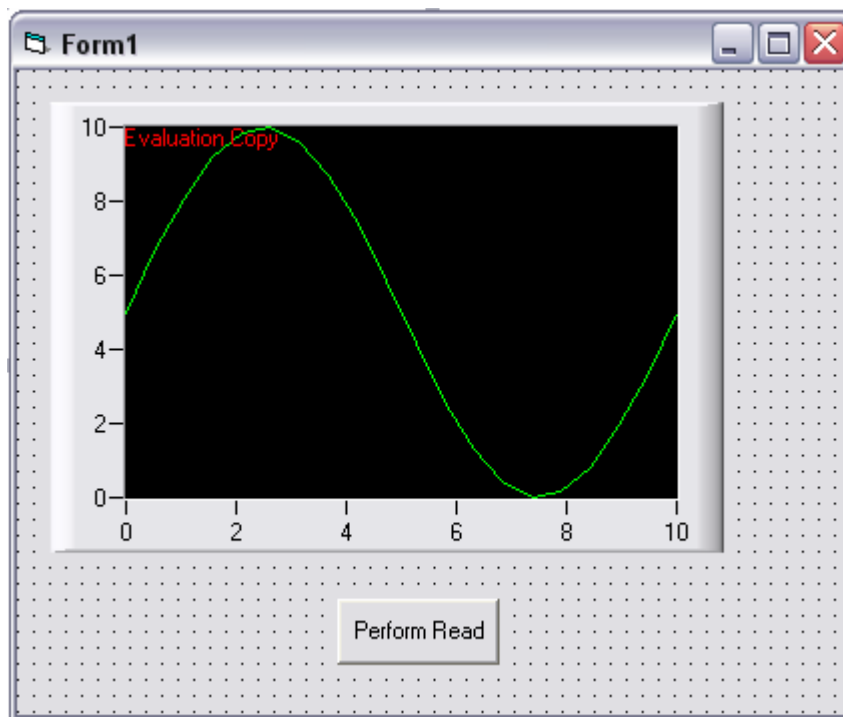
## 7. Testplan

The sections below outline how we will go about testing the individual blocks of our system to ensure that they function correctly. Once testing of all the blocks is complete, we can integrate the system and test with pre-collected sample data or with a user.

### 7.1. Data Collection Module Testing

Online Data Collection Module needs to be tested on its three functionalities which are to collect data, organize the data, and return the data (refer to section 3.2.1 for details).

Our test plan is to feed a sinusoidal signal to the DAQ Analog Input pins and display the data read by graphing them. The result of the testing will be shown by a graph interface as shown in Figure 7.1.1



**Figure 7.1.1: Graphing GUI for displaying Testing Result**

Once single channel testing is done, we will display all of 10 channels simultaneously to test the software's ability to return correct data blocks for all different channels.

The issue of overlapping data will be also tested which will be done by observing a sinusoidal signal at a sufficiently low frequency and observe the correct delay in each channel.

In order to test the offline data collection module, we will request data back from it and then visually inspect to see that it returned the correct values. Here, we will need to ensure that all border cases are covered.

## 7.2. Control and Output Module Testing

An indication of working operation by the control and output module is to test the timer firing events and execution of the callback procedure. Also, we can test if the timers fired at the delay specified by our system requirements. So, the 3 timers associated with the output display should fire either within 100 ms and 300 ms. Similarly, the 2 timers associated with data acquisition should fire within 600 ms. The testing for the controller module is fairly simple. We will place output status messages onto the GUI when any of the timer fires along with their timer IDs. Also, the interrupt controller within Windows kernel keeps track of firing delay referenced to when the timer was instantiated or reset. In other words, the kernel will jump to our callback procedure and pass an extra parameter that indicates the delay, in milliseconds, of when the timer fired since it was last instantiated for one-shot firing or reset in the case of periodic firing. Thus, just by looking at the status message, we can find out which timer fired and delay duration of the firing event. Figure 7.2 is a sample output of the status messages.

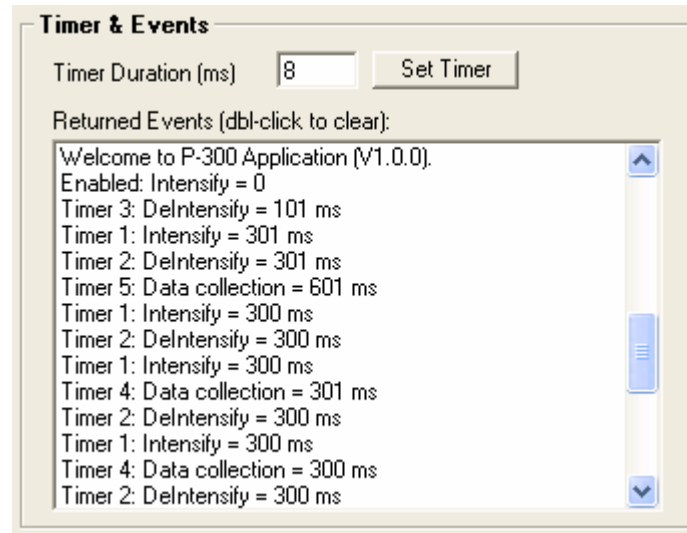


Figure 7.2: Output status messages for control timing testing

Again, these timing events are flexible and can be set by the user. But, the testing procedure remains the same and the only difference would be the returned delay parameter set by the kernel's callback handler.

### ***7.3. P300 Detection Module Testing***

To test the P300 detection module we will input known pre-collected data into the module and observe the output. In particular, we can train the module using pre-collected training data for which we know the correct response (i.e. P300 or no P300). Then we can input individual data epochs for which we also know the values. In general, the module will not return 100% accuracy; however, since we know the expected output (i.e. P300 or no P300), we can compare the actual result of our testing data with the expected result and be satisfied that we are getting the correct result when the correlation between the expected output and actual output is significant.

### ***7.4. Word Prediction Module Testing***

To test the word prediction module we will input a sequence of letters and verify that it returns the most commonly used words containing the input sequence according to rank. Verification can be done by visually inspecting the table of most commonly used words.

## 8. Conclusion

The design for our *P300 BCI Spelling Device* prototype is defined throughout this document, including the Data Collection Module, Control and Output Module, P300 Detection Module, and Word Prediction Module. We are currently in the process of implementing our spelling device and anticipate a functional prototype by April, 2005. During this time and possibly thereafter, we may choose to change certain aspects of the design in order to incorporate new functionality or technology.

## 9. References

- [1] Proposal for a P300 Spelling Device for the Completely Disabled. ThinQ Innovation.
- [2] Functional Specification for a P300 Spelling Device for the Completely Disabled. ThinQ Innovation.
- [3] <http://ida.first.fhg.de/projects/bci/competition/>. Set IIB data collected from the Wadsworth Center, NYS Department of Health.
- [4] A. Bestavros, A. Prezioso, “Windows Timing and Event Scheduling”, *Windows SRMS Service*, [http://www.cs.bu.edu/groups/realtime/SRMS-NT/event\\_scheduling.htm](http://www.cs.bu.edu/groups/realtime/SRMS-NT/event_scheduling.htm).
- [5] Microsoft, “Platform SDK Windows Multimedia”, *Microsoft Development Network*, July 2001.
- [6] V. Bostanov, “BCI competition 2003—Data sets Ib and IIB: Feature extraction from event-related brain potentials with the continuous wavelet transform and the t-value scalogram,” *IEEE Trans. Biomed. Eng.*, vol 51, pp 1057-1061, June 2004.
- [7] M. Kaper, P. Meinicke, U. Grossekhoefer, T. Lingner, and H. Ritter, “BCI competition 2003—Data Set IIB: Support Vector Machines for the P300 Speller Paradigm”, *IEEE Trans. Biomed. Eng.*, vol 51, pp 1073-1076, June 2004.
- [8] LIBSVM, open source found at <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>.