A Comparison of Tripolar Concentric Ring Electrodes to Disc Electrodes and an EEG Real-Time Seizure Detector Design

Xiang Liu
University of Rhode Island, xiang_liu@my.uri.edu

A COMPARISON OF TRIPOLAR CONCENTRIC RING ELECTRODES TO DISC ELECTRODES AND AN EEG REAL-TIME SEIZURE DETECTOR DESIGN

BY

XIANG LIU

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

ELECTRICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND

2013

DOCTOR OF PHILOSOPHY DISSERTATION

OF

XIANG LIU

APPROVED:

Thesis Committee:

Major Professor Walter G. Besio

Ying Sun

Zongqin Zhang

Nasser H. Zawia
DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND
2013

ABSTRACT

The electroencephalogram (EEG) is broadly used for research of brain activities and diagnosis of brain diseases and disorders. Although the EEG provides good temporal resolution, millisecond or less, it does not provide very good spatial resolution. There are two main reasons for the poor spatial resolution, (1) the blurring effects of the head volume conductor, and (2) poor signal to noise ratio. The surface Laplacian of the potential distribution was found to increase the spatial resolution. Several potential interpolation based methods were previously developed to estimate the surface Laplacian. However, these methods are generally complicated in terms of computation, which limits their real-time applications. Previously a special electrode, the tripolar concentric ring electrode (TCRE), was developed and proven to be a much simpler approach to estimate the surface Laplacian while achieving significantly better signal to noise ratio and approximation to the surface Laplacian. In the first part of the dissertation work, computer simulations comparing spatial resolution between conventional EEG disc electrode sensors and TCRE Laplacian sensors were performed. For verification of the computer simulations visual evoked stimulus experiments were performed to acquire visual evoked potentials (VEPs) from healthy human subjects. Analysis of the computer simulation results shows that the TCRE Laplacian sensors can provide approximately a ten-fold improvement in spatial resolution and pass signals from specific volumes. Placing TCRE sensors near the brain region of interest should allow passage of the wanted signals and reject distant interference signals. It was also shown that the TCRE VEPs appeared to separate sources better than disc electrode VEPs. In the second part, a tripolar EEG based automatic seizure detection algorithm was developed for rats, the paramters of the detector was optimized based on the recorded data. According to this algorithm, a Matlab based real-time detector was implimented and tested. In the last part of the dissertation, a prototype of FPGA based automatic seizure detector was Described, which has the ability to detect signal from many more channels real-time. An multi-channel EEG monitor system was also described.

ACKNOWLEDGMENTS

I would like to express the deepest appreciation to my advisor, Dr. Walter Besio, who has the attitude and the substance of a genius: he continually and convincingly conveyed a spirit of adventure in regard to research and scholarship. Without his guidance and persistent help this dissertation would not have been possible.

I would like to thank my committee chair Dr. Jyh-Hone Wang, my committee members Dr. Ying Sun, Dr. Zongqin Zhang and Dr. Resit Sendag for serving as my committee members and providing valuable ideas and suggestions for my dissertation. I would also like to express my appreciation to Dr. Kanthaiah Koka, Dr. Oleksandr Makeyev, Samuel Mucio, Dr. Hiram Luna-Munguía, Dr. Gabriela Rogel-Salazar, Yacine Boudria, Amal Feltane and Zhenghan Zhu for their help on my research during my Ph. D study.

Finally, I would like to express my appreciation and love to my wife Liling Wang, my parents Xinhua Liu and Heping Wang. Without their continuous support and encouragement, I won’t be able to finish my research and dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Electroencephalography

Electroencephalography (EEG) measures voltages from the neural activity of the brain. As a noninvasive method with high temporal resolution, EEG has clinical benefits in the diagnosis of brain related diseases and is useful in research. However, EEG suffers from poor spatial resolution due to the blurring effects primarily from different conductivities of the volume conductor [1].

To improve the spatial resolution the surface Laplacian, which is the second spatial derivative of the potential distribution on a surface, has been applied to EEG [1, 2]. The surface Laplacian is a high pass spatial filter, which sharpens the blurred potential distribution on the surface [2] and produces an image proportional to the cortical potentials.

There are generally two approaches to obtain the surface Laplacian. The first approach, referred to as the global surface Laplacian, is based on the construction of the potential interpolation equations on the surface [3]. The potentials from conventional disc electrodes have been utilized for the interpolation approach. The second derivative of the interpolation equations gives the global surface Laplacian. One of the major advantages of the global surface Laplacian is that it can encompass all points on the surface with a limited number of electrodes. A drawback of the global surface Laplacian is that the second derivative applied to the potential interpolation equations may not always be a valid estimate of the surface Lapclacian, it may produce distorted results [4, 5, 6, 7].

The second approach is the local surface Laplacian. Instead of applying the derivative to the global interpolation equations, the local surface Laplacian method approximates the surface Laplacian based on potentials from nearby electrodes. A typical example is Hjorth's [8] five point method, where the local surface Laplacian is obtained by calculating the difference of the potential on the electrode and the average potential on its neighboring four electrodes. The local surface Laplacian method does not rely on the second derivative of the interpolation equations, but it also has some drawbacks: 1) when the nearby electrodes are too far away, which is usually the case with the 10-20 system configuration, the resulting local surface Laplacian might not be a good approximation of the surface Laplacian [6], 2) the local surface Laplacian can only be estimated on the locations of electrodes but not from the edge electrodes. Although conventional disc electrodes could be used for this approach, there would still be the same limitations present such as poor spatial resolution and signal to noise ratio.

Previously Besio et al. developed a new EEG electrode structure, the tripolar concentric ring electrode (TCRE) [9]. The TCRE is made up of two concentric metal rings and a metal central disc layout on a flat printed circuit board (PCB). Both a conventional electrode and TCRE are shown in Figure 1. Due to this special structure, a linear combination of the potential from the three elements of TCREs directly forms the local surface Laplacian of the potential distribution [9]. We compared the TCREs with the conventional disc electrodes in both computer simulations and real EEG recordings. The results show that the TCREs are superior to conventional EEG electrodes on the surface Lapalcian estimation and spatial resolution.

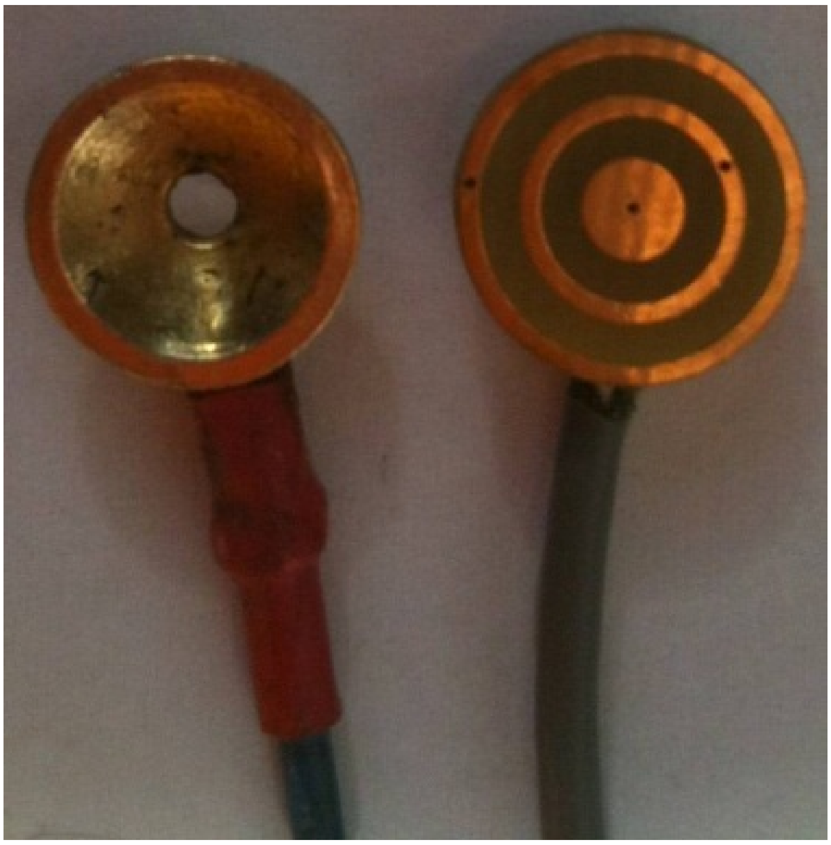


Figure 1 Disc electrode (left) and tripolar concentric ring electrode (right)

1.2 Epilepsy Seizure Detection

Epilepsy affects about 50 million people worldwide, and nearly 80% of them are living in developing countries. Anti-epileptic drugs have been successfully used to treat some patients. According to recent studies, up to 70% of the newly diagnosed children and adults with epilepsy can be successfully treated with anti-epileptic drugs. However, this implies that these drugs are not effective in about 30% of the patients.

In addition, the side effects of these drugs may reduce the quality of life of the patients. Surgery is another approach employed for epilepsy treatment, but it includes risks. Moreover, due to the relatively high cost of the two approaches above, about 75% of the patients in developing countries may not even receive treatment [10].

As a possible alternative, physical stimulation approaches have been gaining interest. Among these approaches, the implantable electrical stimulation approaches such as vagus nerve stimulation (VNS) [11], deep brain stimulation (DBS) [12] and responsive neurostimulation (RNS) [13], have been widely studied recently. Moreover, the VNS has even been approved by FDA in 2005 as a treatment for medication-resistant depression. Meanwhile, noninvasive stimulation methods have also been developed, such as transcranial direct current stimulation (tDCS) [14], and repetitive transcranial magnetic stimulation (rTMS) [15].

Detection of epilepsy is a necessary stage prior to epilepsy treatment. Generally, this is done through visual examination of recorded EEG signals by neural-physiologists or neurologists. However, there are several drawbacks: first, it's a time consuming process, especially for long-term EEG recordings; second, the cost is relatively high; third, it's not convenient, since patients have to stay in the hospital during the detection. Consequently, developing automatic epileptic seizure detectors becomes attractive. To the best of our knowledge, most of the automatic epileptic seizure detection methods are based on Electroencephalography (EEG).

There are mainly two approaches [16] [17]: time domain analysis and spectrum domain analysis. Time domain analysis mainly focuses on spike detection. A spike was defined by Gloor [18] as a triangle wave that is distinguished from the background signal, and has an amplitude at least twice that of the previous 5s of background activities of any EEG channel, and with a duration of at most 200 ms. Algorithms such as mimetic, linear predictive and neural networks [19] are mostly employed for spike detection for epilepsy. For the frequency domain, most works focused on detecting specific features related to seizures. Fast Fourier transform [20], time-frequency analysis [21], wavelet transforms [22], and nonlinear based analysis [23] are the most used methods.

Through the use of the high quality EEG signals recorded with TCREs, we developed a real-time automatic seizure detection algorithm based on the cumulative sum (CUSUM) detector [24]. The parameters of the detector were optimized by analyzing the recorded data from previous animal experiments. Due to the special structure, the TCREs have also shown the ability to perform focal electrical stimulations. Unlike the normal electrical stimulation via conventional disc electrodes that is usually applied across the head, the electrical stimulation via a TCRE is conducted between the outer ring and the central disc. Therefore, the stimulation current is focused on a small volume right underneath the TCRE. This type of electrical stimulations is called transcranial focal electrical stimulation (TFS). Previously we have reported the promising experiments applying TFS on rats, the results showed that TFS significantly reduced the highly synchronized brain activity within the beta and gamma bands at the early stages of PTZ-induced seizure development, also the number of rats survived after TFS significantly increased [25]. The combination of TCREs, automatic seizure detector, and TFS forms a closed loop seizure controller.

We also implemented the real-time seizure detector with automatic TFS triggering based on a laptop personal computer running our Matlab software control algorithm for a small number of channels [26]. Animal experiments were performed to verifying this detector.

1.3 Hardware Implementation of the Seizure Detector

We showed that the personal computer and Matlab based automatic seizure detector is suitable for rats [26]. But our ultimate goal is to develop an automatic seizure detector and controller for humans. The common EEG recording system for humans can be 20, 32, 64 or even 128 channels. Threfore, the seizure detector needs to have the ability to process the data from 20 or more channels. For practicality, we want the detector to be portable, so that patients can carry it with them. To meet these requirements above, we developed a field programmable gate array (FPGA) based embedded EEG recording and signal processing system which can be used as an automatic seizure detection system. Also, we further extended the design with a USB interface and the driver and application software in a PC to form a complete EEG recording system.

1.4 EEG recording system

The first human EEG signal was recorded by German physiologist and psychiatrist Hans Berger in 1924 using a galvanometer [27]. Albert Grass built the first commercial EEG systems, called Grass Model 1 in 1935 [27]. The Grass Model 1 was a three differential channel system built using vacuum tubes. With ever improving electronics technologies, EEG systems with more and more channels have become commercially available. A 16-channel system, Grass model III, was introduced at 1946. The transistor, which was invented in Bell Labs in 1947, made significant contributions to EEG system development. The systems designed with transistors were more reliable and stable with high gain while consuming lower power and space. Franklin Offner built the first transistor based EEG recording system in the 1950s [27]. Computer based EEG recording systems have become very common in the recent two decades with the rapid development of digital techniques. The analog EEG signal is digitized after amplification and transmitted to a microprocessor for further processing or storing. Digital techniques also provide EEG recording systems with new capabilities, such as long term EEG monitoring. Digital signal processing techniques have also been widely applied to EEG signals for brain related disease diagnosis. The emphasis of each EEG recording system design has been application specific, such as seizure detection or sleep monitoring.

CHAPTER 2: THE COMPARISON OF TRIPOLAR CONCENTRIC RING ELECTRODES TO CONVENTIONAL DISC ELECTRODES

This chapter presents a local surface Laplacian that overcomes the disadvantage of previous local surface Laplacian approachs by employing the TCRE introduced by Besio et al. [9]. Instead of utilizing nearby electrodes to estimate the surface Laplacian, the three elements of a single TCRE are used to calculate the surface Laplacian. To illustrate the advantages of the local surface Laplacian method by TCRE, the global surface Laplacian and local surface Laplacian are compared using a four layer concentric inhomogeneous spherical head model [28]. In the comparison, the global surface Laplacian estimation is based on the spherical spline interpolation method introduced by Perrin [3], while the local surface Laplacian estimation is based on the TCRE Laplacian algorithm [9]. Noise is added to the simulations to make the results more realistic.

2.1 Local Surface Laplacian Estimation Based On TCRE

The TCRE is shown in Figure 2. The electrode is made of three elements: outer ring, middle ring, and the central disc. The tripolar Laplacian is given by the combination of the potentials from the three elements of the TCRE [9]:

$$Surface\_Laplacian = -\frac{16\times(V_m - V_d)-(V_o - V_d)}{3R^2} \tag{2.1}$$

In equation (2.1), V_d denotes the potential from the central disc, V_m denotes the potential from the middle ring, V_o denotes the potential from the outer ring and R is the radius of the middle ring.

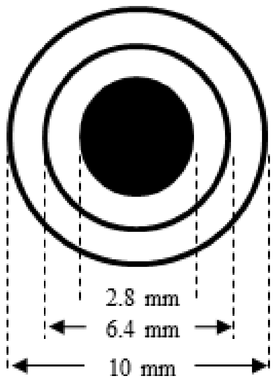


Figure 2 Tripolar concentric ring electrode (TCRE) sensor

In the real TCRE EEG recording (tEEG), a TCRE is connected to two amplifier channels: the disc is connected to the negative inputs of both channels, the middle ring and outer ring are connected to the positive inputs and then the signals are amplified and digitized and combined as in equation (2.1).

As previously mentioned, the local surface Laplacian usually suffers from the estimation made from the combination of the potentials of several nearby electrodes, thus the result may not be accurate if the density of the electrodes recording locations is too low and the electrodes are far apart. However, the TCRE overcomes this problem since each TCRE measures the surface Laplacian at its location. Further the surface Laplacian distribution can be easily calculated by interpolating the surface Laplacian from the Laplacian measured at the TCRE locations, thus the second shortcoming can also be improved.

2.2 Global Surface Laplacian Estimation Based On Spherical Spline Interpolation

The spherical spline interpolation method was introduced by Perrin et al. [3]. Perrin models the head as the surface of a sphere, which is not exactly the same as the shape of the human head, but approximates the head for comparison. The spherical model is commonly used in both research and clinical situations [29]. The equations described by Perrin et al. for the spherical spline interpolation are:

$$V(\boldsymbol{r}) = c_0 + \frac{1}{4\pi}\sum_{i=1}^{N} c_i \sum_{n=1}^{\infty} \frac{2n+1}{n^m (n+1)^m} \mathbf{p}_{\mathrm{n}}\left(\cos(\boldsymbol{r},\boldsymbol{r}_i)\right). \tag{2.2}$$

Where N is the number of electrodes, m is the order of the spline interpolation, $\boldsymbol{r}$ is the vector of the location where the potential is interpolated, $\boldsymbol{r}_i$ is the vector of the location of the i^{th} electrode, $\mathbf{p}_{\mathrm{n}}$ is the n^{th} degree Legendre polynomial. The parameters vector C is the solutions of equations (2.3) and (2.4):

$$GC + Tc_0 = Z, \tag{2.4}$$

$$T'C = 0. \tag{2.5}$$

Where $T' = (1,1,...,1)$, $C' = (c_1, c_2, ..., c_N)$, $Z' = (z_1, z_2, ..., z_N)$,

$G = (g_{ij}) = (g(\cos(\boldsymbol{r},\boldsymbol{r}_i)))$ G, $g(x) = \frac{1}{4\pi}\sum_{\mathrm{n}=1}^{\infty} \frac{2n+1}{n^m (n+1)^m} \mathbf{p}_{\mathbf{n}}(x)$.

The surface Laplacian operator in the spherical coordinates system is defined as:

$$\Delta_{surf} = \frac{1}{r^2 \sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta \frac{\partial}{\partial\theta}\right) + \frac{1}{r^2 \sin^2\theta}\frac{\partial^2}{\partial\phi^2} \tag{2.6}$$

Applying equation (2.6) to equation (2.2) gives the surface Laplacian of the spherical interpolation:

$$\Delta_{surf} V(\boldsymbol{r}) = -\frac{1}{4\pi r^2}\sum_{i=1}^{N} c_i \sum_{n=1}^{\infty} \frac{2n+1}{n^{m-1}(n+1)^{m-1}} \times \mathbf{p}_{\mathbf{n}}\left(\cos(\boldsymbol{r},\boldsymbol{r}_i)\right) \tag{2.7}$$

The truncated singular value decomposition method was applied to solve the inverse problem of the ill-posed matrix in equations (2.4) and (2.5) [30].

2.3 The Four-layer Spherical Head Model and the Analytical Surface Laplacian

Figure 3 shows a four-layer concentric inhomogeneous spherical model [28] to represent the human head. The four layers represent brain, cerebrospinal fluid, skull, and scalp. The corresponding radii of the layers are: 7.9cm, 8.1cm, 8.5cm and 8.8cm; the conductivities of the layers are: 3.3×10^{-3} S/cm, 10.0×10^{-3} S/cm, 3.3×10^{-3} S/cm, 4.2×10^{-5} S/cm, 3.3×10^{-3} S/cm, respectively. Current dipoles, described later, are employed to model the brain activity.

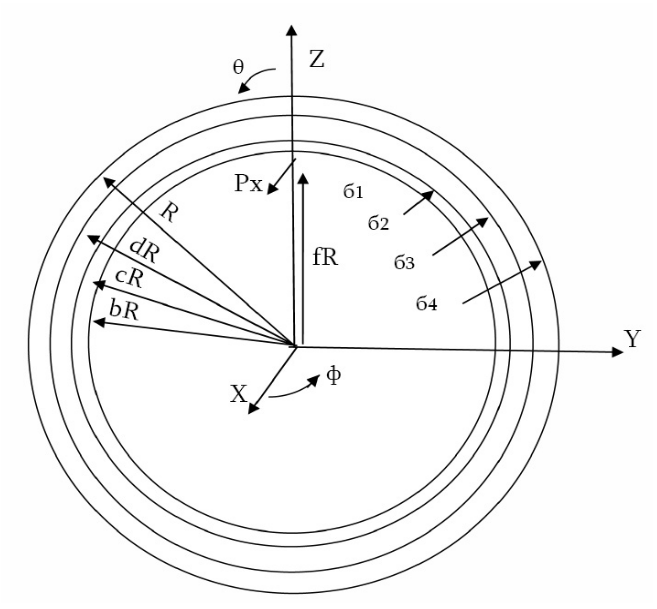


Figure 3 Four layer concentric inhomogeneous spherical head model

The potentials on the surface of the model due to a current dipole located at the z axis inside the brain pointing to x, y, z directions are given by the following equations [28]:

$$V_x = \frac{P_x \cos\phi}{4\pi\sigma_4 R^2} \sum_{n=1}^{\infty} \frac{(2n+1)^4 f^{n-1} (cd)^{2n+1} P_n^1(\cos\theta)}{n\Gamma}, \tag{2.8}$$

$$V_y = \frac{P_y \sin\phi}{4\pi\sigma_4 R^2} \sum_{n=1}^{\infty} \frac{(2n+1)^4 f^{n-1} (cd)^{2n+1} P_n^1(\cos\theta)}{n\Gamma}, \tag{2.9}$$

$$V_z = \frac{P_z}{4\pi\sigma_4 R^2} \sum_{n=1}^{\infty} \frac{(2n+1)^4 f^{n-1} (cd)^{2n+1} P_n^1(\cos\theta)}{n\Gamma}. \tag{2.10}$$

Where

$$\begin{aligned}
\Gamma &= d^{2n+1}\left\{b^{2n+1} n (k_1-1)(k_2-1)(n+1) + C^{2n+1}(k_1 n+n+1)\times(k_2 n+n+1)\right\} \\
&\times\left\{(k_3 n+n+1)+(n+1)(k_3-1)d^{2n+1}\right\}+(n+1)\times c^{2n+1} \\
&\times\left\{b^{2n+1}(k_1-1)(k_2 n+k_2+n)+c^{2n+1}(k_1 n+n+1)(k_2-1)\right\} \\
&\times\left\{n(k_3-1)+(k_3 n+k_3+n)d^{2n+1}\right\}, \\
k_1 &= \sigma_1/\sigma_2, \\
k_2 &= \sigma_2/\sigma_3, \\
k_3 &= \sigma_3/\sigma_4.
\end{aligned}$$

Applying the surface Laplacian operator equation (2.6) to equations (2.8), (2.9) and (2.10), the analytical surface Laplacian is given by:

$$\Delta_{surf} V_x = \frac{P_x \cos\phi}{4\pi\sigma_4 R^2} \sum_{n=1}^{\infty} \frac{1}{n\Gamma} \times \left\{(2n+1)^4 f^{n-1} (cd)^{2n+1} \times \left(-\frac{P_n^1(\cos\theta)}{R^2 \sin^2\theta} + \Delta_{surf} P_n^1(\cos\theta)\right)\right\}, \tag{2.11}$$

$$\Delta_{surf} V_y = \frac{P_y \sin\phi}{4\pi\sigma_4 R^2} \sum_{n=1}^{\infty} \frac{1}{n\Gamma} \times \left\{(2n+1)^4 f^{n-1} (cd)^{2n+1} \left(-\frac{P_n^1(\cos\theta)}{R^2 \sin^2\theta} + \Delta_{surf} P_n^1(\cos\theta)\right)\right\}, \tag{2.12}$$

$$\Delta_{surf}V_z = \frac{P_z}{4\pi\sigma_4 R^2}\sum_{n=1}^{\infty}\frac{1}{n\Gamma}\times\left\{(2n+1)^4 f^{n-1}(cd)^{2n+1}\Delta_{surf}P_n^1(\cos\theta)\right\}.$$

(2.13)

Where

$$\Delta_{surf}P_n^1(\cos\theta) = \frac{1}{R^2\sin^3\theta}\left\{P_n(\cos\theta)\left(n(n+1)^2\cos\theta\sin^2\theta-(n+1)\cos\theta\right)\right.$$
$$\left.+P_{n+1}(\cos\theta)\left((n+1)-n(n+1)\sin^2\theta\right)\right\},$$

and

$$\Delta_{surf}P_n(\cos\theta) = -\frac{n(n+1)}{R^2}P_n(\cos\theta).$$

By rotating the coordinate system, the analytical potential and surface Laplacian imposed by a dipole at an arbitrary location in the brain area can be computed according to equations (2.8) – (2.13).

2.4 Computer Simulation Methods

The computer simulation was conducted to compare the global spline surface Laplacian and the local TCRE surface Laplacian to the analytical Laplacian. To model the activities of the brain cortex area, ten dipoles with eccentricities of around 0.89 were utilized, which are listed in Table 1. The locations of the dipoles were modeled in the visual cortex area of the brain to compare the simulation results to those of actual visual evoked potential (VEP) recording experiments that we conducted. The moments of the first five dipoles had a radial direction, and the remaining five dipoles were at the same locations, but with a tangential direction. Table 1 lists all of the ten dipoles. In each simulation, one of the dipoles listed was selected as the signal source.

Table 1 Locations and the moments of the source dipoles

Dipole #	X (cm)	Y (cm)	Z (cm)	Moment
1	4.3	-5.3	4	RUD
2	6	-3	4	RUD
3	5	-4.6	4.1	RUD
4	-2.3	-4.4	6	RUD
5	-2.2	4.6	6	RUD
6	4.3	-5.3	4	TUD
7	6	-3	4	TUD
8	5	-4.6	4.1	TUD
9	-2.3	-4.4	6	TUD
10	-2.2	4.6	6	TUD

To simulate the potential recorded on the elements of the TCREs, we assume that there are 'sampling points' uniformly distributed on the surface of the electrode elements. The potential of all "sample points" on each specific element was calculated and the average of all the sample points for each specific element was considered the potential for that specific element of the TCREs. To determine the number of 'sampling points' necessary for stable calculations we examined the effect of the 'sampling points' density on the averaged potential. The higher the density of uniformly distributed 'sampling points', the closer the averaged potential is to the real potential. In our initial analysis we incrementally increased the density of 'sampling points' on the TCRE and compared the averaged potential. When the difference in potential due to adding more points was less than 0.1 percent we stopped adding 'sampling points'. For the conventional disc electrodes, we assumed its diameter was the same with that of the outer ring of the TCREs. We used the same process to find the 'sampling points', for the conventional disc electrode.

Three different noise conditions and four different electrodes configurations were considered in the simulation. The noise conditions were: (1) no additive noise, (2) with white Gaussian noise, and (3) with dipole noise (simulating brain activity not considered the brain source of interest). An environment without noise is not practical, but this study is still valuable as a base to reveal the effect of different types of noise to the surface Laplacian estimation methods. White Gaussian noise (WGN) was employed to simulate the noise from the environment and EEG recording equipment. For the conventional disc electrode, 20% WGN was added to the calculated potential on each electrode; for TCREs, 20% of the WGN was added separately to $\left(V_m - V_d\right)$ and $\left(V_o - V_d\right)$, since they physically are amplified with two separate circuits, as described in the previous section. The WGN level is defined as the ratio of the standard deviation of WGN and the standard deviation of the potential, to which the noise is added, over all the electrodes [31]. Moreover, the brain activity interference from the deep part of the brain was modeled as a noise dipole with eccentricity around 0.85. The four electrodes configurations are: (1) 19-electrodes, (2) 32-electrodes, (3) 64-electrodes, and (4) 128-electrodes. The 19-electrodes were placed at the standard 10-20 system locations. The 32-electrodes, 64-electrodes, and 128-electrodes locations were selected from the 5-5 system [32]. The global spline surface Laplacian and the local TCRE surface Laplacian were calculated at the locations of the electrodes and then compared to the analytical surface Laplacian using the correlation coefficient.

All the statistical analysis was performed using Design-Expert software (Stat-Ease Inc., Minneapolis, MN, USA). Full factorial design of analysis of variance (ANOVA) was used with four categorical factors [33]. The first factor (A) was the type of the electrode presented at two levels corresponding to conventional disc electrodes and tripolar concentric ring electrodes. The second factor (B) was the number of electrodes presented at four levels corresponding to 19, 32, 64, and 128 electrodes. The third factor (C) was the presence and type of noise presented at four levels corresponding to no noise, presence of white Gaussian noise (WGN), presence of a noise dipole, and presence of both WGN and the noise dipole. Finally, the fourth factor (D) was the dipole location presented at ten levels corresponding to 10 signal dipole locations from Table 1. The response variable was the correlation coefficient of the simulated surface Laplacian and the analytical surface Laplacian calculated for each of the 2*4*4*10 = 320 combinations of levels of four factors.

2.5 Computer Simulation Results

The correlation coefficients of the TCRE surface Laplacian and disc spline Laplacian to analytical surface Laplacian without any added noise are listed in Table 2. The averaged value and the standard deviation of the correlation coefficient in each column are listed at the third and second rows from the bottom of the table respectively.

Table 3 lists the correlation coefficient of the TCRE surface Laplacian and disc spline Laplacian to analytical surface Laplacian with 20 percent WGN added.

Table 4 shows the correlation coefficient of the TCRE surface Laplacian and disc spline Laplacian to the analytical surface Laplacian with the presence of a noise dipole. As we mentioned above, the electrical activity in the deeper brain was considered as another type of noise. This type of noise was also modeled as dipoles, but with smaller eccentricities, which means the brain source of the dipole is closer to the center of the head. In every simulation performed with this type of noise, a noise dipole with unit moment was randomly selected with the eccentricity of approximately 0.85.

In the last set of simulations, we considered both the 20 percent WGN and the noise dipoles and the results are listed in Table 5. Table 5 shows our most realistic simulation results. Correlation coefficient data obtained in this simulation for 320 combinations of factor levels is summarized in Table 6 (averaged for ten dipole locations).

The full factorial design of our study is presented in Table 6. We used the Box-Cox procedure to select the optimal power transformation improving the spread of studentized residuals [33].

The effect of factors A, B, C, and D on the correlation coefficient was assessed along with the effect of all possible two- and three-factor interactions. The effect of the four-factor interaction ABCD could not be evaluated. The ANOVA results suggest that all the factors and all of the assessed interactions have statistically significant effects in the model (d.f. = 238, F = 17.6, $p < 0.0001$) for the optimal power transformation of 2.81 determined using the Box-Cox procedure. The effects of the main factors were: A (d.f. = 1, F = 2736.5, $p < 0.0001$), B (d.f. = 3, F = 120.1, $p < 0.0001$), C (d.f. = 3, F = 34.7, $p < 0.0001$), and D (d.f. = 9, F = 10.3, $p < 0.0001$).

The ANOVA results show that, in particular, for the case of the factor A the tripolar Laplacian is significantly better than the spline Laplacian at approximating the analytic Laplacian.

A potential limitation of the current full factorial design is that we could not assess the effect of interaction of all four factors. Without replications including this interaction made the model over-specified with all the degrees of freedom being in the model and none assigned to the residual (error). On the other hand, adding replications to the design would be of limited value since all of the factor levels, except for the two levels of factor C involving stochastic WGN, are deterministic in nature so replicating the simulation for most level combinations would have yielded the same results. For the same reason randomization of the simulation run order would have been of limited value as well in our case even though in other cases it may help midigate the effect of nuisance factors [33]. Other assumptions of the ANOVA including normality, homogeneity of variance, and independence of observations were confirmed ensuring the validity of the analysis with no studentized residuals being outliers, i.e. falling outside the [-3, 3] range [33].

Table 2 Correlation in noiseless situation

Electrodes Config. / Source Dipole #	19 electrodes		32 electrodes		64 electrodes		128 electrodes	
	T	S	T	S	T	S	T	S
1	0.998	0.543	0.993	0.402	0.998	0.895	0.996	0.957
2	0.999	0.647	0.999	0.787	0.999	0.875	0.995	0.885
3	0.997	0.723	0.993	0.749	0.982	0.852	0.971	0.896
4	0.993	0.516	0.993	0.585	0.999	0.880	0.998	0.966
5	0.990	0.634	0.999	0.877	0.998	0.936	0.994	0.969
6	0.998	0.756	0.985	0.787	0.981	0.805	0.985	0.965
7	0.996	0.756	0.997	0.741	0.997	0.893	0.951	0.911
8	0.999	0.633	0.992	0.792	0.986	0.865	0.912	0.895
9	0.935	0.277	0.867	0.430	0.993	0.588	0.932	0.662
10	0.999	0.392	0.999	0.515	0.999	0.649	0.997	0.774

Table 3 Correlation coefficient with the presence of white Gaussian noise

Electrodes Config. / Source Dipole #	19 electrodes		32 electrodes		64 electrodes		128 electrodes	
	T	S	T	S	T	S	T	S
1	0.963	0.573	0.944	0.360	0.956	0.684	0.96	0.710
2	0.947	0.206	0.967	0.620	0.960	0.708	0.95	0.732
3	0.968	0.718	0.961	0.725	0.930	0.703	0.96	0.805
4	0.968	0.465	0.971	0.515	0.958	0.760	0.96	0.808
5	0.959	0.682	0.961	0.679	0.958	0.741	0.95	0.799
6	0.953	0.688	0.966	0.700	0.973	0.712	0.96	0.689
7	0.962	0.502	0.969	0.587	0.959	0.742	0.96	0.696
8	0.978	0.181	0.967	0.530	0.978	0.645	0.97	0.713
9	0.983	0.540	0.963	0.594	0.978	0.645	0.96	0.647
10	0.964	0.243	0.961	0.719	0.965	0.749	0.96	0.911

Table 4 Correlation coefficient with the presence of brain dipole noise

Electrodes Config. / Source Dipole #	19 electrodes		32 electrodes		64 electrodes		128 electrodes	
	T	S	T	S	T	S	T	S
1	0.782	0.209	0.840	0.199	0.880	0.882	0.99	0.959
2	0.676	0.035	0.968	0.776	0.969	0.875	0.97	0.959
3	0.999	0.794	0.986	0.885	0.953	0.842	0.99	0.950
4	0.883	0.284	0.871	0.493	0.994	0.837	0.99	0.946
5	0.715	0.552	0.990	0.653	0.927	0.784	0.97	0.787
6	0.937	0.714	0.882	0.751	0.991	0.784	0.99	0.920
7	0.982	0.675	0.992	0.630	0.987	0.849	0.98	0.915
8	0.993	0.111	0.987	0.819	0.981	0.920	0.99	0.914
9	0.959	0.696	0.735	0.519	0.977	0.604	0.99	0.920
10	0.916	0.589	0.980	0.469	0.887	0.568	0.97	0.806

Table 5 Correlation coefficient with the presence of white Gaussian noise and dipole noise

Electrodes Config. / Source Dipole #	19 electrodes		32 electrodes		64 electrodes		128 electrodes	
	T	S	T	S	T	S	T	S
1	0.913	0.542	0.924	0.745	0.959	0.689	0.95	0.752
2	0.938	0.200	0.965	0.678	0.957	0.814	0.96	0.806
3	0.968	0.727	0.965	0.832	0.943	0.743	0.96	0.756
4	0.953	0.404	0.845	0.575	0.961	0.757	0.96	0.776
5	0.961	0.663	0.941	0.598	0.922	0.691	0.94	0.650
6	0.926	0.607	0.914	0.691	0.950	0.713	0.96	0.741
7	0.942	0.600	0.962	0.709	0.976	0.841	0.96	0.878
8	0.976	0.224	0.958	0.647	0.974	0.695	0.96	0.678
9	0.923	0.484	0.965	0.638	0.958	0.743	0.95	0.714
10	0.974	0.296	0.946	0.662	0.948	0.638	0.93	0.857

Table 6 Full factorial design of analysis of variance and obtained response variable

Group averages for 10 levels of factor D (signal dipole location)	Categorical factors			Correlation between the simulated and the analytical surface Laplacians (mean ± standard deviation)
	A: Type of the electrode	B: Number of electrodes	C: Presence and type of noise	
1	Conventional disc	19	No noise	0.5882±0.1581
2	TCRE	19	No noise	0.9908±0.0196
3	Conventional disc	32	No noise	0.6669±0.1693
4	TCRE	32	No noise	0.9823±0.0406
5	Conventional disc	64	No noise	0.8242±0.1141
6	TCRE	64	No noise	0.9937±0.0073
7	Conventional disc	128	No noise	0.8885±0.0989
8	TCRE	128	No noise	0.9737±0.0311
9	Conventional disc	19	WGN	0.4801±0.2041
10	TCRE	19	WGN	0.9649±0.0104
11	Conventional disc	32	WGN	0.6035±0.1138
12	TCRE	32	WGN	0.9634±0.0074
13	Conventional disc	64	WGN	0.7095±0.0139
14	TCRE	64	WGN	0.9619±0.0411
15	Conventional disc	128	WGN	0.7515±0.0783
16	TCRE	128	WGN	0.9633±0.0050
17	Conventional disc	19	Noise dipole	0.4662±0.2787
18	TCRE	19	Noise dipole	0.8846±0.1186
19	Conventional disc	32	Noise dipole	0.6199±0.2052
20	TCRE	32	Noise dipole	0.9236±0.0877
21	Conventional disc	64	Noise dipole	0.7950±0.1177
22	TCRE	64	Noise dipole	0.9549±0.0424
23	Conventional disc	128	Noise dipole	0.9082±0.0904
24	TCRE	128	Noise dipole	0.9877±0.1334
25	Conventional disc	19	WGN + dipole	0.4752±0.0224
26	TCRE	19	WGN + dipole	0.9480±0.1864
27	Conventional disc	32	WGN + dipole	0.6780±0.0738
28	TCRE	32	WGN + dipole	0.9390±0.0376
29	Conventional disc	64	WGN + dipole	0.7329±0.0156
30	TCRE	64	WGN + dipole	0.9551±0.0611
31	Conventional disc	128	WGN + dipole	0.7614±0.0881
32	TCRE	128	WGN + dipole	0.9580±0.0097

2.6 Visual Evoked Surface Potential (VEP) Recording Experiment Setup

We recorded from 15 selected locations on the scalp over the occipital lobe visual cortex area from the standard 10-5 system with the TCREs, the locattions of the electrodes are shown in Table 7. A reference electrode and a ground electrode were placed on the forehead of the subjects. Before the recording, the scalp was first abraided with Nuprep, a mild abrasive cleanser, and then Ten-20 electrode paste was used to match impedances between the TCRE and the scalp. The impedances were measured and any TCREs with impedances above 5 Kohms were attached again. The TCREs were connected to the preamplifiers we developed, A Grass Technologies Comet AS40 amplifier and digitizer (Natus Medical, Grass Technologies West Warwick RI) was cascaded.. A PS60/LED photic stimulator was controlled by the Comet AS40. The pass-band of the filter was set from 1.0 Hz to 70 Hz and a sampling rate of 200 samples per second was used. The frequency of the PS60/LED photic stimulator was 2.0 Hz. The subjects were seated in a comfortable chair with their eyes approximately 4.0 cm from the photic stimulator. For each subject, we recorded about two-and-a-half minutes of EEG signals. There was approximately 30 seconds of baseline EEG, with no photic stimulation, and then approximately two minutes with the 2.0 Hz photic stimulation. The photic trigger signal was also recorded to synchronize epochs during ensemble averaging.

The analysis of the recorded EEG signals varied depending on the type of signals recorded. We used the outer ring of the TCRE as a disc electrode emulation (eEEG). Recorded data was segmented with the peaks of the LED control pulses. We utilized about 200 segments for every subject. The first 150 ms of data for each segement was then ensemble averaged to obtain the visual evoked potential (VEP). The peak value of the VEP signal on each electrode was employed for surface Laplacian mapping. For the eEEG from the outer ring of the TCREs, the spline interpolation and surface Laplacian methods discussed above were applied to calculate the spline surface Laplacian and map them to the surface of the spherical head model over the occipital lobe visual cortex area. For the TCRE EEG surface Laplacian, we simply applied the interpolation algorithm to map the recorded Laplacian values to the corresponding surface. The Matlab code for the surface Laplacian mapping is in Appendix A.

Table 7 Normalized Locations of the electrodes in the experiment (X,Y,Z are coordinates of the simulation)

Locations #	X (cm)	Y (cm)	Z (cm)
CP5	-0.896	-0.338	0.284
P3	-0.567	-0.677	0.469
Pz	0.000	-0.714	0.699
P4	0.566	-0.677	0.469
CP6	0.896	-0.338	0.284
P5	-0.741	-0.635	0.213
P6	0.741	-0.635	0.214
P7	-0.804	-0.586	-0.088
PO7	-0.584	-0.807	-0.070
PO3h	-0.287	-0.910	0.298
POz	0.000	-0.929	0.368
PO4h	0.286	-0.910	0.298
P8	0.804	-0.587	-0.088
O1	-0.307	-0.949	-0.047
O2	0.307	-0.949	-0.047

2.7 Visual Evoked Surface Potential (VEP) Recording Experiment Results

From Figure 4 we can see that the TCRE Laplacian sensor was able to separate VEP sources. In Figure 4A, of the spline Laplacian map from disc electrodes, in the top central area there is a red and orange area (marked with an arrow). In the same area of Figure 4B we can see the TCRE Laplacian sensor map shows that there were two distinct sources. Panel C and D of Figure 4 shows the normalized grand-averaged EEG VEPs and tEEG VEPs that these maps were made with. From Figure 4C and D we can see that there is a positive going wave at approximately 100 ms after the photic stimulation pulse.

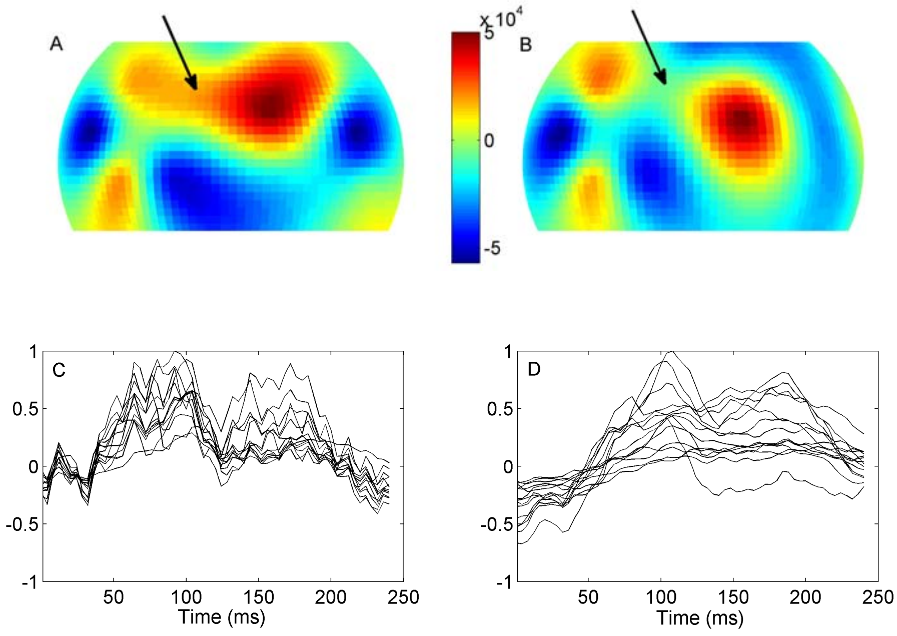


Figure 4 (A) Spline Laplacian VEP map, (B) tripolar Laplacian VEP map, and (C) the normalized grand-averaged EEG VEP signals from each channel, (D) the normalized grand-averaged tEEG VEP signals from each channel.

2.8 Disscusion of the Spatial Resolution Comparison

The computer simulation results show that with an increase in the number of electrodes, the spline surface Laplacian estimation has also been improved, while the TCRE surface Laplacian is not sensitive to the number of electrodes. The spline surface Laplacian estimation relies on the potential recorded on every electrode to optimize the interpolation parameters, therefore the more sensors leads to a better estimation of the parameters. On the other hand, each TCRE measures the surface Laplacian independently, as a result, the tripolar surface Laplacian does not rely on the number of sensors. After comparing the correlation coefficients in Table 2, it is apparent that at least up to 128 sensors, the tripolar surface Laplacian still outperforms the spline surface Laplacian.

The comparison of Table 2 and Table 3 shows that with the added 20 percent WGN the correlation coefficient of the spline surface Laplacian to the analytical surface Laplacian decreased by over 0.15 in the worst cases. In contrast, the correlation coefficient of the TCRE surface Laplacian to the analytical surface Laplacian only decreased less than 0.04 in all cases. The TCRE performed relatively constant regardless of the different number of sensors, while the performance of the conventional disc sensors dropped dramatically with the decreasing number of sensors. The results also indicate that the spline interpolation algorithm is more sensitive to random noise compared to the TCRE Laplacian algorithm.

Table 4 shows the results with the presence of the deep brain activity noise, which was also modeled as dipoles with smaller eccentricities. Both the spline surface Laplacian and TCRE surface Laplacian were affected by brain source dipole noise.

But in most cases the TCRE surface Laplacian still outperforms the spline surface Laplacian by 0.05 or more in terms of the correlation coefficient. Also, even though the statistical analysis for the 128-electrode configuration indicates that there is no significant difference in the spline surface Laplacian and TCRE surface Laplacian, in most cases the TCRE surface Laplacian still outperforms the spline surface Laplacian by 0.018 to 0.165. In real experiments it would usually be known from physiology what areas of the brain the signals should be coming from and those sensors could be preferentially treated while other locations could be less important.

In the last computer simulation, both 20 percent WGN and the dipole noise were added and the results are presented in Table 5. This simulation is the most realistic of our simulations. The results shown in Table 5 suggest that the TCRE surface Laplacian has at least two significant advantages compared to the spline surface Laplacian. First, the TCRE surface Laplacian works nearly the same with different numbers of electrodes (from 19 electrodes to 128 electrodes); second, the TCRE surface Laplacian is stable to different noise dipoles, which is due to its small half sensitivity volume (HSV), or, in other words, local recording characteristic.

In the simulation, the eccentricities of signal dipoles were set at around 0.9. This eccentricity was used since we were mainly interested in the visual cortex area of the brain. In a previous study [34], the eccentricities of the dipoles were usually set at 0.85 or smaller. We want to mention that the eccentricity of the dipole has considerable impact to the Laplacian estimation. Generally, the smaller the eccentricities the better performance of the spline and tripolar Laplacian estimation. In addition, the relative location of the dipole to the sensors is also an important factor regarding the Laplacian estimation. We observed in the simulation that if the dipole was close enough to one of the sensors, there would be a large difference between the analytical and the estimated Laplacian. This holds for both spline and the tipolar surface Laplacian estimation.

The VEP experiments showed that we can acquire VEP signals from humans. Beyond the acquisition we were able to see separate sources in the TCRE Laplacian maps that were not separated in the spline Laplacian maps, which is shown in Figure 4. It should be noted that we are not certain where the sources are in the visual cortex however, Fig. 4A and B are indicative of the other subjects as well.

2.9 Conclusion

From the computer simulations there is a significant improvement in estimation of the Laplacian using TCRE Laplacian sensors compared to disc electrodes and the spline Laplacian. The human experiments verified that we can record VEP signals using TCREs and that the tEEG signals showed two sources while the EEG signals showed only one source.

CHAPTER 3: THE DEVELOPMENT OF AUTOMATIC EPILEPSY SEIZURE DETECTION ALGORITHMS ON RATS

3.1 Introduction of the Seizure Detection

As we described in the first chapter, we have shown the effectiveness of TFS to control seizures in rats [25]. However, we waited until we observed the first strong behavioral change elicited from the convulsants, a myoclonic jerk (MJ), before we turned the TFS on manually. Observing our electrographic data from rats before and during seizures we hypothesized that the tEEG signal could be used to automatically trigger the TFS to control seizures. In reality we do not want to develop an automatic seizure control system for rats but took this opportunity to prove that a seizure warning, and or control, system could be developed utilizing the TCRE and TFS technologies.

In the previous chapter we introduced a unique electrode, the TCRE, and showed that the TCRE outperforms the conventional disc electrode in terms of surface Laplacian estimation. The TCRE has been proven to have several advantages compared to conventional disc EEG, such as better spatial resolution, higher signal to noise ratio, and less mutual information between nearby electrodes [35]. Due to the special structure, the TCRE also has shown the ability to perform focal electrical stimulations. Unlike the normal electrical stimulation via conventional disc electrodes that is usually applied across the head, the electrical stimulation via a TCRE is conducted between the outer ring and the central disc. Therefore, the stimulation current is focused on a small volume right underneath the TCRE. We call this type of electrical stimulations transcranial focal electrical stimulation (TFS). Beyond the focal stimulation the signal acquisition advantages suggest that the TCREs may have benefits for EEG based epileptic seizure detection. In this chapter, we present a real time automatic seizure detection system that was tested on rats.

3.2 Methods of Developing and Optimizeing the Automatic Seizure Detector

The TCREs signals were used to monitor brain activity and when a seizure was detected the TFS was triggered. To develop an efficient detector, we analyzed the recorded TCREs EEG (tEEG) signals from 5 rats, (the details of which are given below). In the tEEG signals we found a very stable pattern in the band of 0 to 100 Hz. The power spectral density of the seizure signals was higher than the non-seizure signal, which is shown in Figure 5. From Figure 5, the seizure and non-seizure situation is clearly separated. Figure 5 also suggests that seizures are usually accompanied by a significant change in the on-going electrical activity of the brain and therefore the power spectral change detectors are appropriate for seizure detection. However, in some more complicated situations, the EEG signal can be corrupted with noise, which makes the determination of the seizure a challenge. To make the detector more stable in noisy situations, we independently detected the power spectral density change in several sub-bands. The 1-100 Hz bandwidth is historically divided into

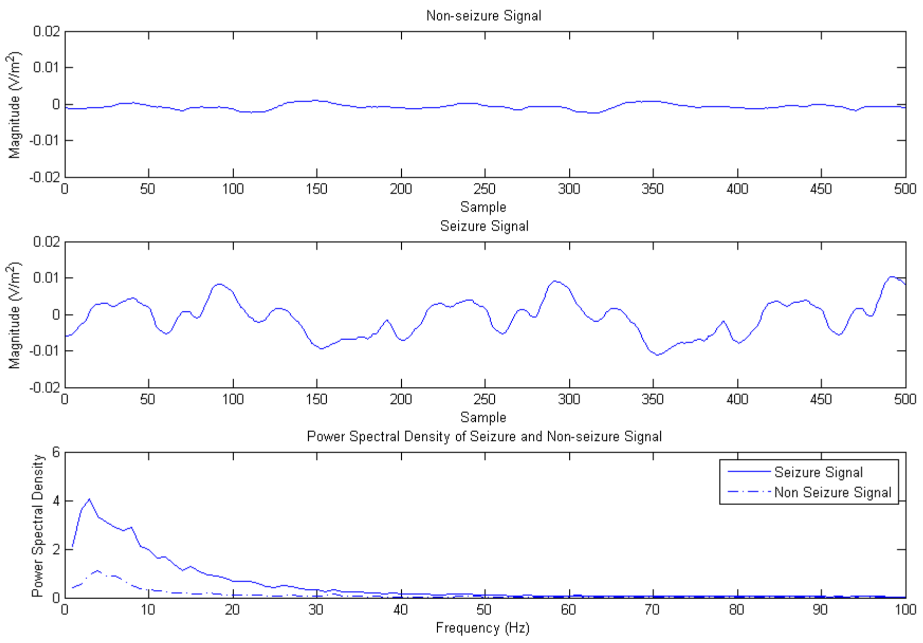


Figure 5 Power spectral density of seizure and non-seizure signal

several sub-bands in EEG research: Delta (0.3-4Hz), Theta (4-8Hz), Alpha (8-13Hz), Beta I (13-20Hz), Beta II (20-36Hz), Gamma (36-59Hz) and high Gamma (59-100Hz), which are shown in Figure 6. We employed this division of the 1-100 Hz bandwidth, except that we set the high Gamma band starting from 61 Hz instead of 5 9Hz to avoid the 60 Hz power line interference. We applied an independent detector for each of the sub-bands, and combined the detection results from all the sub-bands to make a decision about whether there was a real seizure. The cumulative sum (CUSUM) detector [36] was employed for real-time detection of the power spectral change for each sub-band.

Delta	Theta	Alpha	Beta I	Beta II	Gamma	High Gamma
0.3-4Hz	4-8Hz	8-13Hz	13-20Hz	20-36Hz	36-59Hz	59-100Hz

Figure 6 Sub-bands in EEG research

The CUSUM detector is an abrupt change detection algorithm. It determines whether a parameter θ in a probability density function (PDF) has changed. That is, to determine between two hypothesis: $H_0 : \theta = \theta_0$ and $H_1 : \theta \neq \theta_1$. Let p_{θ_0} and p_{θ_1} denote the PDF before and after the change, respectively. Let y_k denote the k^{th} sample of the data sequence (i.e. EEG segment). The basic CUSUM decision function is

$$g_k = \max(g_{k-1} + \ln \frac{p_{\theta_1}(y_k)}{p_{\theta_0}(y_k)}, 0) \qquad (3.1)$$

$$t_a = \min\{k : g_k \geq \bar{h}\} \qquad (3.2)$$

where $\bar{h}$ is a threshold. Here, t_a is the *stopping time*, when the detector identifies a change and raises an alarm. Each time when $g_k \geq \bar{h}$, the CUSUM detector restarts by setting $g_k = 0$ and a new round of detection begins.

When p_{θ_0} is a Gaussian process with mean μ_0, p_{θ_1} is a Gaussian process with mean μ_1, and both have variance σ^2. and equation (3.1) detects a mean change and becomes

$$g_k = \max(g_{k-1} + (y_k - \mu_0 - s), 0) \qquad (3.3)$$

Seizure and non-seizure data was recorded and analyzed to optimize the parameters for the CUSUM detector. The hardware connection for the signal recording is shown in Figure 7. Four TCREs were placed on the head of the rats, among which a1, a2 and b are signal electrodes, c was used as a reference electrode. A multiplexer were employed to switch these electrodes to pre-amp (Pre-Amp), stimulator (Stim) or impedance meter (Imp-Meter). On the signal channels, tEEG signals were pre-amplified (gain 100 and 0.3 Hz high pass filter) with a custom built preamplifier and then amplified using a Grass Model NR2 Neurological Research System with AC amplifiers at a gain of 1,000, bandwidth of 1.0–100 Hz, and with the 60 Hz notch filter on. Finally the signals were digitized at 16 bits and 256 samples per second (SPS)with a Measurement Computing USB-2537. Approximately 24 hours before the induction of seizures, an adult male 220~320g Sprague-Dawley rat was given a combination of 80 mg/kg of ketamine and 12 mg/kg xylazine for anesthesia. The scalp was shaved and prepared with NuPrep abrasive gel. Four TCREs were applied to the scalp using Ten-20 conductive paste and adhered with Teet's dental acrylic at the locations shown on Figure 7. On the following afternoon the rats were placed in a transparent plastic cage and the electrodes were connected via a commutator and cables. The skin-to-electrode impedance was measured to ensure that the impedance for the outer ring and the central disc of electrode (b) was less than 10 KΩ. The tEEG and video recording were then started. For the pentylenetetrazole (PTZ) treated group, after five minutes of baseline tEEG recording a mixture of PTZ and distilled water was administered (55 mg/kg, ip), then the seizure detection recording started. The recording lasted 30 minutes for each rat, and 5 rats were utilized for data collection.

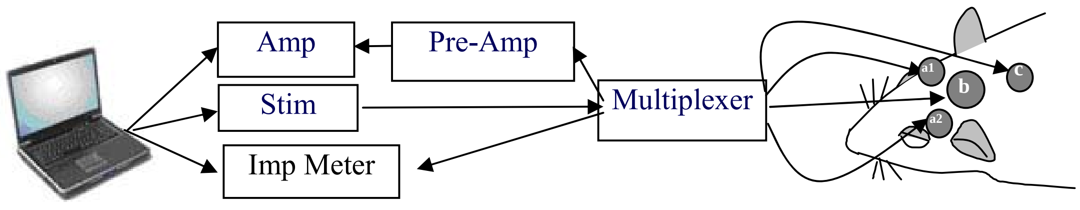


Figure 7 Hardware connection of the data acquisition system

To initialize the detector we used a one-second long non-overlapping Hanning window (256 samples) to segment the baseline tEEG signals. Then the power spectrum was calculated using the fast Fourier transform (FFT) with the "fft" command in Matlab (Mathworks, Natic MA). The spectrum was divided into sub-bands that we described above. For each sub-band the spectrum was summed over all frequencies of that sub-band and was normalized by the average baseline spectrum.

The baseline data from five rats was used to determine μ_0 and the threshold value $\bar{h}$ of the CUSUM detector. μ_0 is the baseline sub-band spectrum average. The value of $\bar{h}$ is determined by the equation: $\bar{\mathrm{h}} = \mu_0 + \beta \mathrm{x}$, where x is the standard deviation of the sub-band spectrum, β is the value determined by analysis of the recorded data. And s controls the sensitivity of the detector. We tested β from 0 to 10 in 0.1 increments and found that β= 0 provided the best seizure detection rate for the 5 controls. Parameter *s* was determined by adjusting *s* from 0 to 1000, in increments of 100, maximizing the true positive (TP) and minimizing false positive (FP) rates. To increase the likelihood that we discriminated seizure from movement artifact we implemented a two-of-three 'seizure' epoch smoothing algorithm: the third epoch was considered the seizure onset if two epochs in three consecutive epoches were marked by the CUSUM detector for power change. For each sub-band of the seizure data, we applied a CUSUM detector to detect the sudden power change. Table 8 shows the optimized parameters and detection results for each sub-band. According to the table, the Delta band (0.3-4Hz) and Theta band (4-8Hz) were the two most reliable sub-bands as seizure indicators.

The real-time automatic seizure detection algorithm was implemented with Matlab. In the implementation, Matlab interacts with the data acquisition hardware (Measurement Computing USB-2537) through a device driver that was embedded in Matlab. For initialization, the hardware was configured to sample 6 channels at 256

SPS. In the recording, a timer callback function was triggered every second to retrieve data of 1536 samples, which is for 256 samples per channel and 6 channels, from the data acquisition hardware. Figure 8 shows a diagram of the procedure which is described below. The retrieved data was further processed by the following steps: first, the data of the two channels from the same electrode was combined according to the formula (2.1), which forms three tEEG data series; second, the Hanning window was applied to the three data series; third, FFT transform was applied to the three data series; fourth, the spectrum sub-band power was utilized to optimize the parameters of the CUSUM detector if in: the (1) baseline session, or (2) passed to the CUSUM detector for sudden change detection if in the detection session; fifth, once the seizure activity was detected, an alarm was triggered, so that we can manually applied the TFS to the rat under seizure detection.

3.3 Real Experimental Testing Resutls

Figure 9 shows typical processed data for a TFS-treated rat. Trace #1 is the tEEG from electrode (b) in Figure 7. Traces #2 and #3 are the relative power and seizure detector output for the Delta band, respectively. Traces #4 and #5 are the relative power and seizure detector output for the Theta band. In Traces #3 and Traces #5, detection results are denoted by values ‘0’ and ‘1’, where ‘0’ means no seizure was detected, ‘1’ means seizure was detected. The two of three smoothing algorithm, which was described previously, was employed for the final detection decision. The vertical dashed line shows the time when the seizure was detected. In this experiment, the first myoclonic jerk was observed at approximately 2 minutes and 10 seconds. The averaged performance of the automatic seizure detector is listed in table 9.

Table 8 Parameters for CUSUM detector and results

Rat	Band	μ_0	$\bar{h}$	s	start epoch
	0-4Hz	0.0701	0.2102	0.1	7
	4-8Hz	0.0371	0.1112	0.1	6
	8-13Hz	0.0239	0.0718	0.15	8
1	13-20Hz	0.0362	0.1086	0.1	13
	20-36Hz	0.023	0.069	0.3	ND
	36-59Hz	0.0241	0.0724	0.3	ND
	61-100Hz	0.1139	0.3417	0.1	4
	0-4Hz	0.1652	0.4955	0.1	19
	4-8Hz	0.1010	0.3030	0.1	19
	8-13Hz	0.0914	0.2742	0.15	21
2	13-20Hz	0.0553	0.1658	0.1	22
	20-36Hz	0.0404	0.1211	0.3	ND
	36-59Hz	0.0730	0.2190	0.3	ND
	61-100Hz	0.2121	0.6363	0.1	24
	0-4Hz	0.1389	0.4168	0.1	6
	4-8Hz	0.1198	0.3595	0.1	ND
	8-13Hz	0.0688	0.2063	0.15	ND
3	13-20Hz	0.0455	0.1365	0.1	ND
	20-36Hz	0.0722	0.2165	0.3	ND
	36-59Hz	0.0742	0.2226	0.3	ND
	61-100Hz	0.1386	0.4157	0.1	ND
	0-4Hz	0.0362	0.1085	0.1	12
	4-8Hz	0.0492	0.1476	0.1	5
4	8-13Hz	0.0430	0.1289	0.15	5
	13-20Hz	0.0388	0.1165	0.1	11
	20-36Hz	0.0376	0.1127	0.3	12
	36-59Hz	0.0371	0.1112	0.3	12
	61-100Hz	0.1165	0.3495	0.1	4
	0-4Hz	0.1044	0.3131	0.1	13
	4-8Hz	0.0984	0.2951	0.1	17
	8-13Hz	0.0820	0.2460	0.15	ND
5	13-20Hz	0.0784	0.2353	0.1	15
	20-36Hz	0.0680	0.2041	0.3	ND
	36-59Hz	0.0789	0.2368	0.3	ND
	61-100Hz	0.1400	0.4199	0.1	ND

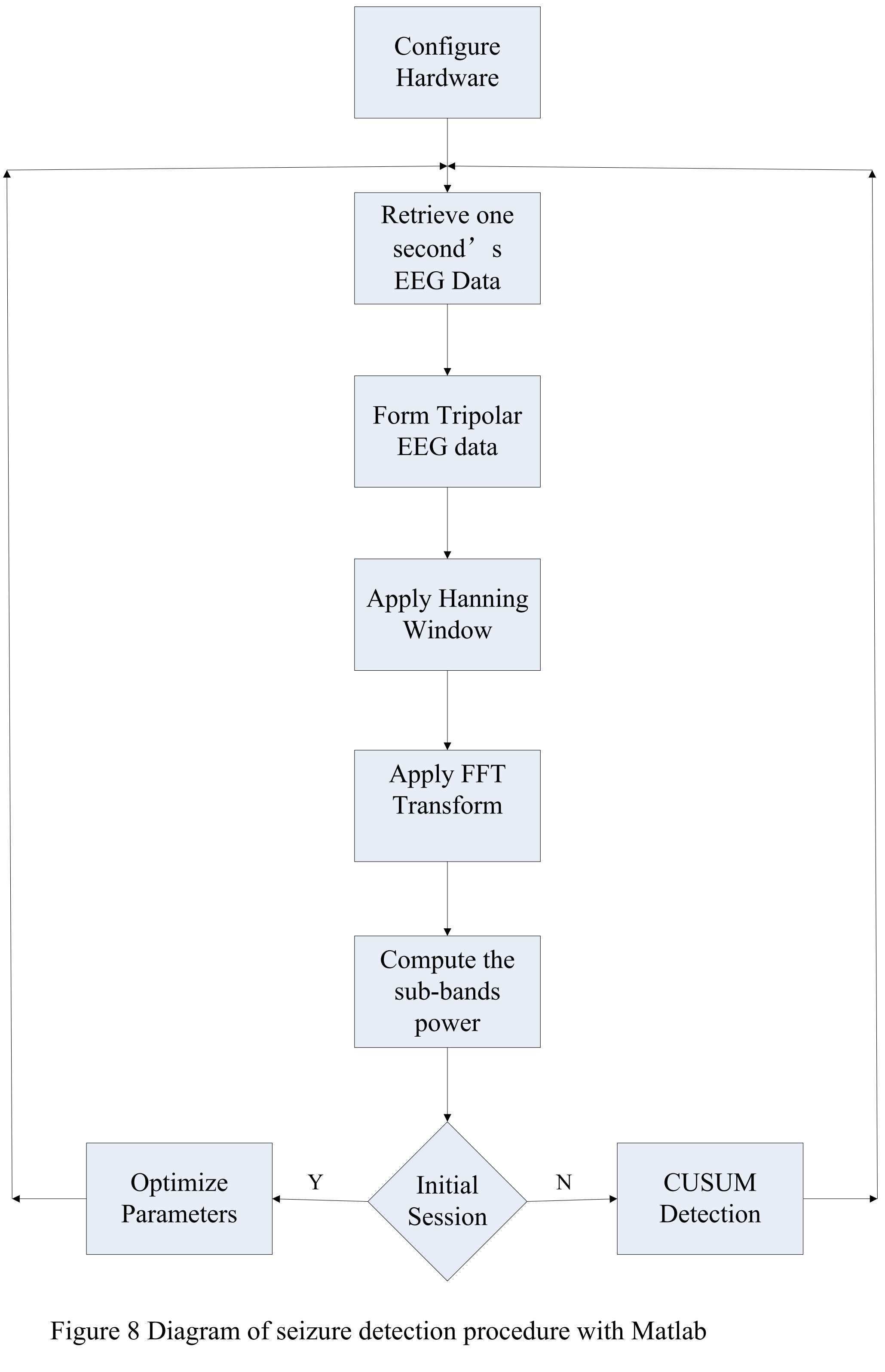


Figure 8 Diagram of seizure detection procedure with Matlab

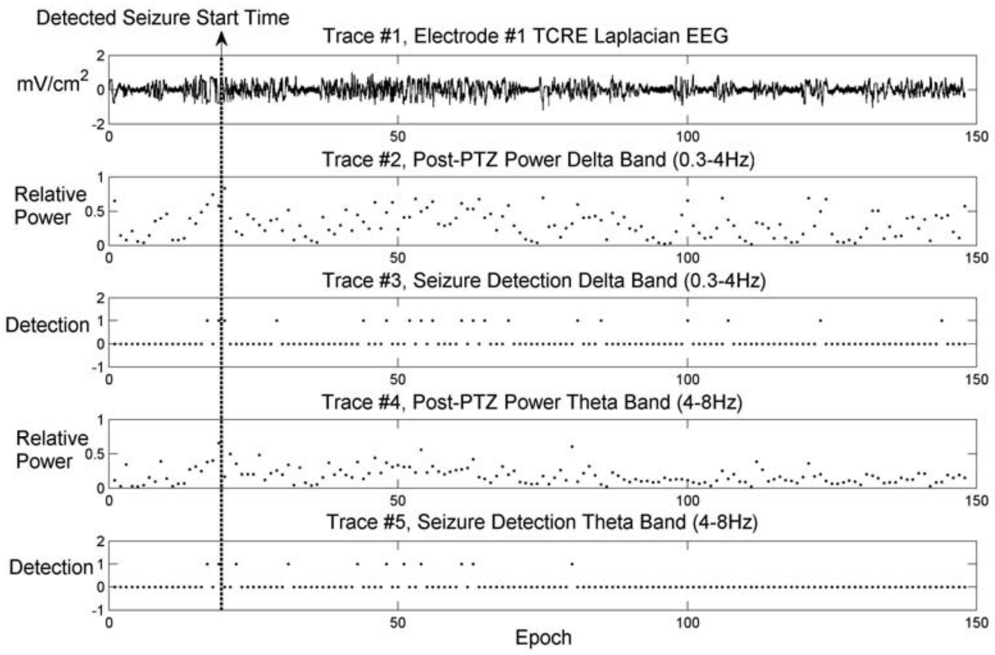


Figure 9 Automatic seizure detection result

Table 9 Performance of automatic seizure detector

	Accuracy (%)	Sensitivity (%)	Specificity (%)
CUSUM	74.47	23.06	91.9

3.4 Discussion

We were able to 'train' our CUSUM detector (i.e., to select the *s* parameters) using the control rat data and apply those parameters to test the detector on data that were not used for training (the generalization property of the CUSUM algorithm). The μ_0 and $\bar{h}$ parameters were chosen from the baseline TCRE EEG for each rat via specific algorithms removing user bias of the selection. The detector determined the seizure onset in the TFS-Treated rats, on average, 79 sec. (STD 43.12 sec.) prior to the first myoclonic jerk.

Much work has been performed in the field of seizure detection [36, 37, 38, 39]. For our experiments we have a special case where we know when the convulsant is given after a baseline period. We do not need to resolve long periods of baseline activity vs. seizure activity. For these experiments we were only interested in determining when the TCRE EEG showed increased activity due to the PTZ. We did not need to discriminate False Positives, 'seizure' during baseline, only during a short period post PTZ. The rest of the data is known 'seizure' data and therefore we only had to discriminate True Positive and False Negative (no 'seizure' during 'seizure'). Although using combinations of bands may be more robust for detection our data suggest that the Delta power in the on-going EEG may be most informative in this regard. This suggestion needs further confirmation in subsequent studies.

3.5 Conclusion

The CUSUM algorithm, in conjunction with TCRE EEG, correctly detects seizure activity from the Delta power changes in advance of the early behavioral manifestations of a seizure (such as MJs). Therefore, this algorithm can be used as a control signal to automatically trigger TFS with the goal to prevent seizure development.

CHAPTER 4: EEG RECORDING SYSTEM AND REAL TIME AUTOMATIC SEIZURE DETECTOR

4.1 Introduction

In the previous chapter we introduced a real time seizure detection algorithm for rats. We also implemented this algorithm in “real-time” with Matlab running on a Dell D630 laptop. The experiment showed that this detector works very well for the rat experiments. However, several drawbacks existed in the seizure detection system described above. First, the central processing unit usage of the laptop was reaching 100% while performing the real-time control with just three TCREs. Meeting the real-time signal processing requirements for multi-channel EEG seizure detection for humans, considering that human EEG recording typical utilizes 20, 32, 64, or even 128 channels, would be beyond the abilities of the laptop. Second, the use of the PC in the closed-loop system increases the system cost and limits the portability of the system. Third, the seizure detector generated an alarm for when to turn the TFS on but the TFS was actually turned on by a person, which introduced some delay between the seizure detection and the application of the TFS.

To overcome the disadvantages listed above, we developed a new FPGA platform to run the real-time seizure control on. This system will automatically detect the seizure activities and apply the TFS, which forms a portable closed-loop automatic seizure control system, as shown in Figure 10. Moreover, the hardware can also be used for a multi-channel human tEEG recording and real-time signal processing and display system.

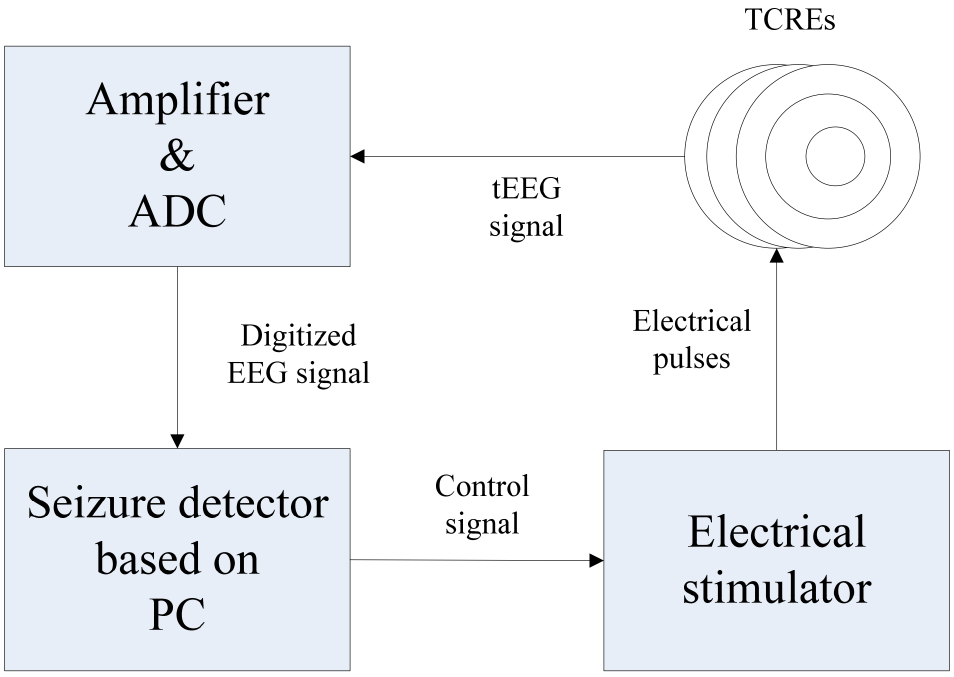


Figure 10 FPGA-based closed-loop seizure control system

4.2 Methods

The automatic seizure detector and EEG monitoring system contains several hardware and software subsystems. These subsystems include A-to-D converters (ADC), FPGA embedded controller and digital signal processor, USB interface, USB driver under Windows operating system, application software for signal display and data storing. The development of these subsystems is desribed in the following sections.

4.2.1 System Considerations

An EEG system records the potential on the scalp caused by the electrical activities from the brain. The EEG recordings for humans usually uses 20 to 128 channels and electrodes. A typical raw human EEG signal has an amplitude between 1μV to 10 mV, which indicates that the EEG signal is weak and has a high dynamic range. The tEEG signals are even weaker and have been reported to be hundreds of nanovolts [35]. The typical bandwidth of EEG recording is between 0.5 Hz to 70 Hz. Recently researches have shown that there are EEG activities in higher bands, with the frequency as high as 500 Hz [40, 41]. It seems that the wider band EEG signal recording and analysis will be the trend in the future. There are several other issues of EEG signals that must be accounted for, such as the high bias voltage caused by the half-cell effects. These are mainly dealt with in the analog domain by signal conditioning techniques, and not discussed here.

As we mentioned in the previous chapter, there are generally two approaches for seizure detection [16, 17], one is spike analysis based on time domain EEG signals, the other is spectral analysis based on the spectral domain of the EEG signal. We employed the second approach, spectral analysis, to develop our epilepsy seizure detector. This was since our earlier work showed that the spectral analysis was more robust in noisy environments such as movement artifacts and mains interference. These requirements forced us to perform the spectral analysis in real-time in the frequency domain and formed the basis for our new system design.

To digitize the tEEG signal with high dynamic range a high resolution ADC is desired. Having multi-channel capability of the ADC is also beneficial, since the capacity of the system is as many as 128 or even 256 channels. Thus we began work on implementing the analog front end, ADS1299, from Texas Instruments. The ADS1299 has 8 differential channels of programmable gain instrumentation amplifiers, and a 24-bit, 8-channel delta-sigma ADC, with daisy-chain port that supports cascading multiple chips as an ADC system with 128 channels or more. For automatic seizure detection, the up to 128 channels of data has to be transformed to the spectral domain in real-time. Two platforms, FPGA and graphics processing unit (GPU) are suitable for this task. However, the FPGA has more resources for communicating with other devices, such as communicating with ADC and USB interfaces. Thus, we used an FPGA as the ADS1299 controller, digital signal processing, and communication core.

We used an ALTERA DE-2 evaluation board with the Cyclone IV serial FPGA. The estimated data recording rate of the ADCs is about 13 MB per second, so the USB 2.0 interface, which provides a maximum data transferring rate over 30MB/s, fits the estimated data rate. On the PC side, a USB driver was developed for real-time data acquisition from the USB interface to save the data in a temporary buffer in the Windows kernel. The application software was developed to fetch the data from the data buffer in the Windows kernel and save the data to the hard disk and also display it on the screen. Figure 11 shows the high level structure of the system.

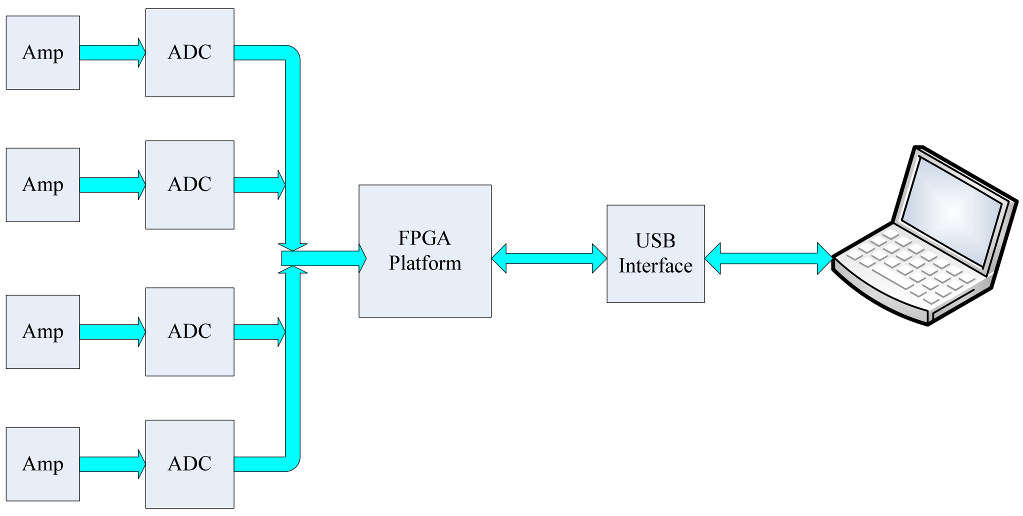


Figure 11 System structure

4.2.2 Hardware Connection

The ADS1299 is an 8-channel ADC and programmable gain preamplifier. To build a data acquisition system with more than 8 channels, multiple ADS1299s need to be connected with the daisy-chain configuration [42], and then connected to the FPGA through a serial peripheral interface (SPI) port, which is shown in Figure 12.

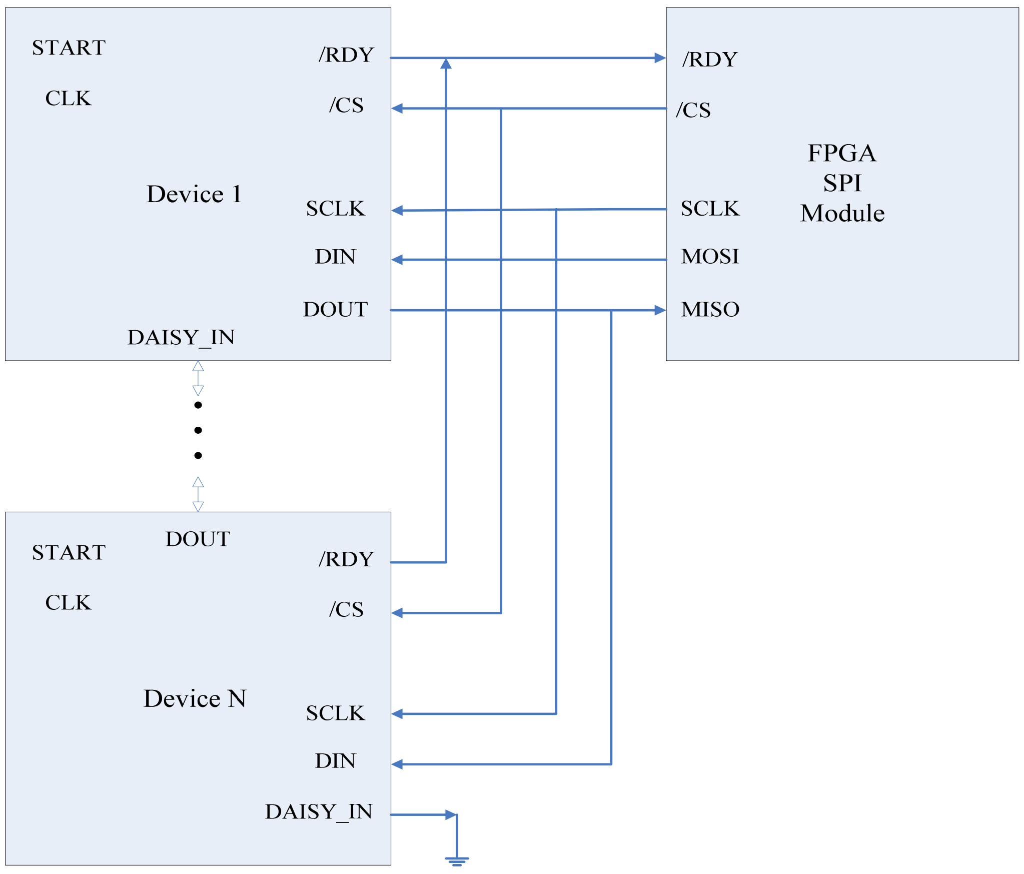


Figure 12 Daisy-chain connection of multiple ADCs

With this connection, the analog to digitized data is available from the DOUT pin to the DAISY_IN pin, and sent out to FPGA through the DOUT pin of the first ADC, which is Device 1 in Figure 12. The timing for the data reading cycle is shown in Figure 13 [42].

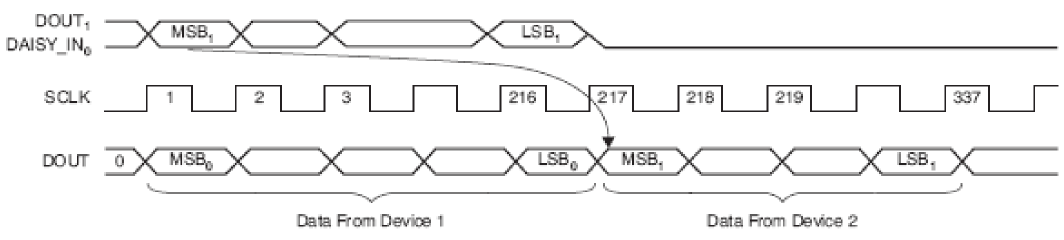


Figure 13 Daisy-chain ADCs data reading timing

4.2.3 FPGA Logic Structure

The FPGA plays two roles in our system. First, the FPGA acquires and transmits the data from the ADS1299 to the computer for storage. On one side, it communicates with the ADCs through the SPI port. The FPGA must configure and acquire data from the ADCs and the FPGA must also pack the data according to the USB frame format. On the other side, the FPGA monitors the IN and OUT FIFOs of the USB interface. When the OUT FIFO signifies that it is not empty, the FPGA reads the data from this FIFO; or if the IN FIFO is not full, and there is data stored in FPGA, then the FPGA will write a data frame to the IN FIFO. Second, the real-time seizure detector runs in the FPGA. Figure 14 shows the logic structure we have progrmmed into the FPGA. There are three clock rates (time domains) needed for our design that were programmed into the FPGA. The SPI interface module to the ADCs runs on a 4 MHz clock, it is responsible for communicating with the ADCs through the SPI port. The seizure detection part, which includes the FFT processor and the CUSUM detector, runs on an 80 MHz clock. The USB controller interface runs on a 50 MHz clock. The FIFOs are placed between the time domains for cross time domain data transmission.

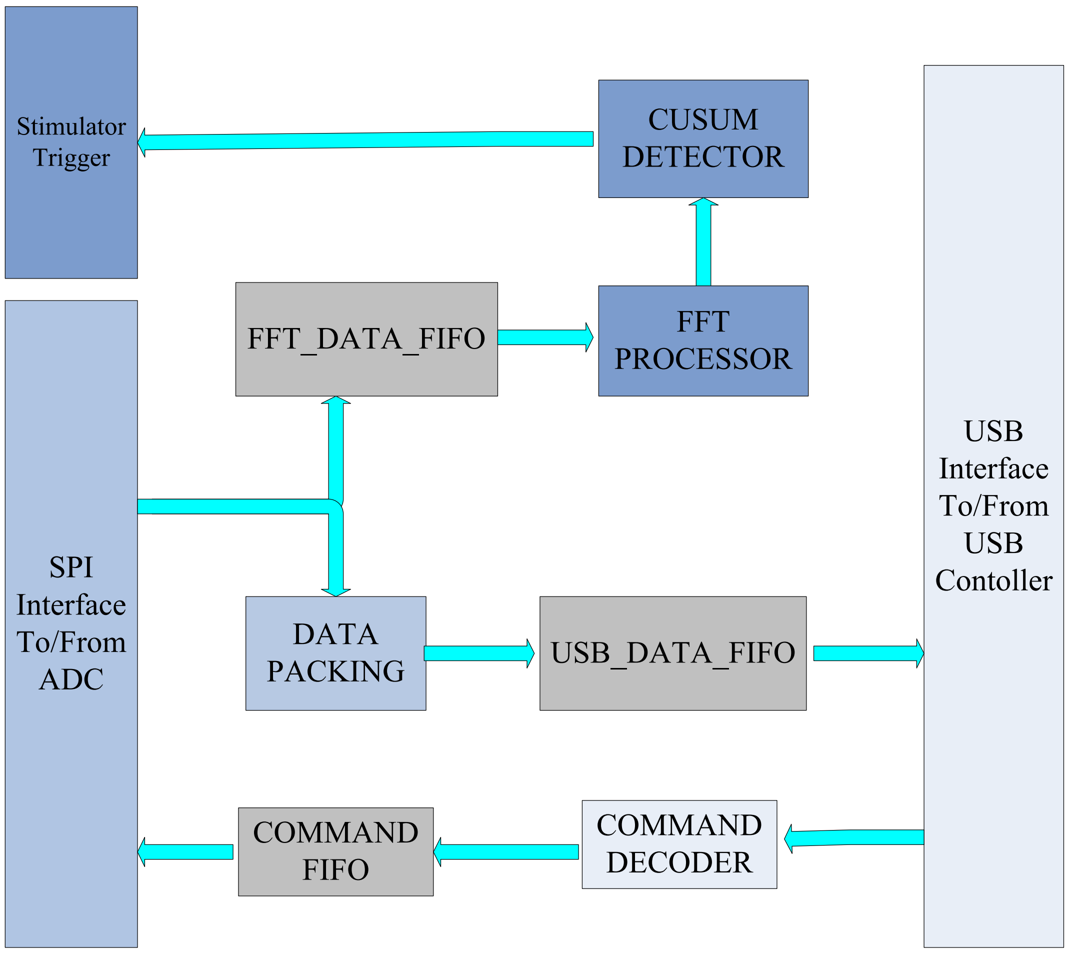


Figure 14 Logic structure in ALTERA Cyclone IV FPGA

4.2.4 Multi-Channel FFT Processor Design

The multi-channel FFT processor is designed to transfer the recorded tEEG signal to the spectral domain. According to the seizure detection algorithm we developed, each time one second of data is recorded it is transferred to the spectral domain by the FPGA FFT algorithm. A commonly used sampling rate of the EEG recording is 250 Hz, therefore the data length of the FFT processor was set to 256. Due to the limit of the resources in the FPGA, an FFT processor needs to be applied to multiple channels.

The data width of the input stage of the FFT processor is 24-bits, which is determined by the output data width of the ADCs.

As mentioned above the FFT processor runs at a much faster clock than the clock of the data input rate. There are two reasons that make the data processing speed of the FFT processor the bottleneck of the seizure detector which is why it is run at a much faster clock rate than the other two processes. First, even though the FFT significantly reduces the computational complexity of the discrete Fourier transform (DFT), it still requires a large amount of multiplications and additions. An N-point FFT with complex inputs needs $2N\log_2 N$ real multiplications and $2N\log_2 N$ real additions. In our case, N is 256, then for each 256 samples there are 4096 multiplications and 4096 additions, which mean that for each sample, there are 16 multiplications and 16 additions needed. Second, the hardware resources, such as embedded multipliers, are limited in the FPGA, so the same hardware resources in the FPGA have to be applied to data from multiple chaneels.

As a result, the pipeline multi-channel interleaving structure was employed to overcome these problems. There are mainly four pipeline FFT structures: radix-2 multi-path delay commutator (R2MDC), radix-2 single-path delay feedback (R2SDF), radix-4 multi-path delay commutator (R4MDC) and radix-4 single-path delay feedback (R4SDF), all are shown in Figure 15. Due to existence of the feedback loop, which complicates the pipelining of the butterfly multipliers [43], the single-path delay feedback structures are not suitable for high throughput rate FFT processors. The radix-4 butterfly multipliers significantly reduce the number of stages of the pipeline structure, which helps to minimize the delay from the input to the output.

However, as we may see in Figure 16, the radix-4 butterfly multiplier is much more complicated than the radix-2 butterfly multiplier. This may not be a problem for very large scale integration (VLSI), but the resources in the FPGA are pre-placed and limited, so the complicated structure introduces significant routing delay, which limits the highest clock frequency on which the logic can run. For these reasons, we employed the radix-2 multi-path delay commutator structure to build the multi-channel FFT processor. The interleaving method was employed to perform the FFT process on data from multiple channels. This method is implemented by making two changes on the single channel radix-2 multi-path delay commutator structure: (1) increase the number of the registers connected to the controllers by N times, and (2) change the controller blocks to reuse every parameter for N times. Figure 17 shows the structure of the interleaving radix-2 multi-path delay commutator for the N-channel FFT transform. The multipliers utilized in the butterflies are further pipelined to ensure the 80 MHz clock rate. Since we utilized the fixed-point data format in the computation, bit growth was considered to preserve the precision and avoid overflow.

However, there are two computations in the butterfly multiplier that may cause bit growth. First, the data width must grow 1 bit in the add/subtract stage of the butterfly multiplier. Second, the multiplication of the sinesoid value and the input data may cause an extra bit growth. The bit growth at , the multiplication of the sinesoid value only needs to be considered once in all the FFT stages [44]. To accommodate for the worst case bit growth, the width of the output of the first butterfly multiplier increases by 2 bits, the width of the output of all the following butterfly multipliers increase by 1 bit. The bit increasing consideration is shown in Figure 18.

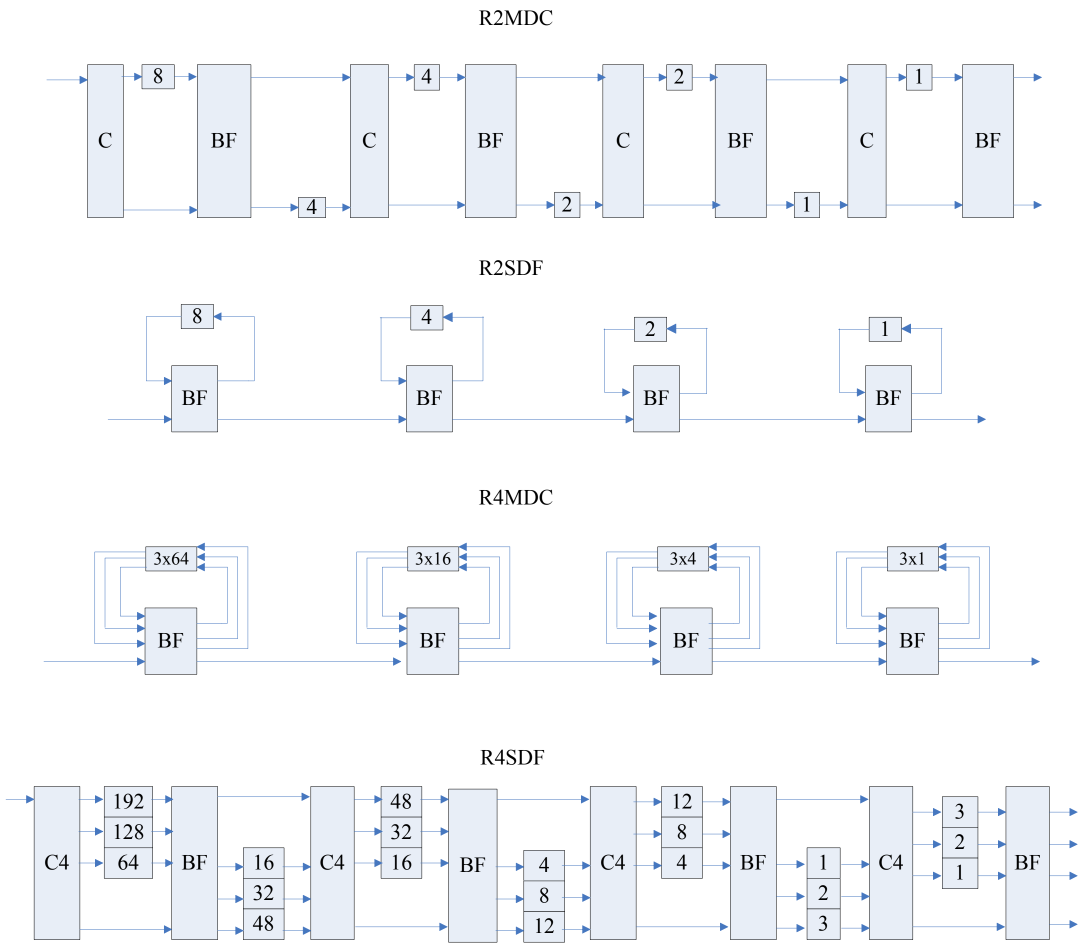


Figure 15 Four typical pipeline FFT structures: R2MDC, R2SDF, R4MDC, R4SDF

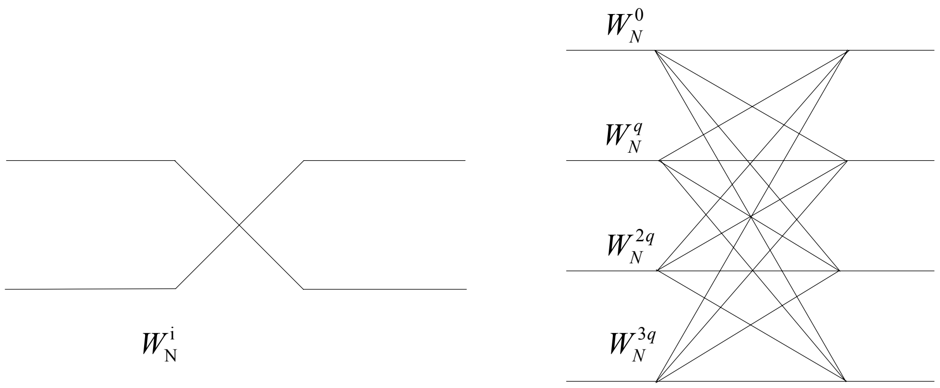


Figure 16 Radix-2 (left) and Radix-4 (right) DIF butterfly multipliers

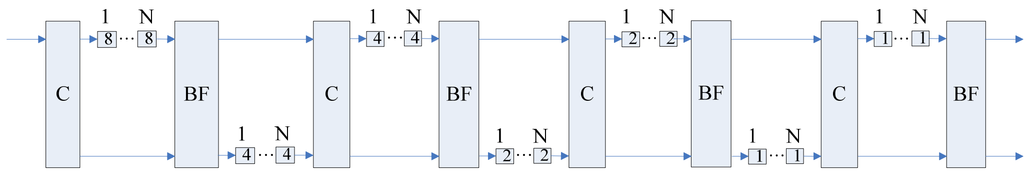


Figure 17 16-point R2MDC with interleaving for N channels

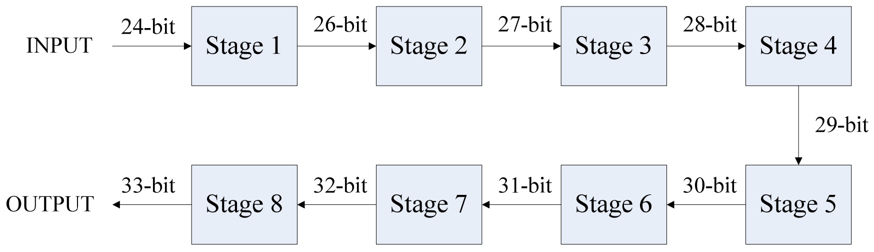


Figure 18 Eight stage Radix-2 FFT bit consideration

Each stage of the FFT is built with a controller, a butterfly multiplier and a parameter module, as shown in Figure 19. The controller works as a router that directs the data and parameters to the input of the butterfly multiplier at the right time by controlling two shift buffers and the address signal for the parameter module. A round-back counter was implemented in the router. By setting the maximum value of of the round-back counter to N the controller can be adjusted to work for an N-channel FFT processor. The butterfly multiplier computes the multiply-add value: $A + BC$, where A and B are full scale complex numbers, and C is a normalized unit complex number. There are two approaches to implement the butterfly multiplier. The first approach, referred to as the CORDIC method, takes advantage of the multiplier of a full scale complex number and a normalized unit complex number, which is actually a rotation of the full scale complex number. The advantage of this approach is that it only takes bit shift and add, so it saves the hardware multiplier resource, but the complicated structure makes it hard for pipelining. The second approach, however, employs the normal multiplication and addition to implement the butterfly multiplier. The parameters in the second approach are pre-computed and stored in the chip. This method, of course, cost more hardware multiplication resources, but the structure is straightforward. We employed the second method in our design. For each multiplier we utilized four multipliers and two adders. The multipliers were further pipelined to maximize the clock rate of the FFT processor.

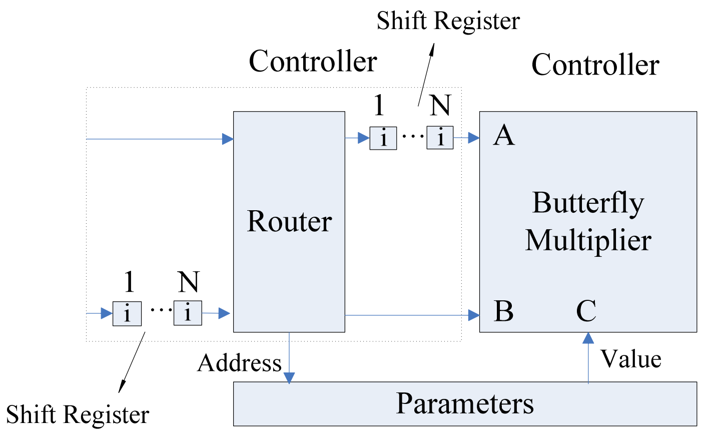


Figure 19 Detailed structure of an FFT Stage

4.2.5 CUSUM Detector Design

According to the seizure detection algorithm we developed, there are two main tasks. The initial task is started at the beginning for training the CUSUM detector on non-seizure data. In this task, the spectral data from the FFT processor is cumulatively averaged. At the end of this task, the average value is loaded for the CUSUM detector. The second task is the seizure detection task. During the seizure detection, the CUSUM detector continuously monitors the power spectrum of the interested bands, and in real-time triggers the stimulator if specific features are detected, and updates the parameters according to the current power spectrum. The two tasks perform different procedures, and are therefore implemented with two different modules. Figure 20 shows the top level modules of the CUSUM detector.

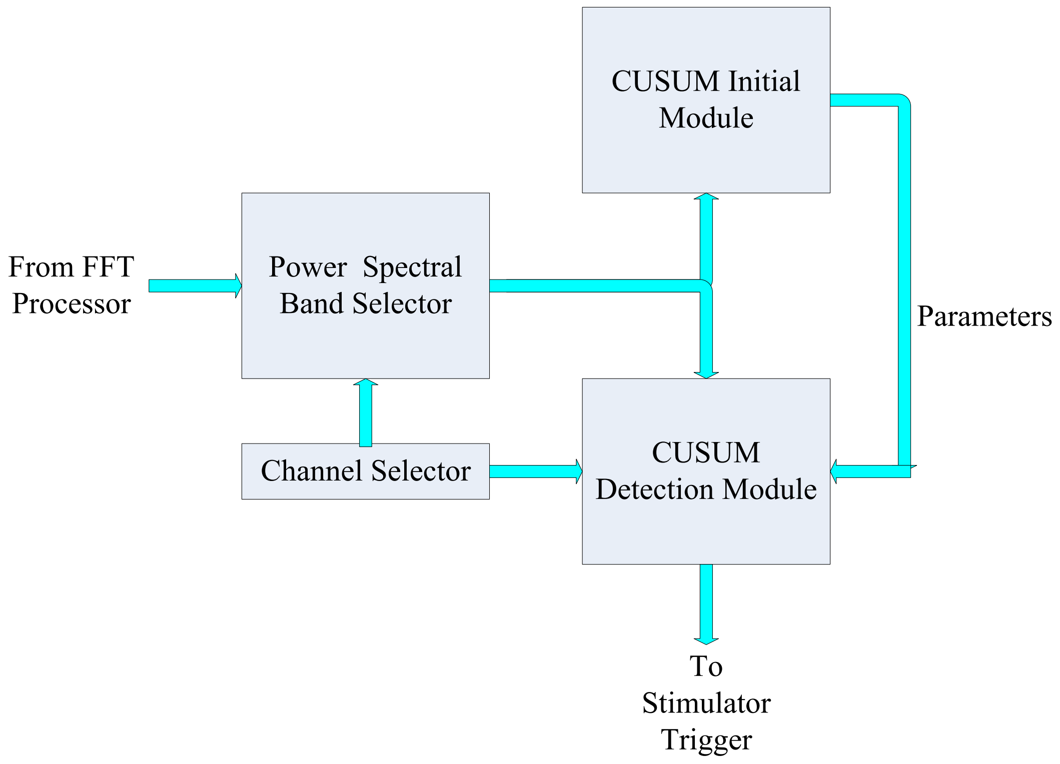


Figure 20 Structure of the CUSUM detector

The spectral data output from the FFT processor is pre-processed by the power spectral band selector, which basically computes the relative power of the specific bands that are determined in the training task. Since we employed the interleaving structure for the FFT processor, the power spectrum for different channels arrives at the CUSUM detector one by one. Also, we are only interested in certain band(s) of the spectrum. Therefore, a single CUSUM detector is sufficient for monitoring all the channels. In the EEG recordings, it is common to find some channels that are corrupted with artifacts; this is usually due to the bad placement of electrodes or loss of contact over time. These channels should be excluded from the detector, since the signal from these channels can mislead the detector. We implemented the “Channel Selector” module in the CUSUM detector that was used to disable the corrupted channels. This “Channel Selector” is implemented as a bit map, in which every channel has a corresponding bit. Value ‘1’ on a bit means the data from corresponding channel would be sent to the CUSUM detector, otherwise the data from the correponding channel would be discarded. The bit map can be configured through our application software.

4.2.6 Software Design

Software was developed to initialize and transfer a data stream from the FPGA to the hard disk in the Laptop with a transfer rate up to 30 MB per second. There are three parts to this software: embedded firmware for the Cypress CY7C68053A USB controller, USB device driver for the Windows kernel, and application software for Windows user space. The dark blocks in Figure 21 are the three parts we developed.

In our design, the Cypress CY7C68053A USB controller was configured as a slave first-in first-out (FIFO) register module. With this configuration, the USB controller works as a pair of passive FIFOs. One transfers data from the FPGA to the PC, the other transfers data from the PC to the FPGA, as shown in Figure 22. The USB protocol is executed by the hardware implemented in the USB controller chip. The embedded software was developed to configure the registers for the USB protocol execution during the initialization stage. Table 11 lists all the registers and the configuration values for correctly configuring the USB controller to slave FIFO mode.

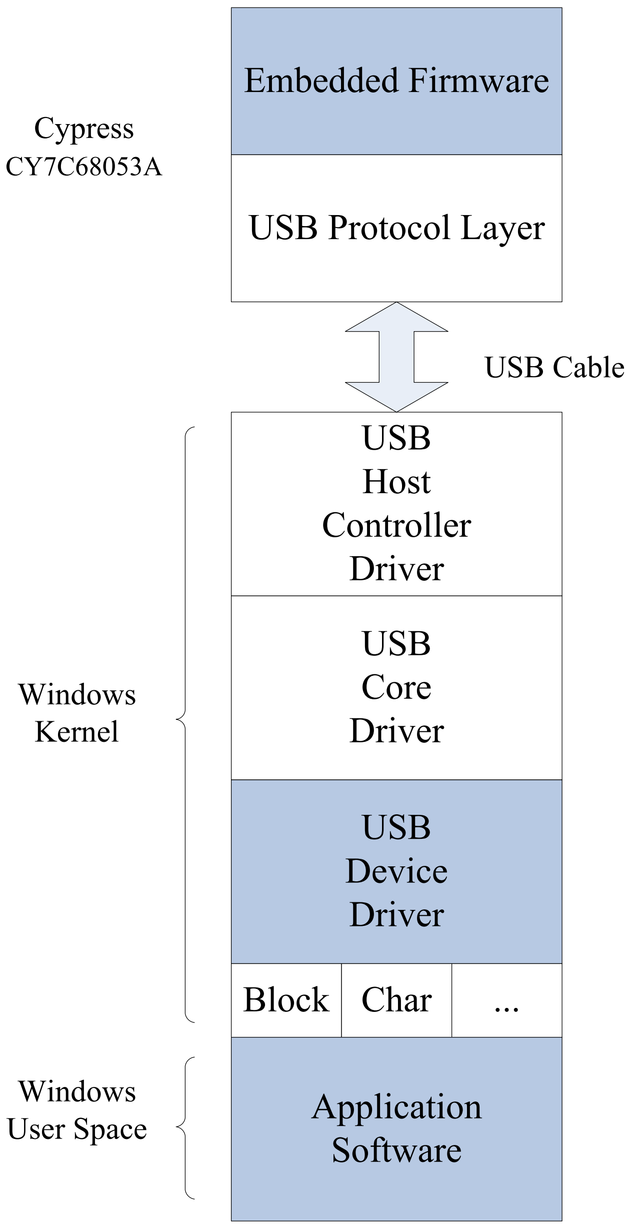


Figure 21 Firmware, driver and application software stack structure

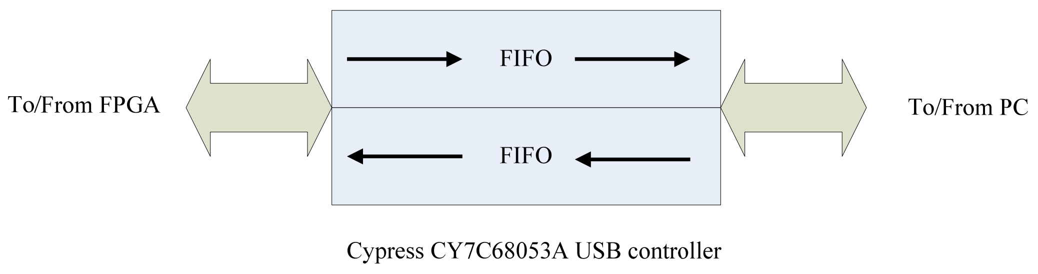


Figure 22 Slave FIFO mode of the Cypress CY7C68053A

Table 10 Register configurations of the Cypress CY7C68053A for slave FIFO code

OEA	0x03
IFCONFIG	0x03
REVCTL	0x03
EP1OUTCFG	0x20
EP1INCFG	0x20
EP4CFG	0x20
EP8CFG	0x20
EP2CFG	0xE2
EP6CFG	0xA0
EP4CFG	0x00
EP8CFG	0x00
PINFLAGSAB	0x00
PINFLAGSCD	0x00
FIFOPINPOLAR	0x00
EP2FIFOPFH	0x90
EP2FIFOPFL	0x00
EP2FIFOCFG	0x0C
EP2AUTOINLENH	0x02
EP2AUTOINLENL	0x00

On the PC side, the USB host controller driver and the USB core driver are provided with the Windows operating system. These two drivers handle almost all the USB protocol affairs, as well as communicate with the device hardware, which is the Cypress CY7C68053A USB controller in our case. The main part we developed at the driver level is the USB device driver. As shown in Fig. 5.11, the USB device driver works as a bridge between the USB core driver and the application software. On one hand, the USB device driver continuously sends data read requests to the USB core driver, once the request is responded to, it saves the data to a buffer. On the other hand, the USB device driver implements several methods for the application software to fetch data, the USB interface information or send commands down to the FPGA.

The USB device driver is developed based on the Windows Driver Foundation (WDF). The WDF is a new driver model for the Windows operating system that is based on Windows Driver Model (WDM). In other words, the WDF framework encapsulates the WDM framework, and exposes a much more user-friendly interface for driver developers. Specifically, the WDF implements a set of default power management callback functions which deals with all the plug and play and power issues very well.

Even with the support of WDF framework, there are still two challenges for developing the USB device driver. First, the USB device driver has to read the data from the FIFO of the Cypress CY7C68053A USB controller immediately when it is available. Second, the buffer in the USB device driver is accessible for both the saving and fetching routine that run asynchronously, so a mechanism is needed to prevent the buffer from being accessed by the two routines at the same time while still maintaining the high speed data transfer. The first challenge is addressed by a mechanism called continuous reader. The idea is that we always maintain two or three data read requests so that while one request is returned with some data for processing, there is still a pending request for new data. There are several synchronization approaches, such as “Critical Section” or “Spine Lock”, which can guarantee only one routine accessing the buffer at any single time by blocking the other routine that is trying to access the buffer. But simply applying these approaches will decrease the performance of the driver in terms of data transfer rate. For instance: if a data read routine for the application software obtained the access to the buffer, and started to copy data, the read completion routine has to wait for the copy procedure to finish before it can access the buffer and save the data. Therefore, we employed the circular buffer structure that stores the data in the kernel. This structure allows us to avoid the synchronization problem. Two pointers are utilized in the circular buffer, one pointer points to the start block of circular buffer, the other pointer points to the end block of the circular buffer. The read completion routine is responsible for updating the end pointer, while the read routing for the application software updates the start pointer. The pointers round back to the minimum address of the buffer once they reach the maximum address. If the start pointer catches up to the end pointer, the buffer is empty; otherwise if the end pointer catches up to the start pointer, the buffer is full. If the end pointer further overlaps the start pointer, data in the buffer is corrupted. The overlap detector triggers an alarm to the application software about this serious problem. However, it's the application software's responsibility to prevent the circular buffer from becoming full. The depth of the buffer is calculated according to the current sampling rate andthe number of recording channels. Three endpoints were implemented in the driver: default control endpoint, input endpoint and output endpoint, as shown in Figure 23. The control endpoint is the utilized to USB interface information retrieval. The input endpoint is employed to transfer data from the hardware to the Windows USB driver. The output endpoint is used to send data from the driver to the hardware. The structure of the driver is shown in Figure 24.

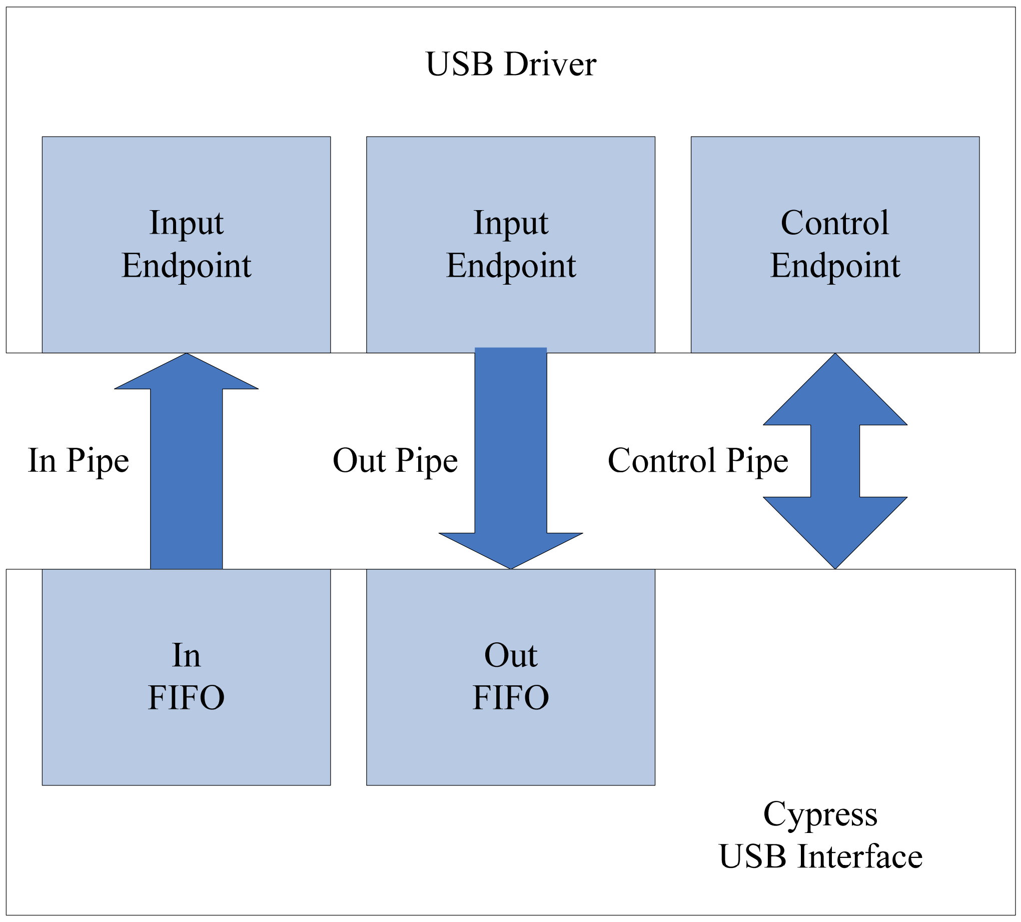


Figure 23 Endpoints of the driver and thevirtual connection to the dardware

As we mentioned above, our application software is responsible for retrieving the data from the USB device driver without any loss. A precise timer is desired to finish this task. Thus, we employed the timer-queue timer, which is mostly applied in the multimedia field. According to our testing on the Windows XP operation system, the timer-queue timer can give a quite constant interval of 1.1 ms with the jitter less than 0.01 ms. In our application, the timer-queue timer triggers a timer call back function every 1.1 ms to retrieve data from the buffer of the USB device driver. According to the two tasks of the applications software, the retrieved data is sent to two buffers in the user space. First, the data is sent to a ping-pong buffer. Every 30 seconds, an extra thread is created to write the data in one memory block of the ping-pong buffer to the hard disk, and in the next 30 seconds the coming data will be sent to the other memory block. Second, the data is also sent to a buffer in the signal display module. In our signal display module, data is processed and mapped to the pixels on the screen. The procedure flow chart of the application software is shown in Figure 25.

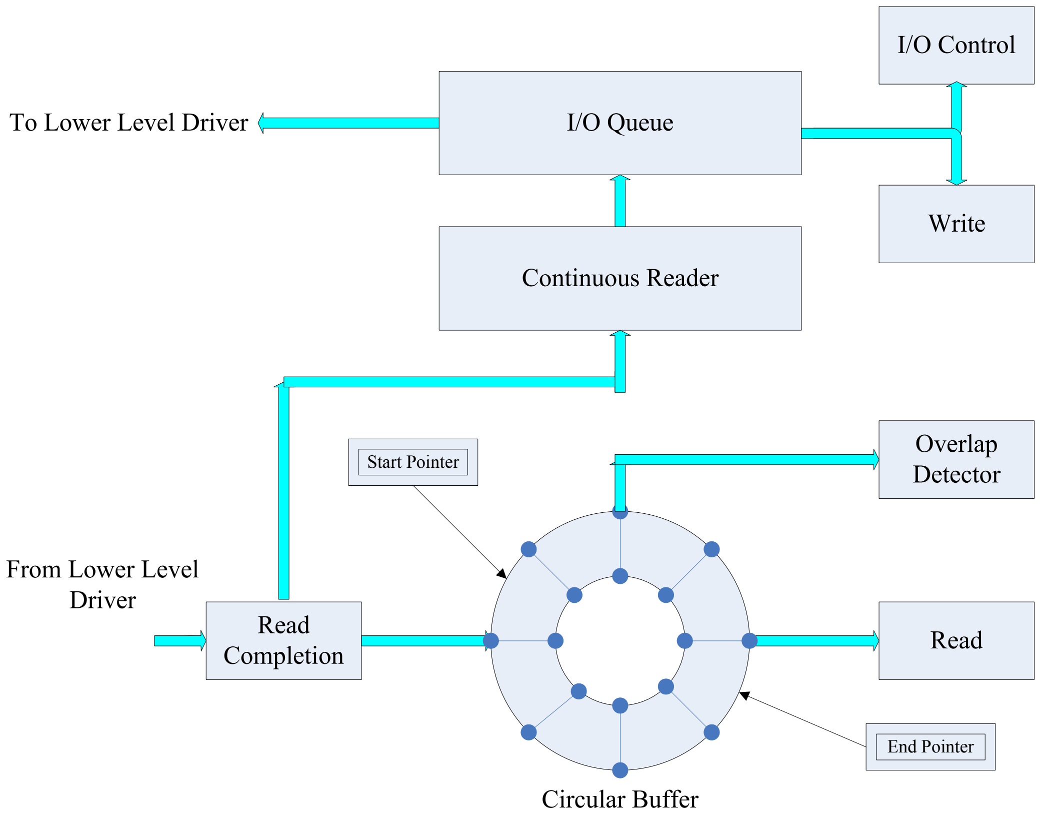


Figure 24 Structure of the USB device driver

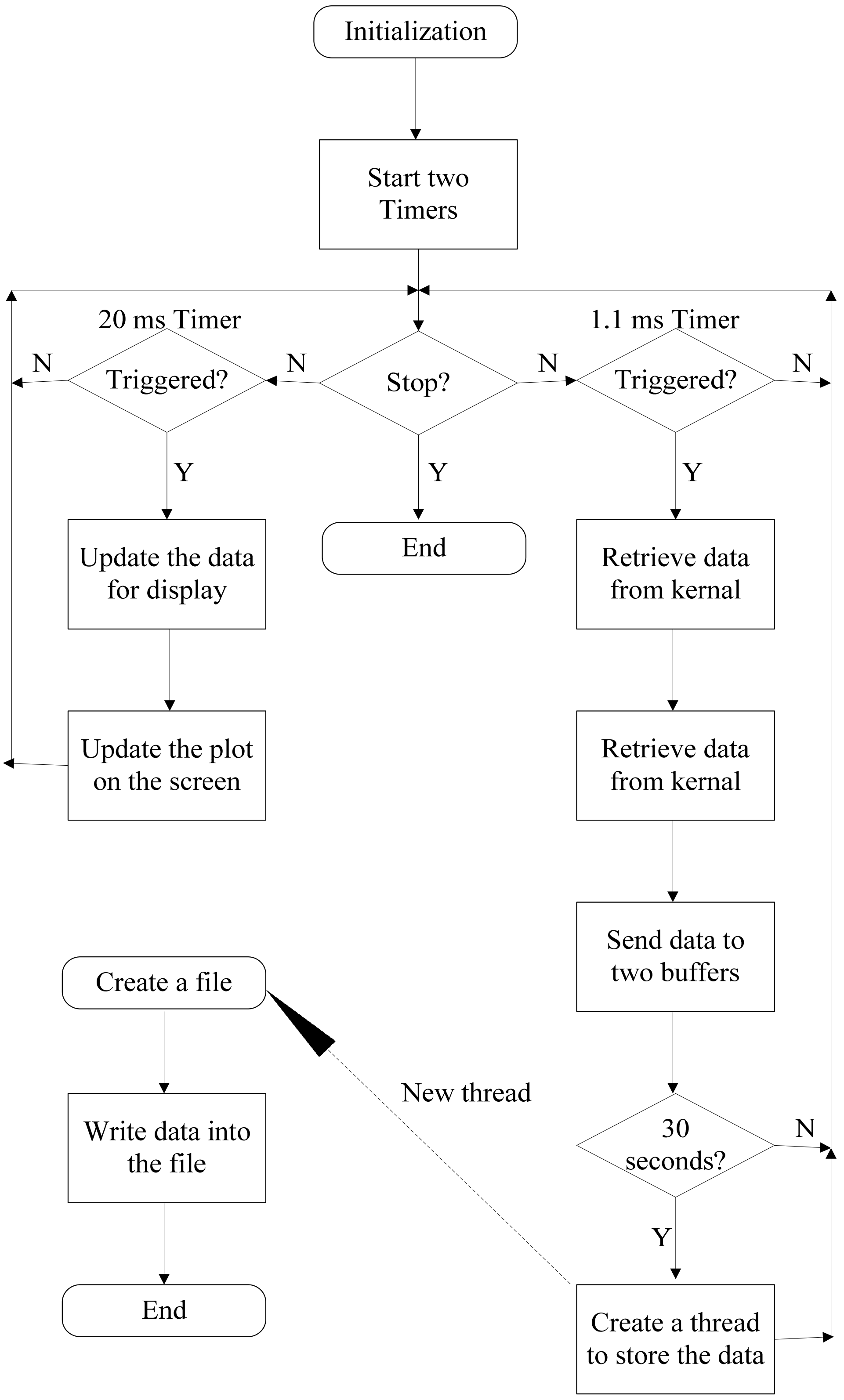
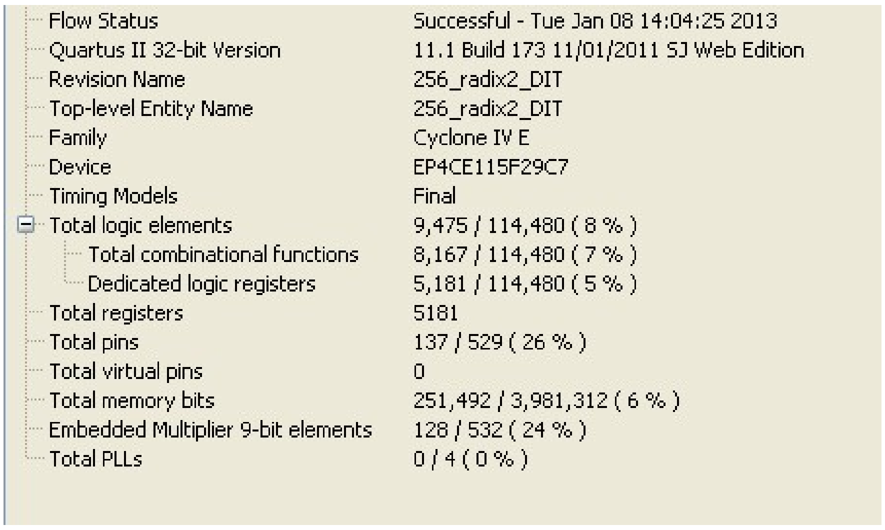


Figure 25 Procedure flow chart of the application software

4.3 Results

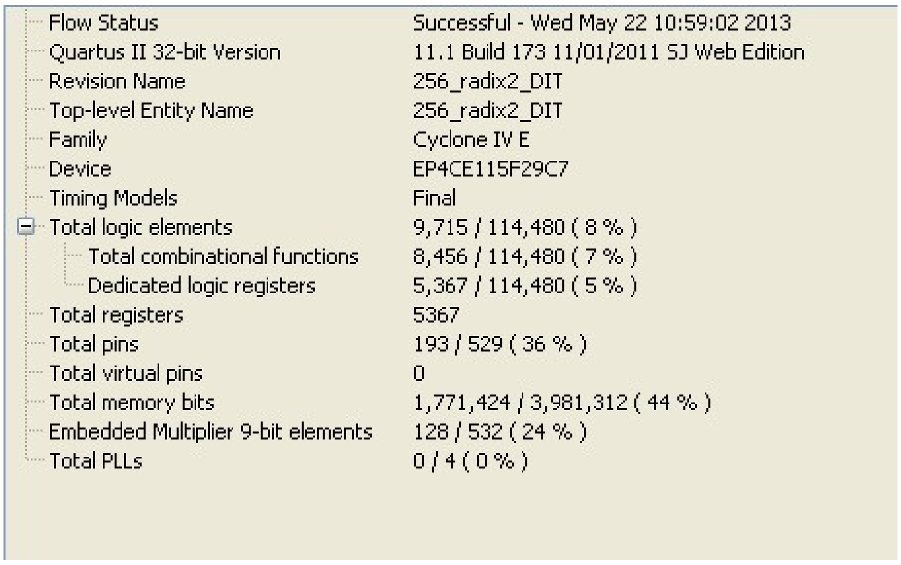
An 8-channel 256-point FFT processor and a 64-channel 256-point FFT processor were synthesized for the target Cyclone IV E FPGA with Altera Quatus II software。 The synthesis results are shown in Figure 26 and Figure 27. To give a comparison, even though it's not a fair comparison, the synthesis results of the standard single channel 256-point fixed bit-width (24-bit) intellectual property (IP) core library from Altera Corporation is shown in Figure 28.



Flow Status	Successful - Tue Jan 08 14:04:25 2013
Quartus II 32-bit Version	11.1 Build 173 11/01/2011 SJ Web Edition
Revision Name	256_radix2_DIT
Top-level Entity Name	256_radix2_DIT
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	9,475 / 114,480 (8 %)
Total combinational functions	8,167 / 114,480 (7 %)
Dedicated logic registers	5,181 / 114,480 (5 %)
Total registers	5181
Total pins	137 / 529 (26 %)
Total virtual pins	0
Total memory bits	251,492 / 3,981,312 (6 %)
Embedded Multiplier 9-bit elements	128 / 532 (24 %)
Total PLLs	0 / 4 (0 %)

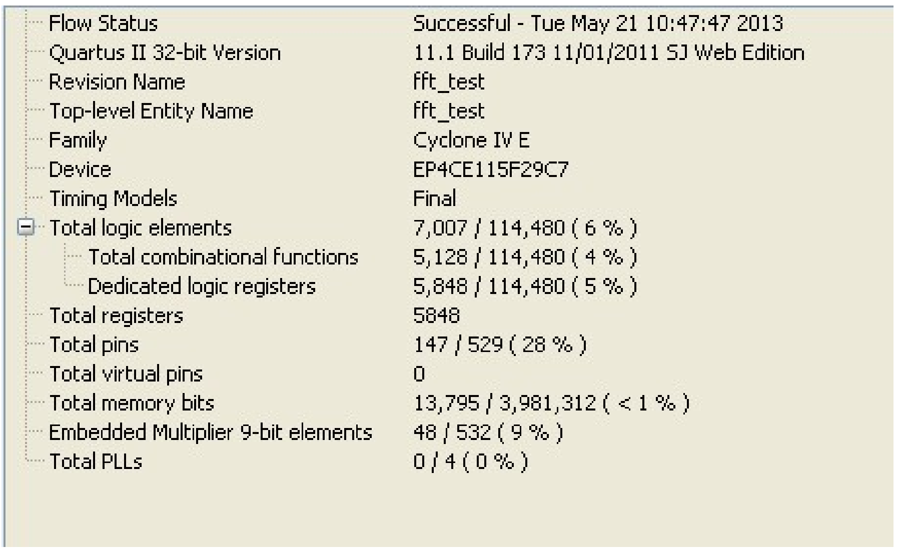
Figure 26 synthesis summary of an 8-channel FFT processor

The main synthesis results are summarized in Table 11. Our 8-channel and 64-channel FFTs consume approximately 35% more of the logic elements compared to the standard single channel FFT core. Considering we process 7 or 63 more channels of data, and only consume 35% more logic elements, this is quite an efficient implementation. The increase in resources is due to the hardware resources being re-used for every channel. However, we can clearly see that the memory bits consumed is increasing with the increasing number of channels. These memory bits are consumed



Flow Status	Successful - Wed May 22 10:59:02 2013
Quartus II 32-bit Version	11.1 Build 173 11/01/2011 SJ Web Edition
Revision Name	256_radix2_DIT
Top-level Entity Name	256_radix2_DIT
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	9,715 / 114,480 (8 %)
Total combinational functions	8,456 / 114,480 (7 %)
Dedicated logic registers	5,367 / 114,480 (5 %)
Total registers	5367
Total pins	193 / 529 (36 %)
Total virtual pins	0
Total memory bits	1,771,424 / 3,981,312 (44 %)
Embedded Multiplier 9-bit elements	128 / 532 (24 %)
Total PLLs	0 / 4 (0 %)

Figure 27 synthesis summary of a 64-channel FFT processor



Flow Status	Successful - Tue May 21 10:47:47 2013
Quartus II 32-bit Version	11.1 Build 173 11/01/2011 SJ Web Edition
Revision Name	fft_test
Top-level Entity Name	fft_test
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	7,007 / 114,480 (6 %)
Total combinational functions	5,128 / 114,480 (4 %)
Dedicated logic registers	5,848 / 114,480 (5 %)
Total registers	5848
Total pins	147 / 529 (28 %)
Total virtual pins	0
Total memory bits	13,795 / 3,981,312 (< 1 %)
Embedded Multiplier 9-bit elements	48 / 532 (9 %)
Total PLLs	0 / 4 (0 %)

Figure 28 synthesis summary of a standard single channel FFT IP core

by the shift registers in our FFT processors.

The 8-channel and 64-channel FFT processors utilize 128 9-bit embedded multipliers, while the standard single channel FFT core utilizes only 48 9-bit embedded multipliers. There are two reasons that make this difference. First, the standard single channel FFT core keeps a 24-bit data width for all the stages, while our FFT processors employed a data width increasing algorithm to achieve a better transform precision, which is shown in Figure 18. Second, the standard single channel FFT core re-used the multipliers in each butterfly. Re-using the multipliers would actually slow down our implementation of the FFT.

For the three FFT processors, we used the same Synopsys design constraints (SDC) file, which requires a clock rate of 200 MHz. The synthesis results show that the maximum clock rate of the standard single channel FFT IP core, the 8-channel FFT processor and the 64-channel FFT processor is 206 MHz, 156 MHz and 101 MHz, respectively. As we can see, the maximum clock rate decreases with the increasing of the number of channels. The reason for this is that more embedded memory blocks are involved with more channels, which eventually increases the length of the wire connections. The wire delay from the wire connection is the main constraint on the maximum clock rate. The 101 MHz maximum clock rate guarantees that the 64-channel FFT processor can run safely at a lower clock rate, such as 50 MHz as we desire. To improve the resource consumption performance, we may lower the maximum clock restraint to 80 MHz, which will give the synthesizer more space to optimize the resource consumption.

Table 11 Summary of the synthesis results for the three FFT processors

	Logic Elements Consumption	Total Memory Bits Consumption	Embedded Multipliers Consumption	Maximum Clock Rate
8-channel FFT Processor	9415	251,492	128	156MHz
64-channel FFT Processor	9715	1,771,424	128	101MHz
Single Channel FFT IP Core	7007	13,795	48	206MHz

We tested the FFTs by using a combination of several sinusoid signals. The sinusoids with different frequencies were generated using Matlab and stored in a data file that was sent to the multi-channel FFT processor. In Figure 23, the top panel shows the floating –point FFT transform results from Maltab with the command "fft", while the bottom panel shows the FFT transform results from our fixed-point FFT processor with the same input. The relative square mean error between them is 6.2632e-005.

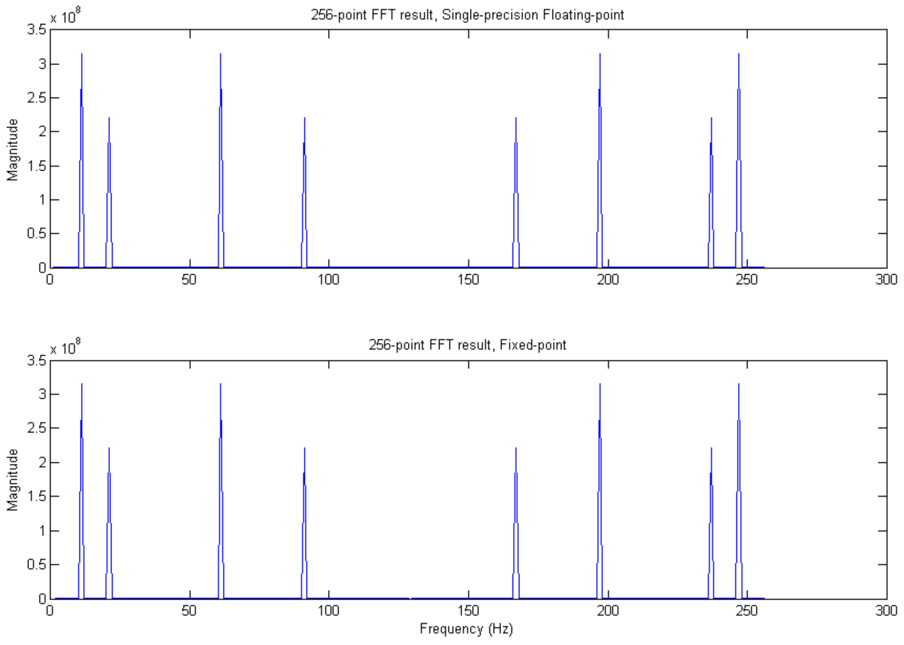


Figure 29 FFT transfer results of Matlab (top) fft command and our FFT processor

The FPGA controller and digital signal processing module, the USB interface, Windows USB driver and application software were first tested and verified separately. The EEG monitoring system was also tested with a 128-channel configuration with simulated data in the FPGA. An 8-channel configuration was also tested with a real ECG recording. Further animal experiments are needed to test the seizure detector in a real environment.

4.4 Discussion

A distinct automatic seizure detector was developed to solve the special problem we met in the animal experiments. We emphazied the multi-channel, up to 128 channels, digital signal processing ability of the detector, a series of digital signal processing functional units were developed for this purpose. Thus, the system has the potential to be applied to real-time human seizure detection with up to 128 EEG channels, while most of the existing seizure detection systems only work for far fewer channels [44].

Accompanied with the automatic seizure detection system, we also developed a multi-channel EEG monitoring system. There are many commercial EEG monitoring systems on the market, such as Grass Comet and Aura EEG monitering systems, g.USB serial EEG monitoring systems. But these systems only allow the user to access the EEG data in the software level, while our system allows us to access and process the data at the hardware level, which significantly improves the real-time signal processing ability. With our system, we can implement and test many more applications which might not be pratical if we use the commercial systems.

4.5 Conclusion

An EEG automatic seizure detection system for rats, or any being, was implemented in an FPGA based embedded system. An EEG monitoring system was also developed. Both systems have the advantage of real-time signal processing ability for large numbers of channels.

CHAPTER 5: FUTURE WORK

5.1 Resistor Inductor Capacitor (RLC) Model of TCRE Recording

With the special tripolar structure, the TCRE has different characteristics in terms of the interaction with paste and the scalp. To help fully understand this type of electrode, a passive RLC network model may be built to mimic the recording behavior of the TCRE, as shown in Figure 30.

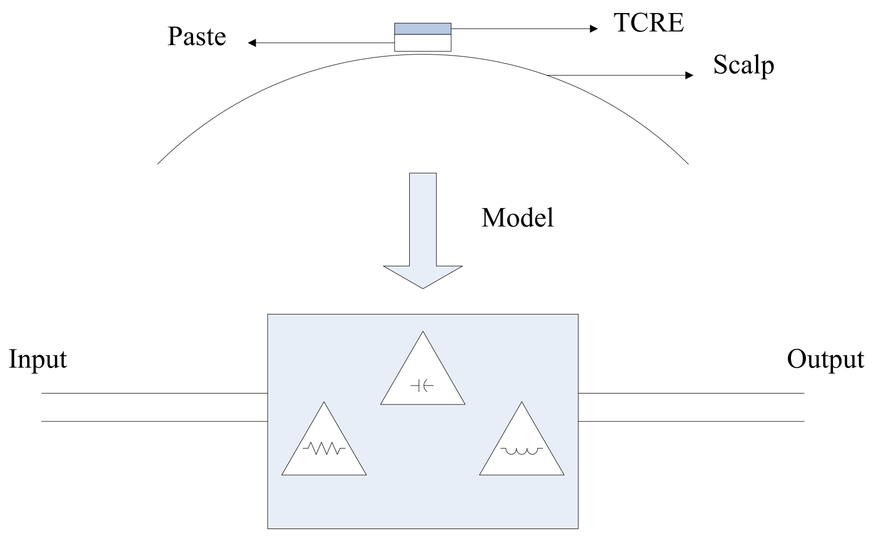


Figure 30 RLC model of EEG recording with TCRE

A possible approach to build the RLC structure is to place a current source at the location of the cortex area of a head model, vary the frequency of the source then measure the simulated output from the TCRE. The RLC model may be built according to the source frequency and magnitude curve.

5.2 Optimized TCRE Preamplifier Development

The precise passive circuit model for the TCRE should give the ability to design a preamplifier with optimization of the noise and cross talk rejection. The input-referred voltage noise of a low noise instrumentation amplifier is typically $50\,\mathrm{nV}/\sqrt{\mathrm{Hz}}$ (with the gain 100, 0-100 Hz band), and the input-referred current noise is typically at $1\,\mathrm{pA}/\sqrt{\mathrm{Hz}}$, the noise model of the instrumentation amplifier is shown in Figure 31 [45]. The input-referred voltage noise is directly added to the input signal, while the input-referred current noise is coupled into the circuit by producing the corresponding voltage on the signal conditioning circuit and the sensor circuit, which is shown in Figure 32.

With the circuit models shown below in Figure 32, the noise of the preamplifier can be quantized easily by hand calculation or PSPICE simulation. Proper usage of this noise quantization method should greatly help to design a low noise preamplifier for the TCRE.

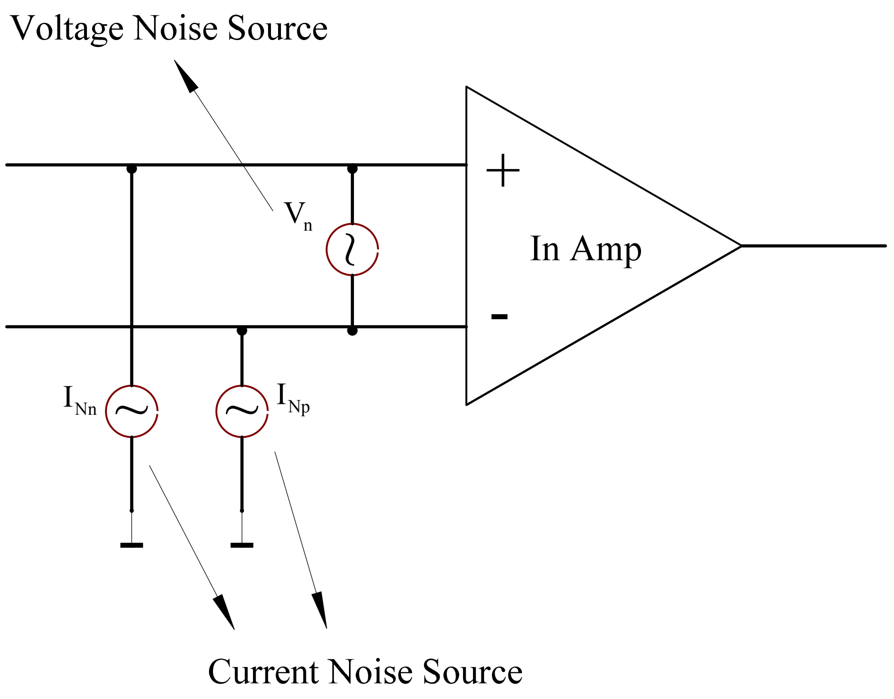


Figure 31 Noise model of the instrumentation amplifier

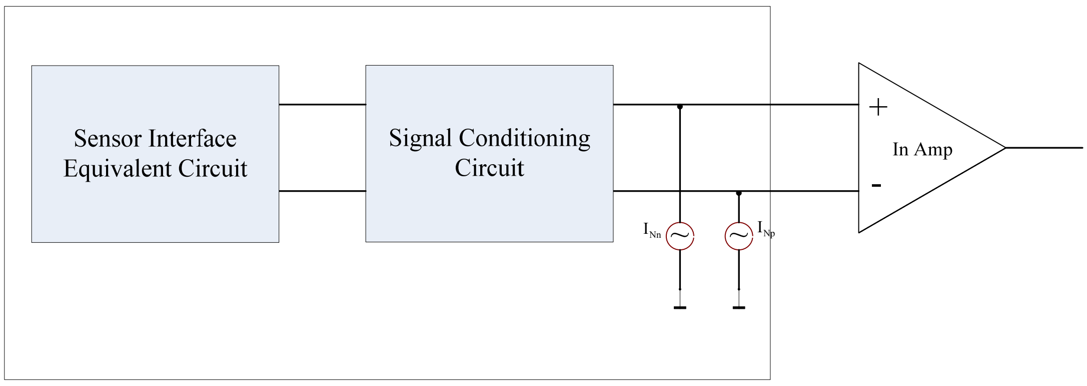
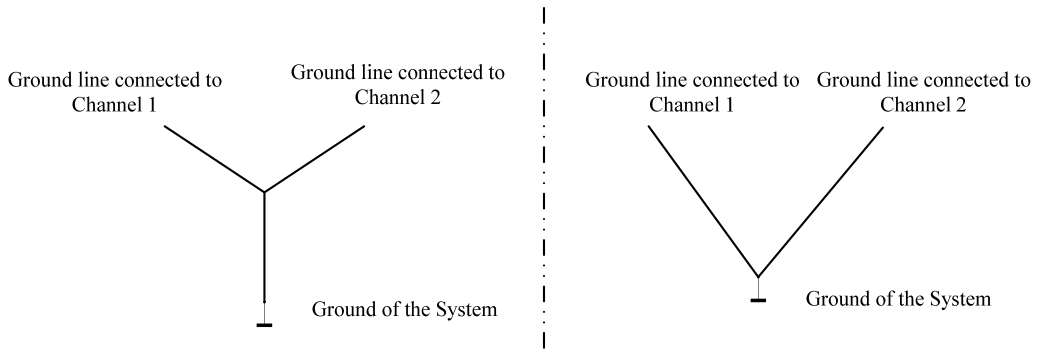

Sensor Interface Equivalent Circuit
Signal Conditioning Circuit
I_{Nn}
I_{Np}
+
In Amp
-

Figure 32 Current noise source coupled to the preamplifier circuit

Another important issue in the development of the preamplifier for the TCRE, or generally for EEG electrodes, is that the crosstalk among channels must be rejected. Crosstalk is the signal from one channel that creates an undesired signal on the other channels. For EEG applications, this phenomenon will directly decrease the spatial resolution of the recording. To lessen or avoid the crosstalk, some attention should be paid when laying out the printed circuit board (PCB) for the preamplifier. The common ground connection for the power supply lines, which is shown in Figure 33, must be prohibited. A star-connection, also shown in Figure 33, is a better alternative, while separate ground and power planes are mostly needed.



Ground line connected to Channel 1
Ground line connected to Channel 2
Ground of the System
Ground line connected to Channel 1
Ground line connected to Channel 2
Ground of the System

Figure 33 Common ground connection (left) and star connection (right)

5.3 Need for a More Portable Automatic Seizure Detection and Alarm System

Our automatic seizure detection system can be applied to human seizure detection with a proper real time human seizure detection algorithm. A compact system with fewer channels may be valuable for remote seizure detection of patients with epilepsy. Some patients have seizures only once per month or even less, which means they can live a normal life most of the time, but need an alarm once the occasional seizure happens. In this case, only a few channels may be needed if the epilepileptic area of the brain can be pre-determined by neurologists. Based on these conditions, a compact 8 channel, or less, automatic seizure detector and alarm system may be developed as shown in Figure 34. It is similar to the seizure detection system we developed, except for two differences. First, the human seizure detection algorithm should be implemented in the FPGA; second, a WIFI (or other wireless) module should be implemented for remote communication. The whole system can be placed on a single PCB with the area of 5 cm by 5 cm. The system would be continuously monitoring the EEG of the patients, if no seizure is detected, the WIFI module is off, if there is some seizure activity detected, an alarm will be sent to the patient, and through the WIFI module to the hospital, and the recorded EEG signal will also be transmitted out though the WIFI module. On the other side, a network driver and application software should be developed based on a PC to communicate with the embedded system to receive the alarm and the transmitted EEG signal. These are some of the possible directions that need to be investigated for future applications of the TCREs.

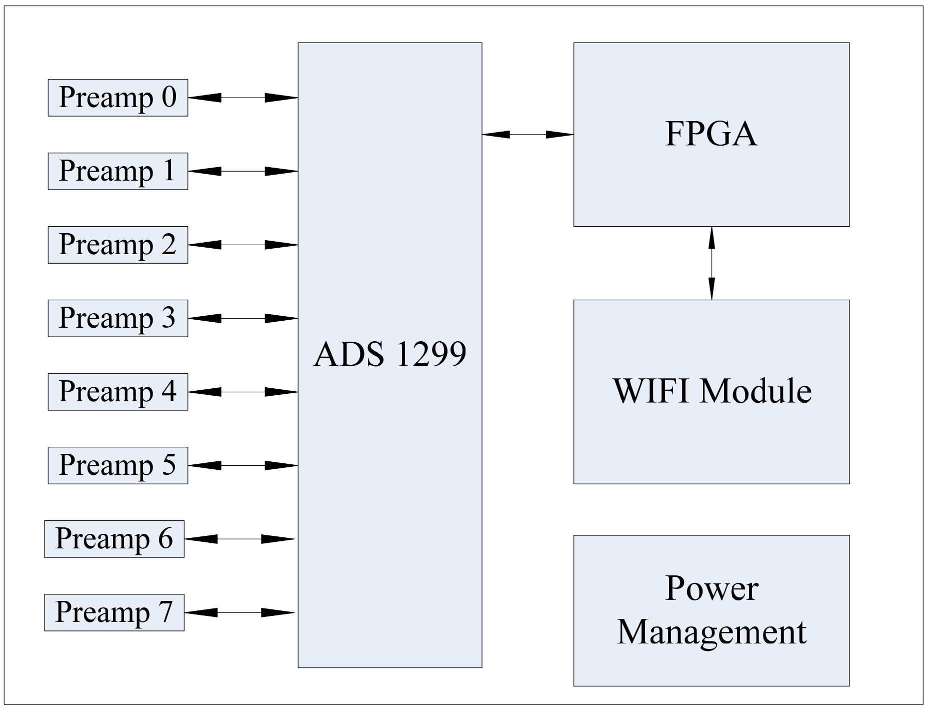


Figure 34 Compact 8-channel seizure detection and alarm system

REFERENCES

[1] Nunez P. L., Silberstein R. B., Cadiush P. J., Wijesinghe J., Westdorp A. F., and Srinivasan R., “A theoretical and experimental study of high resolution EEG based on surface Laplacians and cortical imaging,” Electroencephalography and Clinical Neurophysiology vol. 90, pp. 40-57, 1994.

[2] He B., “Brain electrical source imaging: scalp laplacian mapping and cortical imaging,” *Crit. Rev. Biomed. Eng.*, vol. 27, pp. 149–188, 1999.

[3] Perrin F., Pernier J., Bertrand O. and Echallier J. F., “Spherical splines for scalp potential and current density mapping,” Electroencephalography and Clinical Neurophysiology 72: 184-1987, 1989.

[4] Biggins C., Fein G., Raz J. and Amir A., “Artifactually high coherences results from using spherical spline computation of scalp current density,” Electroenceph. and Clin. Neurophysiol., 79: 413-419, 1991.

[5] Fein G., Raz J., and Turetsky B., “Brain electrical activity: the promise of new technologies,” In: S. Zakhari and E. Witt (Eds.)., Imaging in Alcohol Research. Proc. of a Workshop on Imaging in Alcohol Research, Wild Dunes, SC, 9–11: 49-78, 1991.

[6] Le J., Menon V., and Gevins A., “Local estimate of surface Laplacian derivation on a realistically shaped scalp surface and its performance on noisy data,” Electroencephalography and Clinical Neurophysiology., 92: 433-441, 1994.

[7] Wang K., and Befleiter H., “Local polynomial estimate of surface Laplacian,” Brain Topograpgy, Vol 12, Issue 1, 19-29, 1999.

[8] Hjorth B., “An on-line transformation of EEG scalp potentials into orthogonal source derivations,” Electroencephalography and Clinical Neurophysiology, Vol.. 39,526-530, 1975.

[9] Besio W., Koka K., Aakula R., Dai W., “Tri-polar Concentric Ring Electrode Development for Laplacian Electroencephalography,” IEEE Trans BME, Vol. 53, No. 5, pp. 926-933, 2006.

[10] World Health Organization (WHO). http://www.who.int/mediacentre/factsheets/fs999/en/. Accessed on May 14, 2013.

[11] Groves D. A., Brown V. J., “Vagal nerve stimulation: A review of its applications and potential mechanisms that mediate its clinical effects,” Neurosci Biobehav. Rev., 29(3): 493-500, 2005.

[12] Kringelbach M. L., Jenkinson N., Owen S. L. and Aziz T. Z., "Translational principles of deep brain stimulation," Nat. Rev. Neurosci., 8(8): 623-635, 2007.

[13] Gigante P. R., Gooman R. R., "Responsive neurostimulation for the treatment of epilepsy," Neurosurg. Clin. N Am., 22(4): 477-480, 2011.

[14] Utz K. S., Dimova V., Oppenländer K. and Kerkhoff G., "Eectrified minds: transcranial direct current stimulation (tDCS) and galvanic vestibular stimulation (GVS) as methods of non-invasive brain stimulation in neuropsychology--a review of current data and future implications," Neuropsychologia., 48(10): 2789-2810, 2010.

[15] Barker A. T., Jalinous R. and Freeston I. L., "Non-Invasive Magnetic Stimulation of Human Motor Cortex," Lancet, 1(8437): 1106-1107, 1985.

[16] Bédard M., Agid Y. , Chouinard S. , Fahn S., Korczyn A. and Lesperace P., "Mental and Behavioral Dysfunction in Movement Disorders," Humana Press, 2003.

[17] Stevanovic D., "Epilepsy - Histological, Electroencephalographic and Psychological Aspects," Intech Press, 2012.

[18] Gloor P., "Contributions of electroencephalography and electrocorticography in the neurosurgical treatment of the epilepsies," Adv Neurol, 8:59–105, 1975.

[19] Wilson S., Emerson R., "Spike detection: a review and comparison of algorithms," Clin Neurophysiol. 2002 Dec, 113(12):1873-81.

[20] Polat K., Gunes S., "Classification of epileptiform EEG using a hybrid system based on decision tree classifier and fast Fourier transform," Applied Mathematics and Computation*, 187*(2), 1017–1026, 2007.

[21] Tzallas A. T., Tsipouras M. G., Fotiadis D. I., "Epileptic seizure detection in EEGs using time-frequency analysis," IEEE Trans Inf Technol Biomed, *13*(5), 703-710, 2009.

[22] Ocak H., "Automatic detection of epileptic seizures in EEG using discrete wavelet transform and approximate entropy," Expert Systems with Applications*,* 36(2), 2027–2036, 2009.

[23] Quyen M. L. V., Martinerie J., Baulac M., Varela F., "Anticipating epileptic seizures in real time by a non-linear analysis of similarity between EEG recordings," Neuroreport, Jul 13, 10(10):2149-55, 1999.

[24] Basseville M. and Nikiforov I. V., "Detection of abrupt changes: Theory and Application," Prentice Hall Press, 1993.

[25] Besio W., Makeyev O., Medvedev A., Gale K., "Effects of transcranial focal stimulation via tripolar concentric ring electrodes on pentylenetetrazole-induced seizures in rats", Epilepsy Research, Jan, 2013.

[26] Makeyev O., Liu X., Luna-Munguía H., Rogel-Salazar G., Mucio-Ramirez S., Liu Y., Sun Y. L., Kay S. M. and Besio W. G., "Toward a noninvasive automatic seizure control system in rats with transcranial focal stimulations via tripolar concentric ring electrodes," IEEE Trans. Neural Syst. Rehabil Eng., 20(4): 422-431, 2012.

[27] Collura T., "History and evaluation of electroencephalographic instruments and techniques," J. Clin Neurophysiol., 10(4):476-504, 1993.

[28] Cuffin B. N., and Cohen D., "Comparison of the magnetoencephalogram and electroencephalogram," Electroencephalography and Clinical Neurophysiology, 47, 132-146, 1979.

[29] Tandonnet C., Burle B., Hasbroucq T., Vidal F., "Spatial enhancement of EEG traces by surface Laplacian estimation: comparison between local and global methods," Clin. Neurophysiol. , 116(1):18-24, 2005.

[30] Hansen P., "The truncated SVD as a method of regularization," BIT Numerical Mathematics, Vol. 27, Issue 4, 534-553, 1987.

[31] He B., Yao D., Lian J., and Wu D., "An Equivalent Current Source Model and Laplacian Weighted Minimum Norm Current Estimates of Brain Electrical Activity," IEEE Trans. BME, Vol. 49, No. 4, 2002.

[32] Oostenveld R., Praamstra P., "The five percent electrode system for high-resolution EEG and ERP measurements," Clinical Neurophysiol, 112(4): 713-719, 2001.

[33] Montgomery D.C., "Design and analysis of experiments, Wiley," Hoboken, 2004.

[34] He B., Wang Y., and Wu D., "Estimating cortical potentials from scalp EEGs in a realistically shaped inhomogeneous head model by means of the boundary element method," IEEE Trans. BME, Vol. 46, No. 10, 1999:713-719, 2001.

[35] Koka K., Besio W., "Improvement of spatial selectivity and decrease of mutual information of tripolar concentric ring electrodes," J. Neuroscience Methods, 165(2): 216-222, 2007.

[36] Bragin A., Wilson C. L., Fields T., Fried I. and Engel Jr J., "Analysis of seizure onset on the basis of wideband EEG recordings, " Epilepsia 46 59–63, 2005.

[37] Gotman J., “Automatic detection of seizures and spikes, ” Journal of Clinical Neurophysiology, 16:130–140, 1999.

[38] Chander R., Urrestarrazu E. and Gotman J., “Automatic detection of high frequency oscillations in human intracerebral EEGs,” Epilepsia 47 (4) 37, 2006.

[39] Gotman J., “Automatic recognition of epileptic seizures in the EEG,” Electroencephalography and Clinical Neurophysiology, 54:530-540, 1982.

[40] Urrestarazu E., Chander R., Dubeau F., Gotman J., “Interictal high-frequency oscillations (100-500 Hz) in the intracerebral EEG of epileptic patients,” Brain., 130:2354-66, 2007.

[41] Worrell G., “High-frequency oscillations recorded on scalp EEG,” Epilepsy Curr., 12(2): 57-58, 2012.

[42] Texas Instruments, “Low-Noise, 8-Channel, 24-Bit Analog Front-End for Biopotential Measurements,” 2012.

[43] Parhi K., “VLSI Digital Signal Processing Systems: Design and Implementation,” Wiley-Interscience Press, 1999.

[44] Chandler D., Bisasky J., Stanislaus J. and Mohsenin T., “Real-time multi-channel seizure detection and analysis hardware, ” Biomedical Circuits and Systems Conference 2011, 41-44, 2011.

[45] Kester W., “Practical design techniques for sensor signal conditioning,” Prentice Hall Press, 1999.

APPENDIXES

APPENDEX A: Matlab Code for VEP Data Processing

```
clear all
close all
clc


[data(:,1),data(:,2),data(:,3),data(:,4),data(:,5),data(:,6),dat
a(:,7),data(:,8),...

data(:,9),data(:,10),data(:,11),data(:,12),data(:,13),data(:,14)
,data(:,15),data(:,16),...

data(:,17),data(:,18),data(:,19),data(:,20),data(:,21),data(:,22
),data(:,23),data(:,24),...

data(:,25),data(:,26),data(:,27),data(:,28),data(:,29),data(:,30
),data(:,31),data(:,32), data(:,33)...
 ] =textread('LED2Hz15Hzstimulation_LinDu.txt','%f %f %f %f %f
%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f %f %f %f');


N = size(data(:,30),1);

% for i = 1:30
%     for j = 1:N
%         if abs(data(j,i)) >= 100
%             data(j,i) = 0;
%         end
%     end
% end
%
% for i = 32:33
%     for j = 1:N
%         if abs(data(j,i)) >= 100
%             data(j,i) = 0;
%         end
%     end
% end


i=3e4;
k=0;
while(i<N)
 if data(i,31) > -700
     i=i+1;
```

```
elseif data(i,31) > data(i+1,31)
     i=i+1;
 else
     k=k+1;
     segment(k,1)=i;
     i=i+10;
 end
end



vep_data = zeros(61,33);
for i = 1:100
    for j = 0:60
        k = segment(i,1) + j;

        for ch=1:33
            vep_data(j+1,ch) = vep_data(j+1,ch) + data(k,ch);
        end
    end
end

for j = 0:60
   for ch=1:33
    vep_data(j+1,ch) = vep_data(j+1,ch)/61;
   end
end

%%% normalize the outer ring VEP and tripolar VEP separately
vep_data_tri = zeros(61,15);
vep_data_outer = zeros(61,15);

vep_data_tri(:,1) = vep_data(:, 1);  % Cp5
vep_data_tri(:,2) = vep_data(:, 3);  % P3
vep_data_tri(:,3) = vep_data(:, 17); % Pz
vep_data_tri(:,4) = vep_data(:, 4); % P4
vep_data_tri(:,5) = vep_data(:, 9); % CP6
vep_data_tri(:,6) = vep_data(:, 10); % P5
vep_data_tri(:,7) = vep_data(:, 11); % P6
vep_data_tri(:,8) = vep_data(:, 19); % P7
vep_data_tri(:,9) = vep_data(:, 12); % Po7
vep_data_tri(:,10) = vep_data(:, 16); % Po3
vep_data_tri(:,11) = vep_data(:, 32); % Poz
vep_data_tri(:,12) = vep_data(:, 23); % Po4
vep_data_tri(:,13) = vep_data(:, 25); % P8
vep_data_tri(:,14) = vep_data(:, 27); % O1
vep_data_tri(:,15) = vep_data(:, 29); % O2

vep_data_outer(:,1) = vep_data(:, 2);  % Cp5
vep_data_outer(:,2) = vep_data(:, 5);  % P3
vep_data_outer(:,3) = vep_data(:, 6); % Pz
vep_data_outer(:,4) = vep_data(:, 7); % P4
vep_data_outer(:,5) = vep_data(:, 18); % CP6
vep_data_outer(:,6) = vep_data(:, 8); % P5
vep_data_outer(:,7) = vep_data(:, 13); % P6
vep_data_outer(:,8) = vep_data(:, 14); % P7
vep_data_outer(:,9) = vep_data(:, 15); % Po7
vep_data_outer(:,10) = vep_data(:, 20); % Po3
vep_data_outer(:,11) = vep_data(:, 33); % Poz
vep_data_outer(:,12) = vep_data(:, 24); % Po4
vep_data_outer(:,13) = vep_data(:, 26); % P8
vep_data_outer(:,14) = vep_data(:, 28); % O1
vep_data_outer(:,15) = vep_data(:, 30); % O2

max_vep_data_tri = max(max(abs(vep_data_tri)));
max_vep_data_outer = max(max(abs(vep_data_outer)));


vep_data_tri = vep_data_tri/max_vep_data_tri;
vep_data_outer = vep_data_outer/max_vep_data_outer;

figure(2)

plot(vep_data_tri, 'r');
hold on
plot(vep_data_outer, 'k');
axis([1 61 -2 2])
hold off
figure(1)
C=0:1:60;
C=C*200/61;
plot(C, abs(fft(vep_data(:,3))))
```

APPENDEX B: USB Interface Debugger Source Code

```
// USB_DialogDlg.cpp : implementation file
//


#include "stdafx.h"
#include "USB_Dialog.h"
#include "USB_DialogDlg.h"
#include <setupapi.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <winioctl.h>
#include <Windows.h>
#include <Mmsystem.h>
#include "usb100.h"
#include "usbscan.h"
#include <mmsystem.h>
#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        enum { IDD = IDD_ABOUTBOX };

        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

// Implementation
protected:
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
```

```
END_MESSAGE_MAP()


// CUSB_DialogDlg dialog


CUSB_DialogDlg::CUSB_DialogDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CUSB_DialogDlg::IDD, pParent)
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);

	rcv_data_cnt = 0;
	frame_mark_pre_flag = true;
	frame_mark_pre_flag = false;

}

void CUSB_DialogDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CUSB_DialogDlg, CDialog)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_CONFIGINFO, &CUSB_DialogDlg::OnBnClickedConfiginfo)
	ON_BN_CLICKED(IDC_INFERFACEINFO, &CUSB_DialogDlg::OnBnClickedInferfaceinfo)
	ON_BN_CLICKED(IDC_ENDPOINTSINFO, &CUSB_DialogDlg::OnBnClickedEndpointsinfo)
	ON_BN_CLICKED(IDC_SWITCH, &CUSB_DialogDlg::OnBnClickedSwitch)
	ON_BN_CLICKED(IDC_BulkTest, &CUSB_DialogDlg::OnBnClickedBulktest)
	ON_EN_CHANGE(IDC_BULKIN, &CUSB_DialogDlg::OnEnChangeBulkin)
	ON_BN_CLICKED(IDC_LEDBARREAD, &CUSB_DialogDlg::OnBnClickedLedbarread)
	ON_BN_CLICKED(IDC_LEDBARSET, &CUSB_DialogDlg::OnBnClickedLedbarset)
	ON_BN_CLICKED(IDC_7LEDREAD, &CUSB_DialogDlg::OnBnClicked7ledread)
	ON_BN_CLICKED(IDC_7LEDSET, &CUSB_DialogDlg::OnBnClicked7ledset)
	ON_BN_CLICKED(IDC_SPEEDTEST, &CUSB_DialogDlg::OnBnClickedSpeedtest)
	ON_BN_CLICKED(IDC_STOPSPEEDTEST, &CUSB_DialogDlg::OnBnClickedStopspeedtest)
	ON_BN_CLICKED(IDC_BULKREAD, &CUSB_DialogDlg::OnBnClickedBulkread)
	ON_BN_CLICKED(IDC_BULKWRITE, &CUSB_DialogDlg::OnBnClickedBulkwrite)
	ON_BN_CLICKED(IDC_BufStatus, &CUSB_DialogDlg::OnBnClickedBufstatus)
	ON_BN_CLICKED(IDC_StTimer, &CUSB_DialogDlg::OnBnClickedSttimer)
	ON_BN_CLICKED(IDC_StopTimer, &CUSB_DialogDlg::OnBnClickedStoptimer)
	ON_BN_CLICKED(IDC_start1msres, &CUSB_DialogDlg::OnBnClickedstart1msres)
	ON_BN_CLICKED(IDC_end1msres, &CUSB_DialogDlg::OnBnClickedend1msres)
	ON_BN_CLICKED(IDC_BUFFERREAD, &CUSB_DialogDlg::OnBnClickedBufferread)
	ON_BN_CLICKED(Datatest_Start, &CUSB_DialogDlg::OnBnClickedStart)
	ON_BN_CLICKED(Datatest_End, &CUSB_DialogDlg::OnBnClickedEnd)
	ON_WM_TIMER()
END_MESSAGE_MAP()


// CUSB_DialogDlg message handlers
```

```
        ON_BN_CLICKED(Datatest_End, &CUSB_DialogDlg::OnBnClickedEnd)
        ON_WM_TIMER()
END_MESSAGE_MAP()


// CUSB_DialogDlg message handlers

BOOL CUSB_DialogDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        // Add "About..." menu item to system menu.

        // IDM_ABOUTBOX must be in the system command range.
        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);

        CMenu* pSysMenu = GetSystemMenu(FALSE);
        if (pSysMenu != NULL)
        {
                CString strAboutMenu;
                strAboutMenu.LoadString(IDS_ABOUTBOX);
                if (!strAboutMenu.IsEmpty())
                {
                        pSysMenu->AppendMenu(MF_SEPARATOR);
                        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
                }
        }

        // Set the icon for this dialog.  The framework does this automatically
        //  when the application's main window is not a dialog
        SetIcon(m_hIcon, TRUE);                 // Set big icon
        SetIcon(m_hIcon, FALSE);                // Set small icon

        // TODO: Add extra initialization here
        CTime ctime=CTime::GetCurrentTime();
        CString str;
        str.Format(_T("USB Driver Testing Started at: %d-%2d-%2d %02d:%02d
\r\n"),ctime.GetYear(),ctime.GetMonth(),ctime.GetDay(),ctime.GetHour(),ctime.GetMinut
e());
        //str.Format(_T("USB Driver Testing Started at: %b \r\n"),ctime.GetMonth());
        GetDlgItem(IDC_Display)->SetWindowText(str);
    str.ReleaseBuffer();

        return TRUE;  // return TRUE  unless you set the focus to a control
}

void CUSB_DialogDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
                CAboutDlg dlgAbout;
                dlgAbout.DoModal();
        }
        else
        {
                CDialog::OnSysCommand(nID, lParam);
        }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CUSB_DialogDlg::OnPaint()
{
        if (IsIconic())
        {
                CPaintDC dc(this); // device context for painting

                SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

                // Center icon in client rectangle
                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;

                // Draw the icon
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
        }
}



PSP_DEVICE_INTERFACE_DETAIL_DATA  CUSB_DialogDlg::GetDevicePath(LPGUID InterfaceGuid)
{

    HDEVINFO HardwareDeviceInfo;
    SP_DEVICE_INTERFACE_DATA DeviceInterfaceData;
    PSP_DEVICE_INTERFACE_DETAIL_DATA DeviceInterfaceDetailData = NULL;
    ULONG Length, RequiredLength = 0;
    BOOL bResult;

    HardwareDeviceInfo = SetupDiGetClassDevs(
                             InterfaceGuid,
                             NULL,
                             NULL,
                             (DIGCF_PRESENT | DIGCF_DEVICEINTERFACE));

    if (HardwareDeviceInfo == INVALID_HANDLE_VALUE) {
        // send this notification unless you override the CDialog::OnInitDialog()
        // function and call CRichEditCtrl().SetEventMask()
        // with the ENM_CHANGE flag ORed into the mask.

        // TODO:  Add your control notification handler code here
}

CScrollBar* CUSB_DialogDlg::GetScrollBarCtrl(int nBar) const
{
        // TODO: Add your specialized code here and/or call the base class

        return CDialog::GetScrollBarCtrl(nBar);
}


PSP_DEVICE_INTERFACE_DETAIL_DATA  CUSB_DialogDlg::GetDevicePath(LPGUID InterfaceGuid)
{

    HDEVINFO HardwareDeviceInfo;
    SP_DEVICE_INTERFACE_DATA DeviceInterfaceData;
    PSP_DEVICE_INTERFACE_DETAIL_DATA DeviceInterfaceDetailData = NULL;
    ULONG Length, RequiredLength = 0;
    BOOL bResult;

    HardwareDeviceInfo = SetupDiGetClassDevs(
                             InterfaceGuid,
                             NULL,
                             NULL,
                             (DIGCF_PRESENT | DIGCF_DEVICEINTERFACE));

    if (HardwareDeviceInfo == INVALID_HANDLE_VALUE) {
        printf("SetupDiGetClassDevs failed!\n");
        exit(1);
    }

    DeviceInterfaceData.cbSize = sizeof(SP_DEVICE_INTERFACE_DATA);

    bResult = SetupDiEnumDeviceInterfaces(HardwareDeviceInfo,
                                          0,
                                          InterfaceGuid,
                                          0,
                                          &DeviceInterfaceData);

    if (bResult == FALSE) {
        printf("SetupDiEnumDeviceInterfaces failed.\n");

        SetupDiDestroyDeviceInfoList(HardwareDeviceInfo);
        exit(1);
    }


    SetupDiGetDeviceInterfaceDetail(
        HardwareDeviceInfo,
        &DeviceInterfaceData,
        NULL,
        0,
        &RequiredLength,
        NULL
        );

    DeviceInterfaceDetailData = (PSP_DEVICE_INTERFACE_DETAIL_DATA)
LocalAlloc(LMEM_FIXED, RequiredLength);

    if (DeviceInterfaceDetailData == NULL) {
        SetupDiDestroyDeviceInfoList(HardwareDeviceInfo);
        printf("Failed to allocate memory.\n");
        exit(1);
    }

    DeviceInterfaceDetailData->cbSize = sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);

    Length = RequiredLength;

    bResult = SetupDiGetDeviceInterfaceDetail(
                  HardwareDeviceInfo,
                  &DeviceInterfaceData,
                  DeviceInterfaceDetailData,
                  Length,
                  &RequiredLength,
                  NULL);

    if (bResult == FALSE) {
        printf("Error in SetupDiGetDeviceInterfaceDetail\n");

        SetupDiDestroyDeviceInfoList(HardwareDeviceInfo);
        LocalFree(DeviceInterfaceDetailData);
        exit(1);
    }
  // AfxMessageBox(DeviceInterfaceDetailData->DevicePath);
    return DeviceInterfaceDetailData;
}


void CUSB_DialogDlg::OnBnClickedDirection()
{
        // TODO: Add your control notification handler code here

        CString str, str_pre;

        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);
        GetDlgItem(IDC_Display)->GetWindowText(str_pre);
        str.Format(_T("%s \r\n"),DeviceInfo->DevicePath);
        str=str_pre+_T("\r\nDriver Direction: \r\n")+str;
    GetDlgItem(IDC_Display)->SetWindowText(str);

        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();
```

```
}

void CUSB_DialogDlg::OnBnClickedDeviceinfo()
{
        // TODO: Add your control notification handler code here
        CString str, str_pre;

        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);
    HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
    }

        USB_DEVICE_DESCRIPTOR  UsbDeviceDescriptor;
        DWORD junk;

        if (!DeviceIoControl(hDevice,
                                                 USBSample_IOCTL_801,
                                                 NULL,
                                                 0,
                                                 (LPVOID) &UsbDeviceDescriptor,
                                                 18,
                                                 &junk,
                                                 (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
                exit(1);
        }

         GetDlgItem(IDC_Display)->GetWindowText(str_pre);
         str.Format(_T("\r\nUSB DEVICE DESCRIPTOR\r\nbLength: %d \r\nbDescriptorType:
%d \r\nbcdUSB: %d \r\nbDeviceClass: %d \r\nbDeviceSubClass: %d\r\nbDeviceProtocol: %d
\r\nbMaxPacketSize0: %d \r\nidVendor: %d \r\nidProduct: %d \r\nbcdDevice: %d
\r\niManufacturer: %d \r\niProduct: %d \r\niSerialNumber: %d \r\nbNumConfigurations:
%d \r\n"),
                UsbDeviceDescriptor.bLength,UsbDeviceDescriptor.bDescriptorType,
UsbDeviceDescriptor.bcdUSB,UsbDeviceDescriptor.bDeviceClass,
                UsbDeviceDescriptor.bDeviceSubClass,
UsbDeviceDescriptor.bDeviceProtocol, UsbDeviceDescriptor.bMaxPacketSize0,
                UsbDeviceDescriptor.idVendor, UsbDeviceDescriptor.idProduct,
UsbDeviceDescriptor.bcdDevice, UsbDeviceDescriptor.iManufacturer,
UsbDeviceDescriptor.iProduct,
                UsbDeviceDescriptor.iSerialNumber,
UsbDeviceDescriptor.bNumConfigurations);
```

```
        str=str_pre+str;
    GetDlgItem(IDC_Display)->SetWindowText(str);

        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();
}


void CUSB_DialogDlg::OnBnClickedClear()
{
        // TODO: Add your control notification handler code here
        CTime ctime=CTime::GetCurrentTime();
        CString str;
        str.Format(_T("USB Driver Testing Started at: %d-%2d-%2d %02d:%02d
\r\n"),ctime.GetYear(),ctime.GetMonth(),ctime.GetDay(),ctime.GetHour(),ctime.GetMinut
e());
        GetDlgItem(IDC_Display)->SetWindowText(str);
    str.ReleaseBuffer();
}

void CUSB_DialogDlg::OnBnClickedConfiginfo()
{
        // TODO: Add your control notification handler code here
        CString str, str_pre;
        PUSB_CONFIGURATION_DESCRIPTOR P_UsbConfigDescriptor;
        P_UsbConfigDescriptor = (PUSB_CONFIGURATION_DESCRIPTOR)
LocalAlloc(LMEM_FIXED, 100);
        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);
    HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
    }


        DWORD junk;

        if (!DeviceIoControl(hDevice,
                                                 USBSample_IOCTL_802,
                                                 NULL,
                                                 0,
                                                 (LPVOID) P_UsbConfigDescriptor,
                                                 100,
                                                 &junk,
                                                 (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
                exit(1);
        }

    GetDlgItem(IDC_Display)->GetWindowText(str_pre);
        str.Format(_T("\r\nUSB CONFIGURATION DESCRIPTOR\r\nbLength: %d
\r\nbDescriptorType: %d \r\nwTotalLength: %d \r\nbNumInterfaces: %d
\r\nbConfigurationValue: %d \r\niConfiguration: %d \r\nbmAttributes: %d \r\nMaxPower:
%d \r\n")
                ,P_UsbConfigDescriptor->bLength,P_UsbConfigDescriptor-
>bDescriptorType,P_UsbConfigDescriptor->wTotalLength,P_UsbConfigDescriptor-
>bNumInterfaces,P_UsbConfigDescriptor->bConfigurationValue,P_UsbConfigDescriptor-
>iConfiguration
                ,P_UsbConfigDescriptor->bmAttributes,P_UsbConfigDescriptor-
>MaxPower);
    str=str_pre+str;
    GetDlgItem(IDC_Display)->SetWindowText(str);

        LocalFree(P_UsbConfigDescriptor);
        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();
}

void CUSB_DialogDlg::OnBnClickedInferfaceinfo()
{
        // TODO: Add your control notification handler code here
        CString str, str_pre;
        PUSB_CONFIGURATION_DESCRIPTOR    P_UsbConfigDescriptor;
        PUSB_INTERFACE_DESCRIPTOR     P_UsbInterfaceDescriptor;
        P_UsbConfigDescriptor = (PUSB_CONFIGURATION_DESCRIPTOR)
LocalAlloc(LMEM_FIXED, 100);
        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);
    HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
    }


        DWORD junk;

        if (!DeviceIoControl(hDevice,
                                         USBSample_IOCTL_802,
                                         NULL,
                                         0,
                                         (LPVOID) P_UsbConfigDescriptor,
                                         100,
                                                 &junk,
                                                 (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
                exit(1);
        }

        P_UsbInterfaceDescriptor=(PUSB_INTERFACE_DESCRIPTOR) ( (UCHAR
*)P_UsbConfigDescriptor+P_UsbConfigDescriptor->bLength);

    GetDlgItem(IDC_Display)->GetWindowText(str_pre);
        str.Format(_T("\r\nUSB INTERFACE DESCRIPTOR\r\nbLength: %d
\r\nbDescriptorType: %d \r\nbInterfaceNumber: %d \r\nbAlternateSetting: %d
\r\nbNumEndpoints: %d \r\nbInterfaceClass: %d \r\nbInterfaceSubClass: %d
\r\nbInterfaceProtocol: %d \r\niInterface: %d \r\n"),
                P_UsbInterfaceDescriptor->bLength,P_UsbInterfaceDescriptor-
>bDescriptorType,P_UsbInterfaceDescriptor->bInterfaceNumber,
P_UsbInterfaceDescriptor->bAlternateSetting, P_UsbInterfaceDescriptor->bNumEndpoints,
                P_UsbInterfaceDescriptor->bInterfaceClass,P_UsbInterfaceDescriptor-
>bInterfaceSubClass,P_UsbInterfaceDescriptor-
>bInterfaceProtocol,P_UsbInterfaceDescriptor->iInterface);
    str=str_pre+str;
    GetDlgItem(IDC_Display)->SetWindowText(str);

        LocalFree(P_UsbConfigDescriptor);
        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();
}

void CUSB_DialogDlg::OnBnClickedEndpointsinfo()
{
        // TODO: Add your control notification handler code here
        int i;
        CString str, str_pre;
        PUSB_CONFIGURATION_DESCRIPTOR    P_UsbConfigDescriptor;
        PUSB_ENDPOINT_DESCRIPTOR     P_UsbEndpointDescriptor;
        P_UsbConfigDescriptor = (PUSB_CONFIGURATION_DESCRIPTOR)
LocalAlloc(LMEM_FIXED, 100);
        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);
    HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
    }
```

```
        DWORD junk;

        if (!DeviceIoControl(hDevice,
                                                  USBSample_IOCTL_802,
                                                  NULL,
                                                  0,
                                                  (LPVOID) P_UsbConfigDescriptor,
                                                  100,
                                                  &junk,
                                                  (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
                exit(1);
        }

        for(i=0;i<2;i++){
                P_UsbEndpointDescriptor=(PUSB_ENDPOINT_DESCRIPTOR) ( (UCHAR
*)P_UsbConfigDescriptor+9+9+7*i);

                GetDlgItem(IDC_Display)->GetWindowText(str_pre);
                str.Format(_T("\r\nUSB ENDPOINT DESCRIPTOR\r\nbLength: %d
\r\nbDescriptorType: %d \r\nbEndpointAddress: %d \r\nbmAttributes: %d
\r\nwMaxPacketSize: %d \r\nbInterval: %d \r\n"),
                        P_UsbEndpointDescriptor->bLength,P_UsbEndpointDescriptor-
>bDescriptorType,P_UsbEndpointDescriptor->bEndpointAddress, P_UsbEndpointDescriptor-
>bmAttributes, P_UsbEndpointDescriptor->wMaxPacketSize, P_UsbEndpointDescriptor-
>bInterval);
                str=str_pre+str;
                GetDlgItem(IDC_Display)->SetWindowText(str);
        }
                LocalFree(P_UsbConfigDescriptor);
                str.ReleaseBuffer();
                str_pre.ReleaseBuffer();
}



void CUSB_DialogDlg::OnBnClickedSwitch()
{
        // TODO: Add your control notification handler code here
        UCHAR * buffer;
        CString str, str_pre;
        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);
        buffer = (UCHAR *) LocalAlloc(LMEM_FIXED, 1);

    HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                          NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
    }


        DWORD junk;

        if (!DeviceIoControl(hDevice,
                                                USBSample_IOCTL_803,
                                                NULL,
                                                0,
                                                (LPVOID) buffer,
                                                1,
                                                &junk,
                                                (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
        //      exit(1);
        }

        GetDlgItem(IDC_Display)->GetWindowText(str_pre);

        str.Format(_T("\r\nSWITCH STATUS: 0'x%x \r\n"),*buffer);
        str=str_pre+str;
        GetDlgItem(IDC_Display)->SetWindowText(str);

        LocalFree(buffer);
        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();
}


void CUSB_DialogDlg::OnBnClickedBulktest()
{
        // TODO: Add your control notification handler code here
        //wchar_t * input_buffer;
        wchar_t * output_buffer;
        int i;
        CString str, str_pre,str_buf, str_temp, str_combine;
        CString   send_context, received_context;
        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);

    HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                          GENERIC_READ|GENERIC_WRITE,
                          FILE_SHARE_READ | FILE_SHARE_WRITE,
                          NULL,
                          OPEN_EXISTING,
                          0,
                          NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
    }



    GetDlgItemText(IDC_BULKIN,send_context);
        if (send_context.GetLength()==0) return;

    output_buffer = (wchar_t *) LocalAlloc(LMEM_FIXED, 2*send_context.GetLength());

        DWORD junk;
        if (!DeviceIoControl(hDevice,
                                                 USBSample_IOCTL_804,
                                                 send_context.GetBuffer(),
                                                 2*send_context.GetLength(),
                                                 (LPVOID) output_buffer,
                                                 2*send_context.GetLength(),
                                                 &junk,
                                                 (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
        //      exit(1);
        }

        wchar_t   *test= send_context.GetBuffer();

        for(i=0;i<send_context.GetLength();i++){
                str_temp.Format(_T("%c"),*(output_buffer+i));
        str_combine=str_combine+str_temp;
        }

        str.Format(_T("\r\nBULK TRANSACTION LETTERS: %s \r\n"),str_combine);
        GetDlgItem(IDC_Display)->GetWindowText(str_pre);
        str=str_pre+str;
        GetDlgItem(IDC_Display)->SetWindowText(str);

        send_context.ReleaseBuffer();
        LocalFree(output_buffer);
        str_buf.ReleaseBuffer();
        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();
}


void CUSB_DialogDlg::OnEnChangeBulkin()
{
        // TODO:  If this is a RICHEDIT control, the control will not
        // send this notification unless you override the CDialog::OnInitDialog()
        // function and call CRichEditCtrl().SetEventMask()
        // with the ENM_CHANGE flag ORed into the mask.
        // TODO:  Add your control notification handler code here

}


void CUSB_DialogDlg::OnBnClickedLedbarread()
{
        // TODO: Add your control notification handler code here
    UCHAR * buffer;
        UCHAR a,b;
        CString str, str_pre;
        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);
        buffer = (UCHAR *) LocalAlloc(LMEM_FIXED, 1);

    HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
                exit(1);
    }


        DWORD junk;

        if (!DeviceIoControl(hDevice,
                                                 USBSample_IOCTL_805,
                                                 NULL,
                                                 0,
                                                 (LPVOID) buffer,
                                                 1,
                                                 &junk,
                                                 (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
        //      exit(1);
        }

        GetDlgItem(IDC_Display)->GetWindowText(str_pre);

        a=*buffer;

    b=(a>>5)|(a<<3);
        str.Format(_T("\r\nLED BAR STATUS: 0'x%x \r\n"),b);
        str=str_pre+str;
        GetDlgItem(IDC_Display)->SetWindowText(str);
        LocalFree(buffer);
        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();
}


void CUSB_DialogDlg::OnBnClickedLedbarset()
{
        // TODO: Add your control notification handler code here
        UCHAR a;
        CString str, str_pre;
        CButton    *pButton;
        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);

    a=0;
        pButton = (CButton *)GetDlgItem(IDC_CHECK1);
    a+= 32*pButton->GetCheck();

        pButton = (CButton *)GetDlgItem(IDC_CHECK2);
    a+= 64*pButton->GetCheck();

        pButton = (CButton *)GetDlgItem(IDC_CHECK3);
    a+= 128*pButton->GetCheck();

        pButton = (CButton *)GetDlgItem(IDC_CHECK4);
    a+= pButton->GetCheck();

        pButton = (CButton *)GetDlgItem(IDC_CHECK5);
    a+= 2*pButton->GetCheck();

        pButton = (CButton *)GetDlgItem(IDC_CHECK6);
    a+= 4*pButton->GetCheck();

        pButton = (CButton *)GetDlgItem(IDC_CHECK7);
    a+= 8*pButton->GetCheck();

        pButton = (CButton *)GetDlgItem(IDC_CHECK8);
    a+= 16*pButton->GetCheck();

    HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                        GENERIC_READ|GENERIC_WRITE,
                        FILE_SHARE_READ | FILE_SHARE_WRITE,
                        NULL,
                        OPEN_EXISTING,
                        0,
                        NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
                exit(1);
    }
```

```
        DWORD junk;

        //a=170;

        if (!DeviceIoControl(hDevice,
                                                    USBSample_IOCTL_806,
                                                    &a,
                                                    1,
                                                    NULL,
                                                    0,
                                                    &junk,
                                                    (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
        //      exit(1);
        }

        GetDlgItem(IDC_Display)->GetWindowText(str_pre);

        str.Format(_T("\r\nSET LED BAR SUCCESS. \r\n"));
        str=str_pre+str;
        GetDlgItem(IDC_Display)->SetWindowText(str);

//      LocalFree(buffer);
        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();
}

void CUSB_DialogDlg::OnBnClicked7ledread()
{
        // TODO: Add your control notification handler code here
    UCHAR * buffer;
        UCHAR a,b;
        CString str, str_pre;
        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);
        buffer = (UCHAR *) LocalAlloc(LMEM_FIXED, 1);

    HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
                exit(1);
    }
```

```
        DWORD junk;

        if (!DeviceIoControl(hDevice,
                                                USBSample_IOCTL_807,
                                                NULL,
                                                0,
                                                (LPVOID) buffer,
                                                1,
                                                &junk,
                                                (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
        //      exit(1);
        }

        GetDlgItem(IDC_Display)->GetWindowText(str_pre);

        a=*buffer;

    b=(a>>5)|(a<<3);
        str.Format(_T("\r\nLED BAR STATUS: 0'x%x \r\n"),b);
        str=str_pre+str;
        GetDlgItem(IDC_Display)->SetWindowText(str);

        LocalFree(buffer);
        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();
}

void CUSB_DialogDlg::OnBnClickedt()
{
        // TODO: Add your control notification handler code here
                UCHAR a;
        CString str, str_pre;
        CButton    *pButton;
        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);

    a=0;
        pButton = (CButton *)GetDlgItem(IDC_CHECK9);
    a+= 0x01*pButton->GetCheck();

        pButton = (CButton *)GetDlgItem(IDC_CHECK10);
    a+= 0x40*pButton->GetCheck();

        pButton = (CButton *)GetDlgItem(IDC_CHECK11);
    a+= 0x02*pButton->GetCheck();

        pButton = (CButton *)GetDlgItem(IDC_CHECK12);
    a+= 0x20*pButton->GetCheck();

        pButton = (CButton *)GetDlgItem(IDC_CHECK13);
    a+= 0x10*pButton->GetCheck();

        pButton = (CButton *)GetDlgItem(IDC_CHECK14);
    a+= 0x04*pButton->GetCheck();

        pButton = (CButton *)GetDlgItem(IDC_CHECK15);
    a+= 0x80*pButton->GetCheck();

        pButton = (CButton *)GetDlgItem(IDC_RADIO9);
    a+= 0x08*pButton->GetCheck();

    HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                        GENERIC_READ|GENERIC_WRITE,
                        FILE_SHARE_READ | FILE_SHARE_WRITE,
                        NULL,
                        OPEN_EXISTING,
                        0,
                        NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
                exit(1);
    }


        DWORD junk;

        if (!DeviceIoControl(hDevice,
                                        USBSample_IOCTL_808,
                                        &a,
                                        1,
                                        NULL,
                                        0,
                                        &junk,
                                        (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
        //      exit(1);
        }

//      LocalFree(buffer);
        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();
}


HANDLE m_timerHandle=NULL;
HANDLE hDevice_speed;
PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo_speed;
```

```
char data_send[512],data_rcv[512];
short a=1;
LARGE_INTEGER lpFrequency, test_start, test_end;
DWORD send_counter;

VOID CALLBACK TimerProc(PVOID lpParam, BOOLEAN TimerOrWaitFired)
{
    DWORD junk;

        ////////////// BULK trans ///////////
        if (!DeviceIoControl( hDevice_speed,
                                          USBSample_IOCTL_804,
                                          data_send,
                                          512,
                                          data_rcv,
                                          512,
                                          &junk,
                                          (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
        //      exit(1);
        }

    /////////////////////////////////////


   if (a==128)
        a=1;
   else a=a*2;
   if (!DeviceIoControl( hDevice_speed,
                                         USBSample_IOCTL_806,
                                         &a,
                                         1,
                                         NULL,
                                         0,
                                         &junk,
                                         (LPOVERLAPPED) NULL)
           )
        {
                //   return 0;
        }

    send_counter++;

        if(send_counter==1)  QueryPerformanceCounter(&test_start);
    QueryPerformanceCounter(&test_end);

}
void CUSB_DialogDlg::OnBnClickedSpeedtest()
{
        // TODO: Add your control notification handler code here
         DeviceInfo_speed=GetDevicePath((LPGUID)& USBSample_DEVINTERFACE_GUID);
         hDevice_speed = CreateFile(DeviceInfo_speed->DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );
        if (hDevice_speed == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
    }



    send_counter=0;
        DWORD elTime=5;
        if(!CreateTimerQueueTimer(
                &m_timerHandle,
                NULL,
                TimerProc,
                //&hDevice,
                this,
                0,
                elTime,
                WT_EXECUTEINTIMERTHREAD))
        {
        ;
        }
}

void CUSB_DialogDlg::OnBnClickedStopspeedtest()
{
        // TODO: Add your control notification handler code here
        QueryPerformanceFrequency(&lpFrequency);

    if(m_timerHandle!=NULL)
        {
                DeleteTimerQueueTimer(NULL,
                                                        m_timerHandle,
                                                        NULL
                                                    );
        }


        DWORD time_spend = (test_end.QuadPart-
test_start.QuadPart)/lpFrequency.QuadPart;
        DWORD transfer_speed=send_counter/time_spend;
        CString str, str_pre;

        GetDlgItem(IDC_Display)->GetWindowText(str_pre);

        str.Format(_T("\r\nPacket Transfered: %d\r\n"), send_counter);
        str_pre=str_pre+str;
        str.Format(_T("\r\nTime Spend(sec): %d\r\n"), time_spend);
        str_pre=str_pre+str;
    str.Format(_T("\r\nAverage Transfer Speed(packet/sec): %d\r\n"), transfer_speed);
        str=str_pre+str;
        GetDlgItem(IDC_Display)->SetWindowText(str);

//      LocalFree(buffer);
        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();

}


#define  Num 512
void CUSB_DialogDlg::OnBnClickedBulkread() // read one byte from endpoint of the USB
{
        // TODO: Add your control notification handler code here
//      char * output_buffer;
        CString str, str_pre, str_temp1, str_temp2;;

        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);

    HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
    }


    //data_buffer = (char*) LocalAlloc(LMEM_FIXED, 512);

    unsigned char rec_letter[Num];
        DWORD junk;
        if (!DeviceIoControl(hDevice,
                                                 USBSample_IOCTL_811,
                                                 NULL,
                                                 0,
                                                 &rec_letter,
                                                 Num,
                                                 &junk,
                                                 (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
        //      exit(1);
        }

        int i;


        unsigned int temp1, temp2;
        wchar_t  display_letter[Num*3];
        for(i=0;i<Num; i++)
        {
           temp1=rec_letter[i]&0x0F;
           temp2=rec_letter[i]>>4;

           if(temp1 <= 9 && temp1 >= 0) temp1=temp1+48;
           else if(temp1 <= 15 && temp1 >= 10) temp1=temp1+55;
       //else if(str_in.GetAt(i) <= 102  && str_in.GetAt(i) >= 97)
temp=str_in.GetAt(i)-87;
           else return;

           if(temp2 <= 9 && temp2 >= 0) temp2=temp2+48;
           else if(temp2 <= 15 && temp2 >= 10) temp2=temp2+55;
       else return;

       display_letter[3*i]=temp2;
           display_letter[3*i+1]=temp1;
           display_letter[3*i+2]=' ';
        }

        //str.Empty();
        for(i=0;i<Num*3; i++)
        {
       str_temp1.Format(_T("%c"),display_letter[i]);
           str_temp2 = str_temp2 + str_temp1;
        }

        //str_temp.Format(_T("%c"),display_letter[i]);
    str.Format(_T("\r\nReceived data: %s \r\n"),str_temp2);

        GetDlgItem(IDC_Display)->GetWindowText(str_pre);
        str=str_pre+str;
        GetDlgItem(IDC_Display)->SetWindowText(str);

        //LocalFree(output_buffer);
        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();
}

void CUSB_DialogDlg::OnBnClickedBulkwrite()
{
        // TODO: Add your control notification handler code here
//      wchar_t * output_buffer;
        int i;
        unsigned char letter_in[512], temp;
        CString str_in, str, str_pre,str_buf, str_temp, str_combine;
        CString   send_context, received_context;
        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);

    HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
    }

    for(i=0;i<512;i++){
      letter_in[i]=0;
   }
 //   output_buffer = (wchar_t *) LocalAlloc(LMEM_FIXED, 2*send_context.GetLength());
   GetDlgItem(IDC_BULKIN)->GetWindowText(str_in);
   int atext1 = str_in.GetAt(0);
   int atext2 = str_in.GetAt(1);
   int atext3 = str_in.GetAt(2);
   int atext4 = str_in.GetAt(3);
   int ttt = str_in.GetLength();
   if(str_in.GetLength() == 0) return;
   else if((str_in.GetLength()%2) != 0) return;
   bool high4_flag = TRUE;
   int index=0;
   for(i=0;i<str_in.GetLength();i++){

           if(str_in.GetAt(i) <= 57 && str_in.GetAt(i) >= 48) temp=str_in.GetAt(i)-
48;
           else if(str_in.GetAt(i) <= 70 && str_in.GetAt(i) >= 65)
temp=str_in.GetAt(i)-55;
       else if(str_in.GetAt(i) <= 102  && str_in.GetAt(i) >= 97)
temp=str_in.GetAt(i)-87;
           else return;

           if (high4_flag == TRUE){
                     letter_in[index] = (temp<<4);
                         high4_flag =FALSE;
           }
           else{
                     letter_in[index] += temp;
                         high4_flag =TRUE;
                         index ++;
           }

   }

   int send_text0=letter_in[0];
   int send_text1=letter_in[1];
   int send_text2=letter_in[2];
   int send_text3=letter_in[3];
   int send_text4=letter_in[4];
   int send_text5=letter_in[5];
   int send_text6=letter_in[6];
   int send_text7=letter_in[7];
   int send_text8=letter_in[8];
   int send_text9=letter_in[9];

        DWORD junk;
        if (!DeviceIoControl(hDevice,
                                                USBSample_IOCTL_809,
                                                &letter_in,
                                                512,
                                                NULL,
                                                0,
                                                &junk,
                                                (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
        //      exit(1);
        }


        str.Format(_T("\r\nSent data: %c \r\n"),letter_in);
        GetDlgItem(IDC_Display)->GetWindowText(str_pre);
        str=str_pre+str;
        GetDlgItem(IDC_Display)->SetWindowText(str);

        send_context.ReleaseBuffer();
//      LocalFree(output_buffer);
        str_buf.ReleaseBuffer();
        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();
}

void CUSB_DialogDlg::OnBnClickedBufstatus()
{
        // TODO: Add your control notification handler code here
        char * output_buffer;
        CString str, str_pre, str_temp1, str_temp2;;

        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);

    HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
```

```
GetLastError());
    }


    //data_buffer = (char*) LocalAlloc(LMEM_FIXED, 512);
    //unsigned char rec_letter[1];
        int rec_letter[1];
        DWORD junk;
        if (!DeviceIoControl(hDevice,
                                                    USBSample_IOCTL_812,
                                                    NULL,
                                                    0,
                                                    &rec_letter,
                                                    4,
                                                    &junk,
                                                    (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
        //      exit(1);
        }

        int test0 = rec_letter[0];
    str.Format(_T("\r\nBuffer Status: %d\r\n"),rec_letter[0]);

        GetDlgItem(IDC_Display)->GetWindowText(str_pre);
        str=str_pre+str;
        GetDlgItem(IDC_Display)->SetWindowText(str);

        //LocalFree(output_buffer);
        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();
}

void CUSB_DialogDlg::OnBnClickedSttimer()
{
        // TODO: Add your control notification handler code here
        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);

        HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
    }

        DWORD junk;
        if (!DeviceIoControl(hDevice,
                                                        USBSample_IOCTL_813,
                                                        NULL,
                                                        0,
                                                        NULL,
                                                        0,
                                                        &junk,
                                                        (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
        //      exit(1);
        }

}

void CUSB_DialogDlg::OnBnClickedStoptimer()
{
        // TODO: Add your control notification handler code here
        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);

        HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
    }

        DWORD junk;
        if (!DeviceIoControl(hDevice,
                                                        USBSample_IOCTL_814,
                                                        NULL,
                                                        0,
                                                        NULL,
                                                        0,
                                                        &junk,
                                                        (LPOVERLAPPED) NULL)
           )
        {
                printf("ERROR: DeviceIoControl returns %0x.\n", GetLastError());
        //      exit(1);
        }
}

void CUSB_DialogDlg::OnBnClickedstart1msres()
{
        // TODO: Add your control notification handler code here
        timeBeginPeriod(1);
}
```

```
void CUSB_DialogDlg::OnBnClickedend1msres()
{
        // TODO: Add your control notification handler code here
        timeEndPeriod(1);
}

void CUSB_DialogDlg::OnBnClickedBufferread()
{
        // TODO: Add your control notification handler code here
        CString str, str_pre;
        unsigned char data_buffer[512*128];
        DWORD nNumberOfBytesToRead = 512*128;
        DWORD lpNumberOfBytesRead;
        int i,k;

        PSP_DEVICE_INTERFACE_DETAIL_DATA  DeviceInfo=GetDevicePath((LPGUID)&
USBSample_DEVINTERFACE_GUID);

        HANDLE hDevice = CreateFile(DeviceInfo->DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
    }

           ReadFile(
                                        hDevice,
                                        data_buffer,
                                        nNumberOfBytesToRead,
                                        &lpNumberOfBytesRead,
                                        NULL
                                );

    int test0 = nNumberOfBytesToRead;
    int test1 = lpNumberOfBytesRead;

    int Num_Frame = lpNumberOfBytesRead/512;

        unsigned int temp1, temp2;
        wchar_t  display_letter[Num*3];
        for(k=0;k<Num_Frame;k++)
        {
                CString  str_temp1, str_temp2, str_data, str_num;
                for(i=0;i<Num; i++)
                {
                  temp1=data_buffer[i+512*k]&0x0F;
                  temp2=data_buffer[i+512*k]>>4;

                  if(temp1 <= 9 && temp1 >= 0) temp1=temp1+48;
```

```
                    else if(temp1 <= 15 && temp1 >= 10) temp1=temp1+55;
                    //else if(str_in.GetAt(i) <= 102  && str_in.GetAt(i) >= 97)
temp=str_in.GetAt(i)-87;
                    else return;

                    if(temp2 <= 9 && temp2 >= 0) temp2=temp2+48;
                    else if(temp2 <= 15 && temp2 >= 10) temp2=temp2+55;
                    else return;

                    display_letter[3*i]=temp2;
                    display_letter[3*i+1]=temp1;
                    display_letter[3*i+2]=' ';
                 }

                 //str.Empty();
                 for(i=0;i<Num*3; i++)
                 {
                    str_temp1.Format(_T("%c"),display_letter[i]);
                    str_temp2 = str_temp2 + str_temp1;
                 }

         str_num.Format(_T("\r\nReceived data frame %d: \r\n"),k);
                 str_data.Format(_T("\r\n%s %s \r\n"),str_num,str_temp2);

                 str=str+str_data;

                 str_temp1.ReleaseBuffer();
                 str_temp2.ReleaseBuffer();
                 str_data.ReleaseBuffer();
                 str_num.ReleaseBuffer();
         }
             GetDlgItem(IDC_Display)->GetWindowText(str_pre);
             str=str_pre+str;
                 GetDlgItem(IDC_Display)->SetWindowText(str);
         //LocalFree(output_buffer);
         str.ReleaseBuffer();
         str_pre.ReleaseBuffer();




}


VOID CALLBACK TimerProc_DataTest(PVOID lpParam, BOOLEAN TimerOrWaitFired)
{

         DWORD junk, lpNumberOfBytesRead;
         int i,j;
     int frame_rcvd;
     unsigned char temp[512*128];


         ////////////// BULK read ///////////
```

```
DeviceInfo=CUSB_DialogDlg::GetDevicePath((LPGUID)& USBSample_DEVINTERFACE_GUID);

        HANDLE hDevice = CreateFile(((CUSB_DialogDlg*)lpParam)->DeviceInfo_timer-
>DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );
        if (hDevice == INVALID_HANDLE_VALUE) {
                printf("ERROR opening device: (%0x) returned from CreateFile\n",
GetLastError());
    }

        ReadFile(
                          hDevice,
              &temp,
              512*128,
                          &lpNumberOfBytesRead,
                          NULL
                        );

        frame_rcvd = lpNumberOfBytesRead/512; // count the received frames

        // retrieve frame mark
    int num_frame = 17;
    int test1 = temp[512*0+24*5];
        int test2 = temp[512*0+24*6];
        int test3 = temp[512*0+24*7];
        int test4 = temp[512*0+24*17+4];

        for(i=0;i<frame_rcvd;i++)
        {
                for(j=0;j<4;j++)
                {
                        //((CUSB_DialogDlg*)lpParam)->frame_mark_cur =
temp[512*i+24*num_frame]*(2^24);
                        int test5 = temp[512*i+24*num_frame+0];
                        int test6 = temp[512*i+24*num_frame+1];
                        int test7 = temp[512*i+24*num_frame+2];
                        int test8 = temp[512*i+24*num_frame+3];

                    ((CUSB_DialogDlg*)lpParam)->frame_mark_cur[0] =
temp[512*i+24*num_frame+0];
                        ((CUSB_DialogDlg*)lpParam)->frame_mark_cur[1] =
temp[512*i+24*num_frame+1];
                        ((CUSB_DialogDlg*)lpParam)->frame_mark_cur[2] =
temp[512*i+24*num_frame+2];
                        ((CUSB_DialogDlg*)lpParam)->frame_mark_cur[3] =
temp[512*i+24*num_frame+3];
                }

                ((CUSB_DialogDlg*)lpParam)->Diff0.Add(((CUSB_DialogDlg*)lpParam)-
    return;
}

void CUSB_DialogDlg::OnBnClickedEnd()
{
        // TODO: Add your control notification handler code here

        DeleteTimerQueueTimer(NULL,
                                  m_timerHandle,
                                                  INVALID_HANDLE_VALUE
                                                  );
    timeEndPeriod(1);

        CString str,str_temp, str_pre;

        for(DWORD i =0; i< Diff0.GetSize(); ++i)
        {
           str_temp.Format(_T("%d"),Diff0.GetAt(i));
           str = str + str_temp;

           str_temp.Format(_T("  %d"),Diff1.GetAt(i));
           str = str + str_temp;

           str_temp.Format(_T("  %d"),Diff2.GetAt(i));
           str = str + str_temp;

           str_temp.Format(_T("  %d\r\n"),Diff3.GetAt(i));
           str = str + str_temp;

           if(Diff_End_Flag.GetAt(i) == true)
           {
str_temp.Format(_T("%s\r\n"),_T("++++++++++++++++++++++++++++++++++++"));
                   str = str + str_temp;
           }


        }

        GetDlgItem(IDC_Display)->GetWindowText(str_pre);
        str=str_pre+str;
        GetDlgItem(IDC_Display)->SetWindowText(str);

        str.ReleaseBuffer();
        str_pre.ReleaseBuffer();

}

void CUSB_DialogDlg::OnTimer(UINT_PTR nIDEvent)
{
        // TODO: Add your message handler code here and/or call default
        CDialog::OnTimer(nIDEvent);
}
```

BIBLIOGRAPHY

Barker A. T., Jalinous R. and Freeston I. L., “Non-Invasive Magnetic Stimulation of Human Motor Cortex,” Lancet, 1(8437): 1106-1107, 1985.

Basseville M. and Nikiforov I. V., “Detection of abrupt changes: Theory and Application,” Prentice Hall Press, 1993.

Bédard M., Agid Y. , Chouinard S. , Fahn S., Korczyn A. and Lesperace P., “Mental and Behavioral Dysfunction in Movement Disorders,” Humana Press, 2003.

Besio W., Koka K., Aakula R., Dai W., “Tri-polar Concentric Ring Electrode Development for Laplacian Electroencephalography,” IEEE Trans BME, Vol. 53, No. 5, pp. 926-933, 2006.

Besio W., Makeyev O., Medvedev A., Gale K., “Effects of transcranial focal stimulation via tripolar concentric ring electrodes on pentylenetetrazole-induced seizures in rats”, Epilepsy Research, Jan, 2013.

Biggins C., Fein G., Raz J. and Amir A., “Artifactually high coherences results from using spherical spline computation of scalp current density,” Electroenceph. and Clin. Neurophysiol., 79: 413-419, 1991.

Bragin A., Wilson C. L., Fields T., Fried I. and Engel Jr J., “Analysis of seizure onset on the basis of wideband EEG recordings, ” Epilepsia 46 59–63, 2005.

Chander R., Urrestarrazu E. and Gotman J., “Automatic detection of high frequency oscillations in human intracerebral EEGs,” Epilepsia 47 (4) 37, 2006.

Chandler D., Bisasky J., Stanislaus J. and Mohsenin T., “Real-time multi-channel seizure detection and analysis hardware, ” Biomedical Circuits and Systems Conference 2011, 41-44, 2011.

Collura T., “History and evaluation of electroencephalographic instruments and techniques,” J. Clin Neurophysiol., 10(4):476-504, 1993.

Cuffin B. N., and Cohen D., “Comparison of the magnetoencephalogram and electroencephalogram,” Electroencephalography and Clinical Neurophysiology, 47, 132-146, 1979.

Fein G., Raz J., and Turetsky B., “Brain electrical activity: the promise of new technologies,” In: S. Zakhari and E. Witt (Eds.)., Imaging in Alcohol Research. Proc. of a Workshop on Imaging in Alcohol Research, Wild Dunes, SC, 9–11: 49-78, 1991.

Gigante P. R., Gooman R. R., “Responsive neurostimulation for the treatment of epilepsy,” Neurosurg. Clin. N Am., 22(4): 477-480, 2011.

Gloor P., “Contributions of electroencephalography and electrocorticography in the neurosurgical treatment of the epilepsies,” Adv Neurol, 8:59–105, 1975.

Gotman J., “Automatic detection of seizures and spikes, ” Journal of Clinical Neurophysiology, 16:130–140, 1999.

Gotman J., “Automatic recognition of epileptic seizures in the EEG,” Electroencephalography and Clinical Neurophysiology, 54:530-540, 1982.

Groves D. A., Brown V. J., “Vagal nerve stimulation: A review of its applications and potential mechanisms that mediate its clinical effects,” Neurosci Biobehav. Rev., 29(3): 493-500, 2005.

Hansen P., “The truncated SVD as a method of regularization,” BIT Numerical Mathematics, Vol. 27, Issue 4, 534-553, 1987.

He B., “Brain electrical source imaging: scalp laplacian mapping and cortical imaging,” Crit. Rev. Biomed. Eng., vol. 27, pp. 149–188, 1999.

He B., Yao D., Lian J., and Wu D., “An Equivalent Current Source Model and Laplacian Weighted Minimum Norm Current Estimates of Brain Electrical Activity,” IEEE Trans. BME, Vol. 49, No. 4, 2002.

He B., Wang Y., and Wu D., “Estimating cortical potentials from scalp EEGs in a realistically shaped inhomogeneous head model by means of the boundary element method,” IEEE Trans. BME, Vol. 46, No. 10, 1999:713-719, 2001.

Hjorth B., “An on-line transformation of EEG scalp potentials into orthogonal source derivations,” Electroencephalography and Clinical Neurophysiology, Vol.. 39,526-530, 1975.

Kester W., “Practical design techniques for sensor signal conditioning,” Prentice Hall Press, 1999.

Koka K., Besio W., "Improvement of spatial selectivity and decrease of mutual information of tri polar concentric ring electrodes," J. Neuroscience Methods, 165(2): 216-222, 2007.

Kringelbach M. L., Jenkinson N., Owen S. L. and Aziz T. Z., “Translational principles of deep brain stimulation,” Nat. Rev. Neurosci., 8(8): 623-635, 2007.

Le J., Menon V., and Gevins A., “Local estimate of surface Laplacian derivation on a realistically shaped scalp surface and its performance on noisy data,” Electroencephalography and Clinical Neurophysiology., 92: 433-441, 1994.

Makeyev O., Liu X., Luna-Munguía H., Rogel-Salazar G., Mucio-Ramirez S., Liu Y., Sun Y. L., Kay S. M. and Besio W. G., “Toward a noninvasive automatic seizure control system in rats with transcranial focal stimulations via tripolar concentric ring electrodes,” IEEE Trans. Neural Syst. Rehabil Eng., 20(4): 422-431, 2012.

Montgomery D.C., “Design and analysis of experiments, Wiley,” Hoboken, 2004.

Nunez P. L., Silberstein R. B., Cadiush P. J., Wijesinghe J., Westdorp A. F., and Srinivasan R., “A theoretical and experimental study of high resolution EEG based on surface Laplacians and cortical imaging,” Electroencephalography and Clinical Neurophysiology vol. 90, pp. 40-57, 1994.

Ocak H., “Automatic detection of epileptic seizures in EEG using discrete wavelet transform and approximate entropy,” Expert Systems with Applications, 36(2), 2027–2036, 2009.

Oostenveld R., Praamstra P., “The five percent electrode system for high-resolution EEG and ERP measurements,” Clinical Neurophysiol, 112(4): 713-719, 2001.

Parhi K., “VLSI Digital Signal Processing Systems: Design and Implementation,” Wiley-Interscience Press, 1999.

Perrin F., Pernier J., Bertrand O. and Echallier J. F., “Spherical splines for scalp potential and current density mapping,” Electroencephalography and Clinical Neurophysiology 72: 184-1987, 1989.

Polat K., Gunes S., “Classification of epileptiform EEG using a hybrid system based on decision tree classifier and fast Fourier transform,” Applied Mathematics and Computation, 187(2), 1017–1026, 2007.

Quyen M. L. V., Martinerie J., Baulac M., Varela F., “Anticipating epileptic seizures in real time by a non-linear analysis of similarity between EEG recordings,” Neuroreport, Jul 13, 10(10):2149-55, 1999.

Stevanovic D., “Epilepsy - Histological, Electroencephalographic and Psychological Aspects,” Intech Press, 2012.

Tandonnet C., Burle B., Hasbroucq T., Vidal F., “Spatial enhancement of EEG traces by surface Laplacian estimation: comparison between local and global methods,” Clin. Neurophysiol. , 116(1):18-24, 2005.

Texas Instruments, “Low-Noise, 8-Channel, 24-Bit Analog Front-End for Biopotential Measurements,” 2012.

Tzallas A. T., Tsipouras M. G., Fotiadis D. I., “Epileptic seizure detection in EEGs using time-frequency analysis,” IEEE Trans Inf Technol Biomed, 13(5), 703-710, 2009.

Urrestarazu E., Chander R., Dubeau F., Gotman J., “Interictal high-frequency oscillations (100-500 Hz) in the intracerebral EEG of epileptic patients,” Brain., 130:2354-66, 2007.

Utz K. S., Dimova V., Oppenländer K. and Kerkhoff G., “Eectrified minds: transcranial direct current stimulation (tDCS) and galvanic vestibular stimulation (GVS) as methods of non-invasive brain stimulation in neuropsychology--a review of current data and future implications,” Neuropsychologia., 48(10): 2789-2810, 2010.

Wang K., and Befleiter H., “Local polynomial estimate of surface Laplacian,” Brain Topograpgy, Vol 12, Issue 1, 19-29, 1999.

Wilson S., Emerson R., “Spike detection: a review and comparison of algorithms,” Clin Neurophysiol. 2002 Dec, 113(12):1873-81.

World Health Organization (WHO). http://www.who.int/mediacentre/factsheets/fs999/en/. Accessed on May 14, 2013

Worrell G., “High-frequency oscillations recorded on scalp EEG,” Epilepsy Curr., 12(2): 57-58, 2012.