Design and Performance Analysis of an Ultra-Fast Digital Positron Annihilation Lifetime Spectrometer at The Ohio State University

THESIS

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

James Patrick Ralston, B.A.
Graduate Program in Nuclear Engineering

The Ohio State University
2013

Master's Examination Committee:

Dr. Lei Cao, Advisor

Dr. Thomas Blue
Abstract

Positron Annihilation Lifetime Spectroscopy (PALS) is a highly effective technique to quantify the lattice deformation in solid materials. The technique is based on amount of time it takes a free neutron to annihilate in the sample material. Since the timescales of positron lifetime are on the order of a nanosecond, high precision timing measurements are required to obtain useful measurements. Recent innovations in digital data acquisition technology has allowed digital systems to approach and surpass the timing capabilities of their analog counterparts. Digital acquisition systems also have the advantage of a generally simpler setup that offers equal capabilities at a lower cost. The digital system can potentially offer double the counting rates, as each detector in the system can measure both starting and stopping events. The spectrometer designed allows for high performance timing measurements, equal and better than analog and more expensive digital systems. Custom software, developed in C++/C, has been written for the data analysis and data acquisition and allows for very high counting rates. The system hardware was assembled with an ORTEC 905-21 fast plastic scintillation detector and a CAEN N6751 2 GS/s digitizer, the timing performance of the specific hardware combination used has been previously undetermined. The timing performance analysis was conducted with using a Co-60 button sized radiation source to measure coincident gamma rays. Since the gamma rays are emitted from the source at almost the exact same moment, the measured time difference between detected events can be used to measure
the uncertainty in the timing of the spectrometer. The timing resolution of the spectrometer is characterized by the Full Width at Half Maximum (FWHM) of the measured Co-60 spectra. The measured value of the FWHM of the spectrometer was 208.5 ± 1.1 ps.
Dedication

This document is dedicated to my family and my loving fiancée Katie.
Acknowledgments

I would like to first acknowledge my graduate advisor Dr. Lei Cao. Dr. Cao has given supported my time at Ohio State University both financially and professionally. I would like to also thank him for the opportunity to work on this project. I would also like to thank the reactor staff for the use of the radioactive sources used in the performance analysis, without it no measurements could be done.
Vita

June 2007 ............................................................. Carroll High School

2011 .............................................................. B.S. Physics, Ohio University

2011 to present .............................................. Graduate Research Associate, Nuclear Engineering Program, The Ohio State University

Publications


Fields of Study

Major Field: Nuclear Engineering
Table of Contents

Abstract .......................................................................................................................... ii

Dedication ....................................................................................................................... iv

Acknowledgments ........................................................................................................... v

Vita .................................................................................................................................. vi

List of Tables ................................................................................................................... ix

List of Figures .................................................................................................................. x

Chapter 1: Introduction .................................................................................................... 1

Chapter 2: Theory ............................................................................................................ 4

Chapter 3: Analog vs. Digital Systems ........................................................................... 9

Chapter 4: Experimental Approach ................................................................................ 14

   Design Objectives ......................................................................................................... 14

   System Time Resolution ............................................................................................... 15

   System Instrumentation ............................................................................................... 15

Chapter 5: Software ......................................................................................................... 22

Chapter 6: System Calibration ......................................................................................... 26

Chapter 8: Conclusions ................................................................................................... 35
References.......................................................................................................................... 36

Appendix A: NARS_PLS.cpp ................................................................................................38

Appendix B: Analysis.cpp .................................................................................................. 51

Appendix C: analysis.h ...................................................................................................... 55

Appendix D: ConsoleFunctions.cpp .................................................................................. 57

Appendix E: ConsoleFunctions.h ....................................................................................... 64
List of Tables

Table 1: BC-418 fast plastic scintillation properties.......................................................... 17
Table 2: Detector filter threshold for coincident event detection ..................................... 30
Table 3: FWHM values for the different delay and fraction settings applied to coincident gamma rays measured from a Co-60 button source......................................................... 33
List of Figures

Figure 1: Level Diagram of the radioactive Na-22 positron source [26].......................... 5
Figure 2: Measured positron lifetime spectroscopy for both an as-grown and plastically deformed Czochralski-grown Silicon Sample [27]. This spectra is used only to show the form of a typical positron lifetime spectrum................................................................. 6
Figure 3: General schematic of an analog PALS system................................................. 10
Figure 4: General schematic of a digital PALS system.................................................. 11
Figure 5: PALS system overview .................................................................................. 16
Figure 6: ORTEC 905-21 fast plastic detector assemblies mounted for data collection.. 18
Figure 7: N6751 digitizer and N1471 high voltage power supply connected to a model N147X 2-slot NIM crate, all supplied by CAEN S.p.A....................................................... 20
Figure 8: Representation of a digital CFD filter on the output of a PMT following a radiation event. The dots in the lower figure represent the digital samples, highlighting the fact the true event timing information lies between the sampled data.................. 25
Figure 9: The pulse amplitude versus integrated charge of data collected from a Co-60 button source.............................................................................................................. 27
Figure 10: Integrated charge spectra of the events in Figure 8........................................ 27
Figure 11: Single event triggers from a Na-22 source highlighting the 1.237 MeV decay gamma 511 keV annihilation gamma Compton edges................................................. 29
Figure 12: Coincident event spectra from the same Na-22 source used in Figure 10, showing all the same features.

Figure 13: Optimization results after varying the amplitude fraction and signal delay used in the CFD filter. The best performance was found with a signal delay of 4 samples (2 ns) and a constant fraction of 40% of the peak amplitude.

Figure 14: Histogram of measured time difference of between the arrivals of two coincident gamma rays, which is used to define the resolution function of the PALS system.
Chapter 1: Introduction

Positron Annihilation Lifetime spectroscopy (PALS) is a technique used to study the electronic, void and defect structure in materials [1]. The technique measures the lifetime of a positron in matter to determine material properties. Experiments measuring the interaction of positrons in matter began in the 1940s. These experiments used the positron to study the electronic structure of metals and alloys [2][3]. It wasn’t until the 1960s that it was realized positron annihilation experiments could be used to study lattice imperfections in solids [4]. PALS allows for extreme sensitivity to open volume defects in matter, as low as 1 vacancy per 10^6 atoms [5].

PALS works by measuring positron lifetime, the time difference between the emission of a positron and its annihilation. The positron emitted from the source gets trapped in local defects, due to the lack of a repulsive force from the positively charged nucleus in the lattice. The lower electron density in the defect volume increases the average diffusion time [4, 6-8]. The measurement of the change in positron lifetimes allows for the defects in the sample to be quantized.

The study of semiconductors was began in the mid-1980s, using knowledge acquired from years of studies in metals. The study of defects in semiconductors is more difficult than that of metals, due to an increase in the types of point defects seen and the dependence on the charge state of the defect [9]. Polymers present an even more difficult task when using PALS to measure structure, due to the amorphous structure. PALS has
improved in recent years to be the leading technique for studying local free volumes in polymers [10].

PALS has proven to be a very effective and important method in studying the microstructure of semiconductors, little progress has been made in the timing resolution of the experimental setups. Increasing the time resolution of the experiments allows for measuring higher defect rates in material samples [11]. In order to improve the time resolution of PAL several research teams have moved to ultrafast digital systems, which offer several advantages over traditional analog techniques [12-19]. Digital systems also offer the advantage of being more compact than their traditional analog systems. In order to achieve the high timing resolution several discrimination methods must be used, including constant fraction discrimination, pulse amplitude, pulse rise time and the ratio of the rise time and amplitude.

The counting rates possible using digital PALS systems are typically limited by the PC hardware operating the data collection software, as opposed to detection hardware of analog systems. This is because event discrimination is done with the PC or an onboard FPGA [18-20]. FPGA based discrimination requires special programming and limits the complexity of the analysis program [20]. If a PC is to be used for event discrimination, efficient algorithms and thread management will allow count rates to approach that of traditional analog systems [21]. A combination of FPGA hardware for simple filters and discrimination with more complex analysis handled by the PC allows for the counting rate of digital systems to approach that of analog systems. Another advantage of the digital systems is the preservation of the experimental data.
One of the present challenges in building a cost efficient digital PALS system is the cost. Many recent studies have utilized BaF2 crystal detectors, due to their fast timing response and optimal efficiency [5, 11, 13, 22-24]. Large BaF2 detectors are very expensive, but necessary because the smaller detector volumes decreases the likelihood of simultaneous event detection. Larger plastic detectors, with very fast time response, are much less expensive than the BaF2 detectors. The disadvantage of the fast plastic scintillators is the poor energy resolution. This is a manageable disadvantage since the two photons of interest differ in energy by a significant amount.

The focus of this project is the design of a PALS system using a unique hardware combination to achieve a timing resolution near or below 200 ps. The design includes a custom written C/C++ software application to interface with the hardware and process the data efficiently. The system design will allow for high counting rates and timing resolution equivalent to state of the art systems using low resolution plastic detectors and efficient computing techniques.
Chapter 2: Theory

PALS is based on the measured time between the birth and annihilation of a positron in a material sample. Positrons are introduced to the sample using radioactive materials, specifically $\beta^+$ emitters. The positron lifetime is experimentally defined as the time difference between detection of the decay gamma ray of the radioactive nuclide and the detection of an annihilation photon. The traditional isotope used in PALS is Na-22, the Decay Scheme for Na-22 is shown in Figure 1. The 1.2746 MeV excited state of Ne-22 is populated following 99.94% of decays and has a half-life of 3.7 ps, which allows it to be treated as the birth time of the positron [11]. The lifetime of the positron is dependent on the local atomic structure near its origin. The energetic positron loses energy in the sample due scattering and electronic interactions. Once thermalized, the local vacancies defects will act as a potential well, trapping the positron and increasing its lifetime [4, 6-8]. Positron lifetimes are on the order of 1 ns, thus very precise timing is required to properly quantify the vacancy and dislocation defects.
The positron lifetime decay spectrum of a sample with \( n \) defect types at time \( t \) is given by

\[
D(t) = \sum_{i}^{n+1} I_i e^{-\frac{t}{\tau_i}}, \tag{1}
\]

Where \( I_i \) and \( \tau_i \) are respectively the intensity and lifetime component of the defect type. The function \( D(t) \) represents the probability of annihilation for a positron in the sample at time \( t \). The absolute value of the derivative of equation 1 is the positron lifetime spectrum,

\[
N(t) = \sum_{i}^{n+1} \frac{I_i}{\tau_i} e^{-\frac{t}{\tau_i}}. \tag{2}
\]

The value of \( \tau_1 \) represents the lifetime of a positron in a defect-free sample of the material being tested. Typical experiments can reliably measure three components in a spectrum \( (n = 2) \), due to timing resolution and sample configuration background. The positron lifetime spectrum measured experimentally is superimposed onto the resolution function of the system, which is a Gaussian for plastic scintillators. Figure 2 shows the
measured Positron lifetime spectrum of both an as-grown and plastically deformed Czochralski-grown Silicon Sample, which highlights the lifetime components that can be observed [19]. Several software packages have been developed to determine the lifetime components of a sample given the experimentally measured spectrum, a detailed discussion of which can be found in [5-7].

![Figure 2:Measured positron lifetime spectroscopy for both an as-grown and plastically deformed Czochralski-grown Silicon Sample [27]. This spectra is used only to show the form of a typical positron lifetime spectrum.](image-url)
The time resolution of an experimental system is characterized by its full width at half maximum for the detection of coincident events. This value is the width of the Gaussian peak at half of its amplitude and is related to the standard deviation, \( \sigma \), by

\[
\text{FWHM} = 2.355\sigma. \tag{3}
\]

Several techniques exist to determine the timing information for radiation detection events, each having its own advantages and disadvantages. Sources of error common to all time pick-off methods are divided into two categories: time-jitter and amplitude walk. Time jitter is associated with uncertainty sources that affect signals with constant pulse amplitude while amplitude walk is associated uncertainty sources arising from the range of pulse amplitudes [28]. Time jitter is primarily caused by electronic noise in the system, but amplitude walk can be caused by a large number of factors.

The most common method is leading edge triggering, in which timing information is based on the pulse signal crossing a static threshold. This method is effective if the dynamic range of pulse amplitudes is small, but will be very inaccurate if the dynamic range of pulse amplitudes is wide. This effect can be minimized by setting the static threshold low, typically 10%-20% of the expected amplitude. The effects of time jitter are reduced when the slope of the pulse is large [28-29]. The ideal threshold setting for a leading edge for a given system depends on the balancing of the effects of time jitter and amplitude walk. The uncertainty introduced by the amplitude walk of leading edge triggering can be removed with a timing technique called fast zero-crossing timing. Crossover timing requires the input signal be bipolar. The time at which the input signal crosses from positive to negative and is independent of the pulse amplitude. This method
is superior to leading edge timing if the signal shape and rise time are constant. The effect of time jitter on fast zero-crossing timing is larger than that of leading edge timing [knoll]. Another method exists that is both independent of pulse amplitude and suffers little from time jitter effects is constant fraction discrimination (CFD). The timing information is extracted when the pulse reaches a preset percentage of its maximum. This method, like fast zero-crossing, requires the signal rise time be constant. CFD is the superior method for coincidence timing, due to the lower effect of jitter and amplitude independence. A detailed discussion of the digital implementation of the digital CFD is given in chapter 5.
Chapter 3: Analog vs. Digital Systems

A traditional analog PALS system is shown in Figure 3. The system is comprised of two detectors, one that detects a 1.237 MeV start pulse and another that detects the annihilation photon stop signal. The pulses from each of the detectors are passed to a CFD unit, which generates a logic pulse when the pulse amplitude reaches 20% of its maximum value. These CFD units also act to discriminate the pulse so that it only accepts pulses within a certain amplitude window, allowing for start or stop pulses to be set individually. The delay boxes are used to compensate for transmission delay that would affect the timing of the two pulses. The time-to-amplitude converter then generates a signal with an amplitude that is proportional to the time difference between the start and stop signals. This signal is then passed to the multichannel analyzer with digitizes the pulse and builds a histogram of positron lifetimes. The fast coincidence ensures that the logic pulse is started in the TAC only if a start and stop event are detected within a certain time window (typically ~100 ns).
A digital PALS system is shown in Figure 4. The digital system requires far fewer pieces of equipment, as the pulse is converted to a digital value directly from the photo multiplier tube (PMT). The digital values can then either be sent to a PC for processing or processed with hardware with an FPGA. If an FPGA is used to process the digital signal, the complexity of the processing algorithm is limited. The digital waveform passed to the PC are processed with digital algorithms limited only by the processing speed of the PC used. Typical systems use a combination of onboard FPGA processing coupled with the
PC. The digital setup lacks designated start and stop detectors, due to the ability to distinguish and filter start and stop events in each detector. The start and stop pulses from an individual event must still occur in separate detectors.

![Figure 4: General schematic of a digital PALS system.](image)

The use of a digital system has both advantages and disadvantages over traditional analog systems. The advantages and disadvantages of digital systems, summarized below, are described in detail in refs. [30-32]. The advantages of digital systems include:

- Digital systems allow for dead-timeless data acquisition
- Energy, timing, and pulse analysis measurements are done on a single device
- Reduced size, cabling, power consumption and cost
- Highly flexible filtering
- Better pile-up, ballistic deficit and baseline fluctuation effect corrections
• Multichannel trigger synchronization
• Faster and automatic tuning and calibration

The disadvantages of digital systems are the shallow learning curve of their use, as well as the extensive knowledge required for customization of the FPGA logic board. The FPGA customization will nearly always require substantial technical support from the vendor. The use of software based filters requires knowledge of computer programming, interfacing software and hardware, as well as extended knowledge of digital filtering techniques, and for this reason it may take a longer time to learn how to effectively build an experimental configuration [30-32]. Digital systems are also disadvantaged because the sampling rate and resolution cannot be improved without replacing the unit entirely; with an analog system it may be possible to replace a specific device in the system to improve performance.

Digital systems will use either a digital oscilloscope or digitizer to record the pulses produced in the PMT. Digital oscilloscopes are just a simple waveform digitizer; converting the analog signal to a digital ADC value for a number of samples following a trigger event. The recorded event is then moved into a memory buffer that is available for readout. Digitizers share this same basic function, but have a few additional features. Digitizers have no dead-time between triggers, so long as the data read rate of the memory buffers is faster than the write rate. Digitizers also allow for trigger syncing between multiple units. This trigger syncing makes it possible for thousands of channels to share a common trigger. Digitizers are typically equipped with higher bandwidth readout links, allowing for high counting rates. Digitizers also allow for online
information processing with hardware based logic filters (FPGA or otherwise.) The online processing of the digital signal allows for significant reductions in data throughput to the PC. This reduced data throughput allows much higher counting rates and efficient data storage.

Digital coincidence detection systems using a digitizer allows for software based event correlation. With a software approach to event correlation the pulse information is preserved, as all of the digital information is passed to the PC for processing. This is an extremely advantageous, as analysis of the collected information can be processed offline at another time. The preservation of data also allows for direct comparison of digital processing algorithms. When used with ultrafast digitizers, the data throughput can be enormous (over 4 GB/s for a 2 GS/s 10 bit digitizer at maximum readout.) High data throughputs require efficient FPGA trigger filtering methods to reduce the amount of data being transferred to manageable levels.
Chapter 4: Experimental Approach

Design Objectives

The primary focus of the design is to achieve the lowest possible timing resolution for coincident events with a reasonable capability for counting rate and minimal data output. The system was designed around two fast plastic scintillation detectors attached to low-noise photomultiplier tubes. The electronic signal produced from the PMTs is fed into an ultrafast digitizer, which produces a digital waveform of each event. This digital waveform is passed to a PC and processed with custom written software. The development of the positron lifetime spectroscopy system includes equipment assembly and calibration, timing measurements and software development. The performance of the system is checked with the detection of two gamma rays that were birthed simultaneously in the same decay, i.e., from Co-60.

Calibration

In order to implement the proper discrimination levels for the system and measure the timing performance of the systems, several button sized radiation sources spectra were used. The first set of measurements were done to determine the most effective high voltage settings for each detector. Since the detector used is primarily composed of hydrogen and carbon, there will be no photopeak detected for any radiation source used.
This is due to the relatively large Compton scattering cross section. Instead, the Compton edge of the gamma ray spectra can be used to distinguish the gamma rays. The ideal high voltage setting would allow the Compton edge of the decay gamma ray to be clearly distinguishable from that of the annihilation photons.

System Time Resolution

The time resolution that characterizes the PALS system is derived from the response of the detectors and digitizers to two events which are known to have been produced at the same time. The ideal candidate for this kind of test is Co-60, which, following radioactive decay to 60Ni, emits a 1.17 MeV and 1.33 MeV gamma ray. The energy of the gamma rays are near that of the 1.1237 MeV gamma ray used as the starting signal of a positron decay event from Na-22. A button sized Co-60 source was used in the measurement. The source was calibrated on April 1, 2006 with an activity of 384.4 kBq. With a half-life of 5.27 years, the Co-60 source strength was 153 kBq.

System Instrumentation

The PALS system, shown in Figure 5, is composed of two ORTEC 905-21 fast plastic scintillation detectors, a mounting system designed by ORTEC, a CAEN High Voltage Supply, a CAEN N6751 digitizer and a PC for data processing/readout. The two ORTEC 905-21 fast plastic detector was used for event detection. A dual channel data
acquisition system, including a digitizer and high voltage power supply from CAEN S.p.A., is used for data acquisition. The system is housed in a two slot Nuclear Instrumentation Module crate and is capable of measuring data at a rate of 2 GS/s in two channels simultaneously. A description of each component of the experimental system is given below.

Figure 5: PALS system overview

1.) ORTEC 905-21 Fast Plastic Scintillator
This fast plastic scintillation device contains 12.9 cm$^3$ Bicorn BC-418 plastic and has a truncated-cone shape. BC-418 is a premium plastic with a polyvinyltoulene base, a density of 1.032 g/cm$^3$ and a refractive index of 1.58. The scintillation properties of the detector are shown in Table 1. The light output of the material is compared to anthracene, a common plastic scintillator material. The PMT and PMT base came packaged with the detector from ORTEC, and include an 8850 PMT (or equivalent) and a 265A PMT base. The 265A PMT base can accept a HV of up to -3000 V at 2 mA. The PMT assembly is shielded with an ORTEC Model 218 magnetic shield. A picture of the detector assemblies are shown in Figure 6.

<table>
<thead>
<tr>
<th>Light Output (% Anthracene)</th>
<th>67</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rise Time (ns)</td>
<td>0.5</td>
</tr>
<tr>
<td>Decay Time (ns)</td>
<td>1.4</td>
</tr>
<tr>
<td>Pulse Width, FWHM (ns)</td>
<td>1.2</td>
</tr>
<tr>
<td>Wavelength of Max emission (nm)</td>
<td>391</td>
</tr>
<tr>
<td>Bulk Light Attenuation (cm)</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 1: BC-418 fast plastic scintillation properties.
2.) CAEN N1471 High Voltage Power Supply

A high voltage power supply unit capable of producing a maximum voltage of 5500 V with an accuracy of 0.02% ± 2V. The unit has very high long term voltage stability and very low voltage ripple. The unit is programmable via front-facing hardware controls. The polarity and be changed from positive or negative with a hardware modification.

3.) CAEN N6751 Digitizer

The N6751 digitizer serves to rapidly digitize the PMT output and control event triggering based on a simple ADC count threshold. The digitizer
samples the PMT output voltage of each detector assembly at a rate of 2 GS/s.
The digitizer includes firmware controlled FPGA that is can be interfaced
with software for filtering. If the ADC value in either detector surpasses the
programmed threshold a user programmed number of samples are saved in
each detector are saved to a memory buffer for readout. The samples saved for
readout contain the raw event information. Additionally, a trigger timestamp
is generated at the arrival of each trigger. The HV power supply and digitizer
attached in the 2-slot NIM crate can be seen in Figure 7.
Figure 7: N6751 digitizer and N1471 high voltage power supply connected to a model N147X 2-slot NIM crate, all supplied by CAEN S.p.A.

4.) Personal Computer

A PC running the system software is used as the brain of the data acquisition system. The PC polls the digitizer for events saved in a memory
buffer and reads out any stored data. The data is then handled by the custom software according to the users wishes. The rate at which events can be collected is bottlenecked by the host PC running the application, so a faster PC will allow for faster event collection. A modern desktop PC will, however, be able to process events fast enough that no bottlenecking will occur.
Chapter 5: Software

Custom software was written for the DAQ system. The software uses four external libraries; three C API libraries from CAEN S.p.A used for interfacing with the digitizer and one C++ library from the ALGlib Project [33] for event analysis. The libraries are only used for hardware interfacing and building a cubic spline interpolant respectively. The software also is used to log activity of the system, including information about each data collection run. When data is collected the DAQ software stores the raw waveform of the events along with timing information, pulse amplitude and the integrated charge of the pulse. The raw waveforms can be collected for each channel individually or for only coincident events. The flexibility in data storage will ease possible system recalibrations if necessary. The software determines two events to be coincident if a pulse exits from both detectors in the stored samples following an event trigger in either detector.

The IO events in the software were found to be an important factor in maximizing count rates, so efforts to minimize the amount of data written were explored. If the event is to be written to an output file, the event data is parsed to only output the detected waveform along with several leading and trailing zeros. In the case of coincident event, the data is written to ensure the first sample written out is identical for each wave. The parsed data allows for much smaller data files, along with greatly improved performance.
The source code for the software is given in App A. Upon running the application, the user is prompted either view the default digitizer setting, change the digitizer settings, add information to the log file, process an existing data file, enter the data collection menu or exit the application. The data collection menu prompts the user to start/stop collection, open a file for data storage, save and close an open data collection file, toggle coincident event collection / all event collection or exit the application.

When the user selects to process an existing data file, after the file is selected, the data is processed one event at a time in the following manner:

1.) The events are checked to see if one of each start and stop events are present. If they are not the program reads the next event. This continues until the initial criteria is established.

2.) The program check to see which event contains the starting pulse, event 1, and which contains the stopping pulse, event 2.

3.) Each timing of each event is determined with the Digital CFD filter.

4.) The positron lifetime is calculated as the timing difference between event 1 and event 2.

A digital CFD is done by calculating for each sampled value, \( S_i \),

\[
 f_i = F \cdot S_{i+d} - S_i , \tag{4}
\]

Where \( F \) is the constant fraction value and \( d \) is the number of samples the event samples are delayed. Figure 8 shows what the effect of the CFD filter. The start time of the event is the time when the filtered signal crosses the x-axis. Since it is unlikely that this value
will occur on a sampled data point, interpolation methods are used to find the value. When using interpolation methods one must consider the computational cost; high cost methods can serve to reduce the counting rate in the system when the calculation is done online. The effectiveness of different interpolation methods when used with CFD have previously been investigated and cubic spline interpolation was found to be the accurate [18-20]. When calculating the zero-crossing value, the software builds a cubic spline interpolation, with code developed by ALGlib, to calculate the event start times. The algorithm identifies the two samples in the filtered wave which are above and below the x-axis, and starts by calculating the value at their midpoint. Depending on whether this value is above or below the x-axis, the maximum or minimum of the zero-crossing interval is changed to the midpoint tested. The algorithm stops searching when a zero is found (with a tolerance of 10e-4.) Once both event start times are found the lifetime is calculated and written to an output file. Future software revisions may allow for the data to be binned into a histogram, which would reduce the data output further.
Figure 8: Representation of a digital CFD filter on the output of a PMT following a radiation event. The dots in the lower figure represent the digital samples, highlighting the fact the true event timing information lies between the sampled data.
Chapter 6: System Calibration

The first step in the system calibration was to identify the proper high voltage input for each of the detectors. The proper high voltage setting would place the 1.237 MeV event pulses to be contained in the upper 80-90% of the pulse height spectrum, allowing for the highest possible separation between the start and stop pulse events. Due to the 10-bit resolution of the detector, the thresholds were based on the integrated charge of the detected events. The amount of charge collected is expected to be proportional to the pulse height, but allows for greater flexibility in tuning due to its larger range. Figure 9 shows the pulse height VS integrated charge for each both detectors used. This data was collected from a Co-60 sample, which generated pulses with amplitudes across the entire 10-bit resolution. The linearity of the pulse-height and integrated charge can be clearly observed, allowing for threshold settings to be established on the basis of collected charge. It is also evident that the amplitude walk increases with increasing photon energy. A histogram of the integrated charge for the events is shown in Figure 10. While it is impossible to distinguish the decays gammas, the presence of both can be observed by the increase of counts near 7000 LSB. The measurements in Figs X were taken with input voltages of 1815 V and 1940 V for channels 1 and 3 respectively. This value was found to create the largest separation between the high energy start pulse and the lower energy stop pulse.
Figure 9: The pulse amplitude versus integrated charge of data collected from a Co-60 button source. The plot shows the amplitude of a pulse is proportional to the total charge collected, with the exception of very low energy events where the resolution limitations of the detector are much more significant. The plot also highlights the increasing amplitude walk for high energy events.

Figure 10: Integrated charge spectra of the events in Figure 8. The two decay gamma rays from the decay of Co-60 cannot be distinguished, but the superposition of their Compton continuum is observable. No photopeaks are present due to the detector composition.
Figure 11 shows the raw measured spectra of a Na-22 sample, while Figure 12 shows the measured coincident spectrum for a Na-22 sample; each with the same high voltage settings mentioned previously. Due to the detectors hydrocarbon composition, no photopeaks are detected. The Compton edge for the both the 1.237 MeV decay peak and annihilation photons are clearly observed.

The magnitude of the Compton edge is of the annihilation photons noticeably larger than that of the 1.237 MeV in the both the raw data and coincident spectra, which is can be explained with simple physical reason. One of these reason is the higher detection efficiency at the lower photon energy. Another reason is that nearly a third of the collected events with pulses near the annihilation photon’s Compton edge will be due to the 1.237 MeV decay gamma, since the annihilation events will occur in the Compton continuum of the 1.237 MeV decay gamma. Additionally, in the case of the raw spectra, the number of annihilation photons is twice that of the decay gammas. In the case of the coincident spectra, a number of events where both annihilation photons are detected will increase the number of counts. The Integrated charge thresholds to be used for determining whether a detected event will be a starting or stopping event, which are determined from the Compton edge of the two photon energies of interest, are listed in Table 2. The effect of background events on the coincident is nearly non-existent, due to the extremely low counting rates.
Figure 11: Single event triggers from a Na-22 source highlighting the 1.237 MeV decay gamma 511 keV annihilation gamma Compton edges.

Figure 12: Coincident event spectra from the same Na-22 source used in Figure 10, showing all the same features.
Table 2: Detector filter threshold for coincident event detection.

<table>
<thead>
<tr>
<th>Channel</th>
<th>Start event Range (LSB)</th>
<th>Stop event Range (LSB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7000-10000</td>
<td>500-2000</td>
</tr>
<tr>
<td>3</td>
<td>7000-10000</td>
<td>500-2000</td>
</tr>
</tbody>
</table>
Chapter 7: Timing Measurements

The timing measurements were conducted to obtain the resolution function to be used in determining the positron lifetimes. The measurement was done with using the same high voltage settings mentioned above for the Co-60 spectra. The coincident event filters were set to accept events from both detectors if the integrated charge was between 6000 and 10000 LSB. The lower portion of the Compton continuum was excluded to minimize the effect of effect of photons that are backscattered into the detectors. Despite the inability to distinguish the decay gammas from one another coincident events were considered to be from the same decay event. This is a reasonable assumption because the counting rates in both detectors were on average 400 counts/s, thus the chances of unrelated events being detected in the same 20 ns interval are negligible. When the timing difference between should be very near to 0, the shape of the measured time differences will yield the resolution function. The FHWM of this spectrum will define the performance of the system.

The optimal settings for fraction of pulse height to be used and the delay to be used depend on the shape of the event waveform. To find the optimum values a large number of coincident events was collected and each of the parameters was varied and the FWHM of each setting was calculated. Figure 13 shows the results of the optimization, which varied delay from 4 to 7 samples (2-3.5 ns) and the fraction of the pulse amplitude from 15%-40%. The numerical values are also listed in Table 3. The ideal settings were found to be a fraction of 40% of the amplitude and a delay of 4 samples (2 ns.)
fraction of the amplitude is higher than that of similar systems, but this may be due to the fact the delay setting is less than the waveform rise time (3-3.5 ns). This lower delay setting would mean the zero crossing of the digital CFD filter actually occurs at a fraction of the real pulse waveform that is lower than the actual set value.

Figure 13: Optimization results after varying the amplitude fraction and signal delay used in the CFD filter. The best performance was found with a signal delay of 4 samples (2 ns) and a constant fraction of 40% of the peak amplitude.
Table 3: FWHM values for the different delay and fraction settings applied to coincident gamma rays measured from a Co-60 button source.

<table>
<thead>
<tr>
<th>Amplitude Fraction</th>
<th>Delay (Samples)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>0.15</td>
<td>0.290</td>
</tr>
<tr>
<td>0.2</td>
<td>0.252</td>
</tr>
<tr>
<td>0.25</td>
<td>0.226</td>
</tr>
<tr>
<td>0.3</td>
<td>0.214</td>
</tr>
<tr>
<td>0.35</td>
<td>0.210</td>
</tr>
<tr>
<td>0.4</td>
<td>0.209</td>
</tr>
<tr>
<td>0.45</td>
<td>0.211</td>
</tr>
</tbody>
</table>

A histogram of the measured time difference between coincident events with the ideal settings is shown in Figure 14. When a Gaussian fit is calculated for the measured results. The peak is found to be centered at \( x = 2.139 \pm 0.004 \) ns with a FWHM of \( 208.5 \pm 1.1 \) ps, thus the timing resolution of the system is found to be 209 ps. This result is comparable to PALS systems utilizing similar sampling frequencies and expensive BaF3 scintillation detectors.
Figure 14: Histogram of measured time difference of between the arrivals of two coincident gamma rays, which is used to define the resolution function of the PALS system.
Chapter 8: Conclusions

A Positron Annihilation Lifetime Spectroscopy system utilizing a digitizer has been constructed at NARS lab at the Ohio State University for the measurement of material defects. The use of efficient digital algorithms and data storage techniques allow for the system to operate with high counting rates. The timing resolution of the system is found to be 209 ps, very near that of state of the art systems. For a current generation PC, the counting rate will be limited by the trigger rate of unrelated events in the coincidence window. This limitation depends on the source-sample geometry as well as the sample composition. The system also allows for easy modification of the digitizer parameters should experimental conditions change.
References


Appendix A: NARS_PLS.cpp

/#***********************************************************************#
#**                                                                         **#
#**                                                                         **#
#**                                                                         **#
#**                                                                         **#
#**                                                                         **#
#**************************************************************************#
#**                                                                         **#
#**************************************************************************#
#include <CAENDigitizer.h>
#include "ConsoleFunctions.h"
#include <string>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
// alglib includes
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <sstream>
// disable some irrelevant warnings
#if (AE_COMPILER==AE_MSVC)
#pragma warning(disable:4100)
#pragma warning(disable:4127)
#pragma warning(disable:4702)
#pragma warning(disable:4996)
#pragma warning(disable:4244)
#endif
#include "alglibmisc.h"
#include "alglibinternal.h"
#include "linalg.h"
#include "statistics.h"
```cpp
#include "dataanalysis.h"
#include "specialfunctions.h"
#include "solutions.h"
#include "optimization.h"
#include "diffequations.h"
#include "fasttransforms.h"
#include "integration.h"
#include "interpolation.h"

// analysis headers
#include "analysis.h"

#include "Functions.h"
#define pdelay 4
#define pfrac 0.4

using namespace std;

/* ###########################################################################
 * Functions
 * ########################################################################### */
void analysis();

/* ###########################################################################
 * MAIN
 * ########################################################################### */
int main(int argc, char *argv[])
{
    CAEN_DGTZ_ErrorCode err; // output of most CAEN functions, used to determine errors
    CAEN_DGTZ_EventInfo_t EventInfo; // Event header data
    CAEN_DGTZ_UINT16_EVENT_t *Events=NULL; // Events data buffer
    CAEN_DGTZ_BoardInfo_t BoardInfo; // Board Info buffer

    // digitizer Parameter defaults
    PLS_Parameters Params;
    Params.AcqMode = CAEN_DGTZ_SW_CONTROLLED;
    Params.ChannelMask = 0xA; // enable 2GS/s sampling
    Params.MaxBLT = 1024; // number of events per buffer
```
Params.Polarity = CAEN_DGTZ_PulsePolarityNegative; // signal has negative polarity
Params.PostTriggerSize = 10; // 10% of sample come after trigger
Params.RecordLength = 1024; // number of samples per event
Params.SelfTrigger = CAEN_DGTZ_TRGMODE_ACQ_ONLY; //
Params.Threshold = 1022; // threshold for trigger (1023 = no signal)
Params.TriggerMode = CAEN_DGTZ_TRGMODE_ACQ_ONLY;
Params.DCOffset = 0;

string filename,str,str1,str2; // Output file buffer for writing wave info
bool eventactive=false;
time_t start,stop,current,old; // time variables used to record run statistics
int handle,key,i,j; // board id, 3 general use variables
int totalevents=0,writeevents=0; // run statistic variables
int configerr; // used for config function call
int readrate = 0,writerate= 0 ,eventrate= 0; // runtime statistic variables
float tdiff; // used to calculate runtime statistics
char *buffer=NULL; // readout buffer
int wavesum1, wavesum2; // charge integration variables
uint32_t AllocatedSize,BufferSize,NumEvents; // event variables
bool Quit = false; // used to control end of program
bool OK = false; //
// Configure Bools
bool Configure = true;

ofstream digconfig ; // log file Ostream
digconfig.open("Experimental_config.log", std::ios::out | std::ios::app); // open and append

// Data Processing bools
bool ProcessData = false;
bool Output_OK = false;
// Aquisition bools
bool Acq = false;
bool expparams = false;
bool Write_OK = false;
bool wave1 = false;
bool wave2 = false;
bool Coincidence = true;
// event pointer
char *EventPointer = NULL;
uint16_t event1[100],event2[100];
int wave1beg=0,wave2beg=0,wave1end=0,wave2end=0; // used to store only relevant data for writing
ofstream file1; // ch1 output file
ofstream file2; // ch3 output file
int wavefirst; // used to start saved wave data at same point
PrintConfigMenu();

while(!Quit)
{
    while(Configure)
    {
        if(_kbhit())
        {
            key = _getch();

            switch(key)
            {

            case 'v':  PrintParams(Params);
                        break;
            case 'e':  EditParams(Params);
                        break;
            case 'a':  Acq = true;
                        Configure = false;
                        break;
            case 'l':  expparams=true;
                        break;
            case 'P':  analysis(); break;
            case 'q':  Quit = true;
                        Configure = false;
                        break;
            default: break;
            }

        }
    }

    // Connect to Board
    err = CAEN_DGTZ_OpenDigitizer(CAEN_DGTZ_USB,0,0,0,&handle);
    if(err)
    { cout << "Cannot open digitizer: error code " << err << "n"; return -1;)

    // program digitizer
configerr = ConfigureDigitizer(handle, Params);
if(configerr != 0)
{cout<< "config error " << configerr << "\n"; return configerr;}

// allocate memory for events
CAEN_DGTZ_AllocateEvent(handle, (void**)Events);
err = CAEN_DGTZ_MallocReadoutBuffer(handle, &buffer, &AllocatedSize);
if(err)
{ cout << "Cannot allocate buffer" << "\n"; return -1;}

LogEntry(digconfig,handle);
if(expparams)
ExpParams(digconfig);

PrintRunMenu();

while(Acq)
{
    if(_kbhit())
    {
        key = _getch();

        switch(key)
        {
            case 's': err =CAEN_DGTZ_SWStartAcquisition(handle);
                       start = clock();
                       old = start;
                       if(err)
                       { cout << "Cannot start aquisition" << "\n"; return -1;}
                       OK = true;
                       break;

            case 'S':err = CAEN_DGTZ_SWStopAcquisition(handle);
                       stop = clock();
                       if(err)
                       { cout << "Cannot stop aquisition" << "\n"; return -1;}
                       OK = false;
                       cout << "Run lasted " << (float(stop) - float(start)) / (60*CLOCKS_PER_SEC) << " minutes" << "\n";
                       cout << "Run Measured " << totalevents << " total events." << "\n";
        }
    }
}
cout << writeevents << " total events written" << "n";
writeevents=0;
totalevents = 0;
break;

case 'w' : cout << "Please name the output file" << "n";
cin >> filename;
str = filename + ".Ch1.txt";
file1.open(str.c_str());
str2 = filename + ".Ch3.txt";
file2.open(str2.c_str());
cout << "n" << "Events saved to " << str << " and " << str2 << "n";
Write_OK = true;
break;

case 'T' : err =CAEN_DGTZ_SendSWtrigger(handle);
if(err)
{ cout << "Cannot send SW trigger" << "n"; return -1;}
break;

case 'W' : file1.close();
file2.close();
cout << "Event output files closed" << "n";
break;

case 'C' : if(Coincidence)
{ Coincidence = false; cout << "Saving All events\n"; }
else
{ Coincidence = true; cout << "Only Saving Coincident events\n"; }
break;

case 'q' : Quit = true;
Acq = false;
break;

default: break;
if(!OK)
{
    Sleep(10);
    continue;
}
if(Write_OK)
{
    current = clock();
    tdiff = (float(current) - float(old)) / CLOCKS_PER_SEC;
    if(tdiff > 10)
    {
        PrintRunMenu();
        cout << "event rate:\t" << float(eventrate)/tdiff << " events/s" << "n";
        cout << "data write rate:\t" << float(writerate)/(tdiff) << " events/s" << "n";
        cout << "Total events written:\t" << writeevents << " events\n" << "n;
        readrate = 0;
        writerate = 0;
        eventrate = 0;
        old = current;
    }
    err = CAEN_DGTZ_ReadData(handle,
CAEN_DGTZ_SLAVE_TERMINATED_READOUT_MBLT, buffer, &BufferSize);
    if(err)
    {
        cout << "Cannot read data" << "n"; return -1;
    }
    err = CAEN_DGTZ_GetNumEvents(handle, buffer, BufferSize, &NumEvents);
    if(err)
    {
        cout << "Cannot get num of events" << "n"; return -1;
    }
    totalevents += NumEvents;
    eventrate += NumEvents;
    for( i = 0; i < NumEvents ; i++)
    {
        memset(event1,0,sizeof(event1));
        memset(event2,0,sizeof(event2));
    }
}
err = CAEN_DGTZ_GetEventInfo(handle, buffer, BufferSize, i, &EventInfo, &EventPointer); // read single event
if (err)
    { cout << "Cannot get event info" << "\n"; return -1; }

//readrate += sizeof(EventInfo);

err = CAEN_DGTZ_DecodeEvent(handle, EventPointer, (void**)&Events);
if (err)
    { cout << "Cannot decode event" << "\n"; return -1; }

if (Events->ChSize[1] > 0)
{
    wavesum1 = 0;
    wavesum2 = 0;
    wave1beg = 1;
    wave1end = Events->ChSize[1];
    wave2beg = 1;
    wave2end = Events->ChSize[1];

    for (j = 1; j < Events->ChSize[1]; j++)
    {
        wavesum1 += 1023 - Events->DataChannel[1][j];

        if (Events->DataChannel[1][j-1] == 1023 && Events->DataChannel[1][j] < 1023)
            {wave1beg = j;}

        if (Events->DataChannel[1][j] == 1023 && Events->DataChannel[1][j-1] < 1023)
            {wave1end = j; wave1=true; break;}
    }
}
for(j = 1; j < Events->ChSize[3]; j++)
{
    wavesum2 += 1023 - Events->DataChannel[3][j];
    if(Events->DataChannel[3][j-1] == 1023 &&
        Events->DataChannel[3][j] < 1023)
        {wave2beg = j;}
    if(Events->DataChannel[3][j-1] < 1023 && Events->DataChannel[3][j] == 1023)
        {wave2end = j;wave2=true; break;}
}
if (wave1 && !Coincedence)
{
    file1 << EventInfo.TriggerTimeTag<< 't' << wavesum1 << 't';
    writerate += 1;
    writeevents +=1;
    for(j = wave1beg-10; j < wave1end+10; j++)
        file1 << 1023 - Events->DataChannel[1][j] << 't';
    file1 << 'n';
}
if (wave2 && !Coincedence)
{
    file2  << EventInfo.TriggerTimeTag<< 't' << wavesum2 << 't';
    for(j = wave2beg-10; j < wave2end+10; j++)
        file2 << 1023 - Events->DataChannel[3][j] << 't';
    file2 << 'n';
}
writeevents += 1;
writerate += 1;

}

if(Coincidence && wave1 && wave2)
{

if(wave1beg <= wave2beg)
    wavefirst = wave1beg;
else
    wavefirst = wave2beg;

EventInfo.TriggerTimeTag << '\t' << wavesum1 << '\t';
file1 << "\t" <<
writerate += 1;
writeevents +=1;

j++)
  >DataChannel[1][j] << '\t';

EventInfo.TriggerTimeTag << '\t' << wavesum2 << '\t';
for(j = wavefirst - 10; j < wave1end + 10;
    file1 << 1023 - Events-

file1 << "\n";

for(j = wavefirst-10; j < wave2end+10;
    file2 << 1023 - Events-

file2 << "\n";

}
wave1= false;
wave2= false;

}
err = CAEN_DGTZ_SWStopAcquisition(handle);
if (err)
    { cout << "Cannot stop acquisition" << "n"; return -1; }
err = CAEN_DGTZ_FreeReadoutBuffer(&buffer);
if (err)
    { cout << "Cannot free readout buffer" << "n"; return -1; }
err = CAEN_DGTZ_CloseDigitizer(handle);
if (err)
    { cout << "Cannot close digitizer" << "n"; return -1; } 

digconfig.close(); 
return 0;

} 

void analysis()
{
    double datum; 
    ifstream chan1, chan3; 
    ofstream out1; 
    int count = 0; 
    string filename, chan1wave, chan3wave, str, str1, str2; 
    int wave1dat[1100], wave2dat[1100];
    memset(wave1dat, 0, sizeof(wave1dat));
    memset(wave2dat, 0, sizeof(wave1dat));
    long long int readout;
    PLSEvent eventq;
    long long timetag1, timetag2;
    int sum1 = 0, sum2 = 0;

cout << "Please name the input file" << "n";
    cin >> filename;
    str = filename + ",Ch1.txt";
    chan1.open(str.c_str());
    str1 = filename + ",Ch3.txt";
chan3.open(str1.c_str());
str2 = filename + "_Lifetimes.txt";
out1.open(str2.c_str());

while(getline(chan1,chan1wave) && getline(chan3,chan3wave)){

    stringstream ss1(chan1wave);
    stringstream ss2(chan3wave);

    ss1 >> timetag1 >> sum1;
    ss2 >> timetag2 >> sum2;

    if(timetag1 != timetag2)
    {
        cout << "ERROR: Event timetags are not the same" << endl;
        break;
    }

    count = 10;
    while(ss1 >> readout)
    {
        wave1dat[count] = readout;
        count++;
    }

    count = 10;
    while(ss2 >> readout)
    {
        wave2dat[count] = readout;
        count++;
    }
}
if( sum1 > 7000 && sum1 < 10000 && sum2 > 500 && sum2 < 2000)
{
    eventq.LoadEvent(wave1dat, wave2dat, pdelay, pfrac, sizeof(wave1dat)/sizeof(int));
    datum = eventq.DTCubic();
    out1 << datum << "\n";
    eventq.ResetData();
}

if( sum2 > 7000 && sum2 < 10000 && sum1 > 500 && sum1 < 2000)
{
    eventq.LoadEvent(wave2dat, wave1dat, pdelay, pfrac, sizeof(wave1dat)/sizeof(int));
    datum = eventq.DTCubic();
    out1 << datum << "\n";
    eventq.ResetData();
}

memset(wave1dat, 0, sizeof(wave1dat));
memset(wave2dat, 0, sizeof(wave2dat));

chan1.close();
chan3.close();
Appendix B: Analysis.cpp

```cpp
#include "analysis.h"
#include <iostream>
#include <stdlib.h>
#include <new>

void PLSEvent::LoadEvent( int * event1 , int * event2 , int chan_delay , double const_frac, const int size)
{
    dtime = new double [size - chan_delay];

    for(int i = 0; i < size - chan_delay; i++)
        dtime[i] = i*0.5;

    time.setcontent(size-chan_delay, dtime);

    iwave1 = event1;

    iwave2 = event2;

    Delay = chan_delay;
    CF = const_frac;
    EventSize = size;

    delete[] dtime;
}

void PLSEvent::CRRCSmooth()
{
```

51
for( int i = 0; i < (EventSize - 3); i++)
{
}
}

void PLSEvent::BuildCFD()
{
    Interval1Min = 0;
    Interval1Max = 0.5*(EventSize - Delay);
    Interval2Min = 0;
    Interval2Max = 0.5*(EventSize - Delay);
    dwave1 = new double [EventSize-Delay];
    dwave2 = new double [EventSize - Delay];
    dwave1[0] = CF*iwave1[Delay] - iwave1[0];
    for(int i = 1; i < (EventSize - Delay);i++)
    {
        dwave1[i] = CF*iwave1[i+Delay] - iwave1[i];
        if( dwave1[i] < 0 && dwave1[i-1] > 0)
        {
            Interval1Min = 0.5*(i-1);
            Interval1Max = 0.5*i;
        }
    }
    dwave2[0] = CF*iwave2[Delay] - iwave2[0];
    for(int i = 1; i < (EventSize - Delay);i++)
    {
        dwave2[i] = CF*iwave2[i+Delay] - iwave2[i];
        if( dwave2[i] < 0 && dwave2[i-1] > 0)
        {
            Interval2Min = 0.5*(i-1);
            Interval2Max = 0.5*i;
        }
    }
}

wave1.setcontent(EventSize - Delay, dwave1);
wave2.setcontent(EventSize - Delay, dwave2);

delete[] dwave1;
delete[] dwave2;

}

void PLSEvent::ResetData()
{
    memset(iwave1,0,EventSize*sizeof(int));
    memset(iwave2,0,EventSize*sizeof(int));
}

double PLSEvent::DTLinear()
{
    double zero1,zero2,slope;
    BuildCFD();
    slope = (dwave1[(int)(Interval1Max/0.5)] - dwave1[(int)(Interval1Min/0.5)])/0.5;
    zero1 = Interval1Min - dwave1[(int)(Interval1Min/0.5)]/slope;

    slope = (dwave2[(int)(Interval2Max/0.5)] - dwave2[(int)(Interval2Min/0.5)])/0.5;
    zero2 = Interval2Min - dwave2[(int)(Interval2Min/0.5)]/slope;

    return(zero1 - zero2);
}

double PLSEvent::DTCubic()
{
    double ytemp,xtemp,zero1,zero2;
    int err;
    //err = CheckEvents();
    //if( err != 0)
    //    return -1;
    CRRCSmooth();
    BuildCFD();

    alglib::spline1dbuildcubic(time,wave1,spline1);
    alglib::spline1dbuildcubic(time,wave2,spline2);
    dwave1 = wave1.getcontent();
dtime = time.getcontent();

do
{
    xtemp = Interval1Min + (Interval1Max - Interval1Min)/ 2.0;
    ytemp = alglib::spline1dcalc(spline1,xtemp);
    if(ytemp < 0)
        Interval1Max = xtemp;
    if(ytemp > 0)
        Interval1Min = xtemp;
} while(abs(ytemp) > 1e-4);

zero1 = xtemp;

do
{
    xtemp = Interval2Min + (Interval2Max - Interval2Min)/ 2.0;
    ytemp = alglib::spline1dcalc(spline2,xtemp);
    if(ytemp < 0)
        Interval2Max = xtemp;
    if(ytemp > 0)
        Interval2Min = xtemp;
} while(abs(ytemp) > 1e-4);

zero2 = xtemp;

return(zero1 - zero2);
Appendix C: analysis.h

#ifndef ANALYSIS_H
#define ANALYSIS_H

#include "ap.h"

// disable some irrelevant warnings
#if (AE_COMPILER==AE_MSVC)
#pragma warning(disable:4100)
#pragma warning(disable:4127)
#pragma warning(disable:4702)
#pragma warning(disable:4996)
#endif

#include "alglibmisc.h"
#include "alglibinternal.h"
#include "linalg.h"
#include "statistics.h"
#include "dataanalysis.h"
#include "specialfunctions.h"
#include "solvers.h"
#include "optimization.h"
#include "diffequations.h"
#include "fasttransforms.h"
#include "integration.h"
#include "interpolation.h"

class PLSEvent
{

private:

    alglib::real_1d_array time, wave1, wave2;

    int *iwave1, *iwave2;
    double *dwave1, *dwave2, *dtime;
    double CF;
    int Delay;
    double Interval1Min, Interval1Max, Interval2Min, Interval2Max;
    int EventSize;
}
alglib::spline1dinterpolant spline1, spline2;

//int CheckEvents();
void BuildCFD();

public:

double DTCubic();
double DTLinear();
double DTPoly( int );
void CRRCSmooth( int );
void LoadEvent( int *, int *, int, double, int);
void ResetData();
};

#endif
# Appendix D: ConsoleFunctions.cpp

```cpp
#include "ConsoleFunctions.h"
#include <stdio.h>
#include <fstream>
#include <iostream>
#include <string.h>
#include <time.h>

void Welcome()
{
    system("CLS");
    std::cout << "********************************************************************************
        |       |       |        |        |        |        |        |        |        |        |        |        |
        |       |       |        |        |        |        |        |        |        |        |        |        |
        |       |       |        |        |        |        |        |        |        |        |        |        |
        |       |       |        |        |        |        |        |        |        |        |        |        |
        |       |       |        |        |        |        |        |        |        |        |        |        |
        |       |       |        |        |        |        |        |        |        |        |        |        |
        |       |       |        |        |        |        |        |        |        |        |        |        |
        |       |       |        |        |        |        |        |        |        |        |        |        |
        |       |       |        |        |        |        |        |        |        |        |        |        |
        |       |       |        |        |        |        |        |        |        |        |        |        |
    std::cout << "********************************************************************************
        |       |       |        |        |        |        |        |
        |       |       |        |        |        |        |        |
        |       |       |        |        |        |        |        |
        |       |       |        |        |        |        |        |
        |       |       |        |        |        |        |        |
        |       |       |        |        |        |        |        |
        |       |       |        |        |        |        |        |
    std::cout << "********************************************************************************
        |       |       |        |        |        |        |        |
        |       |       |        |        |        |        |        |
        |       |       |        |        |        |        |        |
        |       |       |        |        |        |        |        |
    
    std::cout << "Please select one of the following options \n";
    std::cout << "'v' View Digitizer Parameters\n";
    std::cout << "'e' Edit Digitizer Parameters\n";
    std::cout << "'a' Goto aquisition menu\n";
    std::cout << "'l' Add details to log file (executed after initialization)\n";
    std::cout << "'P' Goto data processing menu\n";
    std::cout << "'q' Quit the program\n";
}

void PrintConfigMenu()
{
    Welcome();
    std::cout << "Please select one of the following options \n";
    std::cout << "'v' View Digitizer Parameters\n";
    std::cout << "'e' Edit Digitizer Parameters\n";
    std::cout << "'a' Goto aquisition menu\n";
    std::cout << "'l' Add details to log file (executed after initialization)\n";
    std::cout << "'P' Goto data processing menu\n";
    std::cout << "'q' Quit the program\n";
}
```

57
int LogEntry(std::ofstream& logfile, int handle)
{
    CAEN_DGTZ_EnaDis_t DESmode;
    uint32_t outputvalues;
    int err = 0;

    struct tm *newtime;
    char am_pm[] = "AM";
    __time64_t long_time;
    _time64( &long_time ); /* Get time as long integer. */
    newtime = _localtime64( &long_time ); /* Convert to local time. */
    _time64( &long_time );         /* Get time as long integer. */
    newtime = _localtime64( &long_time ); /* Convert to local time. */

    if( newtime->tm_hour > 12 )    /* Set up extension. */
        strcpy( am_pm, "PM" );
    else if( newtime->tm_hour > 12 )   /* Convert from 24-hour */
        newtime->tm_hour -= 12;    /* to 12-hour clock. */
    else if( newtime->tm_hour == 0 )   /* Set hour to 12 if midnight. */
        newtime->tm_hour = 12;

    logfile << "Log for Run starting " << asctime( newtime ) << am_pm << "n;";
    err += CAEN_DGTZ_GetDESMode(handle, &DESmode);
    logfile << "DES Mode: " << std::endl << DESmode << "n;";
    err += CAEN_DGTZ_GetChannelEnableMask(handle, &outputvalues);
    logfile << "Channel Mask: " << std::hex << outputvalues << "n;";
    err += CAEN_DGTZ_GetChannelDCOffset(handle,1, &outputvalues);
    logfile << "Channel 1 DC offset: " << std::hex << outputvalues << "n;";
    err += CAEN_DGTZ_GetChannelDCOffset(handle,3, &outputvalues);
    logfile << "Channel 3 DC offset: " << std::hex << outputvalues << "n;";
    err += CAEN_DGTZ_GetChannelTriggerThreshold(handle,1,&outputvalues);
    logfile << "Channel 1 Threshold: " << std::dec << outputvalues << "n;";
    err += CAEN_DGTZ_GetChannelTriggerThreshold(handle,3,&outputvalues);
    logfile << "Channel 3 Threshold: " << std::dec << outputvalues << "n;";

    return err;
}

void ExpParams(std::ofstream& logfile)
{
    char str[256];
    std::cout << "What is the voltage for Channel 1: ";
    std::cin.getline(str,256);
    logfile << "Channel 1 voltage:\t" << str << "n;";
std::cout << "What is the voltage for Channel 3: ";
std::cin.getline(str,256);
logfile << "Channel 3 voltage:\t" << str << " V\n";
std::cout << "Enter source isotope (e.g. NA22): ";
std::cin.getline(str,256);
logfile << "Source Isotope:\t" << str << " \n";
std::cout << "Enter source strength in Bq: ";
std::cin.getline(str,256);
logfile << "Source strength:\t" << str << " Bq\n";
std::cout << "Enter any other Sample information: ";
std::cin.getline(str,256);
logfile << "Sample details: " << str << " \n";

int ConfigureDigitizer(int handle, PLS_Parameters Params)
{

CAEN_DGTZ_ErrorCode err;

err = CAEN_DGTZ_Reset(handle);
if(err)
{ std::cout<< "Cannot reset digitizer" << std::endl; return (int)err;}

err = CAEN_DGTZ_SetDESMode(handle,Params.DESMode); // enable/disable dual edge sampling ( 2GS/s )
if(err)
{ std::cout<< "Cannot set DES mode" << std::endl; return (int)err;}

err = CAEN_DGTZ_SetChannelEnableMask(handle,Params.ChannelMask); // enable/disable channels
if(err)
{ std::cout<< "Cannot set channel enable mask" << std::endl; return (int)err;}

err = CAEN_DGTZ_SetAcquisitionMode(handle,Params.AcqMode); // set acq control mode
if(err)
{ std::cout<< "Cannot set acquisition mode" << std::endl; return (int)err;}

err = CAEN_DGTZ_SetRecordLength(handle,Params.RecordLength); // samples per event
if(err)
{ std::cout<< "Cannot set recored length" << std::endl; return (int)err;}

err = CAEN_DGTZ_SetPostTriggerSize(handle,Params.PostTriggerSize); // % of event samples come after trigger
if(err)
{
    std::cout<< "Cannot set post trigger size" << std::endl; return (int)err;
}

err = CAEN_DGTZ_SetChannelPulsePolarity(handle,1,Params.Polarity); // set input polarity for channel 1
if(err)
{
    std::cout<< "Cannot set pulse polarity for channel 1" << std::endl; return (int)err;
}

err = CAEN_DGTZ_SetChannelPulsePolarity(handle,3,Params.Polarity); // negative input polarity for channel 3
if(err)
{
    std::cout<< "Cannot set pulse polarity for channel 3" << std::endl; return (int)err;
}

err = CAEN_DGTZ_SetSWTriggerMode(handle,Params.TriggerMode); // Set trigger mode (acq only, acq and trig. out, or trig out only)
if(err)
{
    std::cout<< "Cannot set software trigger" << std::endl; return (int)err;
}

err = CAEN_DGTZ_SetChannelTriggerThreshold(handle,1,Params.Threshold); // set trigger threshold in ADC counts for channel 1
if(err)
{
    std::cout<< "Cannot set trigger threshold " << err << std::endl; return (int)err;
}

//err = CAEN_DGTZ_SetChannelTriggerThreshold(handle,3,Params.Threshold); // set trigger threshold in ADC counts for channel 1
//if(err)
//{
//    std::cout<< "Cannot set trigger threshold " << err << std::endl; return (int)err;
//}

err = CAEN_DGTZ_SetChannelDCOffset(handle,1,Params.DCOffeset); // set DC offset for channel 1
if(err)
{
    std::cout<< "Cannot set dc offset for channel 1" << std::endl; return (int)err;
}

//err = CAEN_DGTZ_SetChannelDCOffset(handle,3,Params.DCOffeset); // set DC offset for channel 1
//if(err)
//{
//    std::cout<< "Cannot set dc offset for channel 1" << std::endl; return (int)err;
//}

err = CAEN_DGTZ_SetChannelSelfTrigger(handle,Params.SelfTrigger,Params.ChannelMask); // enable channel self triggers for channel mask set channels
if(err)
{
    std::cout<< "Cannot set dc offset" << std::endl; return (int)err;
}

err = CAEN_DGTZ_SetMaxNumEventsBLT(handle,Params.MaxBLT); // max number of events stored in each buffer
if(err)
{
    std::cout<< "Cannot set max blt" << std::endl; return (int)err;
}
err = CAEN_DGTZ_WriteRegister(handle,0x1388,0x5d); // enable channel 3 for setting DC offset via register setting
if(err)
{
    std::cout<< "Cannot enable channel 3 for DC offset configuration" << std::endl; return (int)err;
}

err = CAEN_DGTZ_WriteRegister(handle,0x1398,Params.DCOffeset);// Set channel 3 DC offset via register setting
if(err)
{
    std::cout<< "Cannot set dc offset for channel 1" << std::endl; return (int)err;
}

err = CAEN_DGTZ_WriteRegister(handle,0x1398,Params.DCOffeset);// Set channel 3 DC offset via register setting
if(err)
{
    std::cout<< "Cannot set dc offset for channel 3" << std::endl; return (int)err;
}

err = CAEN_DGTZ_WriteRegister(handle,0x1380,Params.Threshold); // Set Channel 3 trigger threshold via register setting
if(err)
{
    std::cout<< "Cannot set Trigger Threshold for channel 3" << std::endl; return (int)err;
}
return (int)err;

// Register based configurations will hopefully be removed, but due to a bug setting DESMODE = true the config functions used for chan 1 do not work
uint32_t temp1,temp2;

err = CAEN_DGTZ_WriteRegister(handle,0x809C,0X2);
do{
    err = CAEN_DGTZ_ReadRegister(handle,0x1188,&temp1);
    err = CAEN_DGTZ_ReadRegister(handle,0x1388,&temp2);
    if(!((temp1 & ( 1 << 6)) >> 6) && !((temp2 & ( 1 << 6)) >> 6))
        continue;
}while(1);
err = CAEN_DGTZ_WriteRegister(handle, 0x809C, 0x0);

void PrintParams(PLS_Parameters Params)
{
    PrintConfigMenu();
    std::cout << "nDC offset:\t\t" << Params.DCOffset;
    std::cout << "nMax events per buffer:\t\t" << std::dec << Params.MaxBLT;
    std::cout << "n% samples post trigger:\t\t" << Params.PostTriggerSize;
    std::cout << "nSamples per event:\t\t" << Params.RecordLength;
    std::cout << "nTrigger threshold:\t\t" << Params.Threshold;
}

void EditParams(PLS_Parameters& Params)
{
    PrintConfigMenu();
    std::cout << "nSet DC offset:\t\t"; std::cin >> Params.DCOffset;
    std::cout << "nSet Max events per buffer:\t\t"; std::cin >> std::dec >> Params.MaxBLT;
    std::cout << "nSet % samples post trigger:\t\t"; std::cin >> Params.PostTriggerSize;
    std::cout << "nSet Samples per event:\t\t"; std::cin >> Params.RecordLength;
    std::cout << "nSet Trigger threshold:\t\t"; std::cin >> Params.Threshold;
    std::cout << "n";
    // Need to implement enum types for input in the future
}

void PrintRunMenu()
{
    Welcome();
    std::cout << "Please Select one of the following options:" << std::endl;
    std::cout << "s' start run" << std::endl;
    std::cout << "S' Stop run" << std::endl;
    std::cout << "t' Send Software Trigger" << std::endl;
    std::cout << "w' Write events to file" << std::endl;
    std::cout << "w' Close Output Files" << std::endl;
    std::cout << "c' Toggle Cointidence filter" << std::endl;
    //std::cout << "l' Set Time limit for a run " << std::endl;
    std::cout << "q' Quit the program\n\n" << std::endl;
}
Appendix E: ConsoleFunctions.h

#ifndef CONSOLE_H
#define CONSOLE_H

#include <iostream>
#include <CAENDigitizer.h>
#include <CAENDigitizerType.h>

void Welcome(); // Initial Welcome menu for startup

void PrintConfigMenu(); // Menu for setting Run parameters

void PrintRunMenu(); // Print menu for Runtime options

void ExpParams(std::ofstream&); // Add voltage and source details to log file entry

int LogEntry(std::ofstream&, int); // log entry for each run

typedef struct PLS_Parameters
{
    CAEN_DGTZ_EnaDis_t DESMode;
    CAEN_DGTZ_AcqMode_t AcqMode;
    uint32_t RecordLength;
    uint32_t PostTriggerSize;
    CAEN_DGTZ_PulsePolarity_t Polarity;
    CAEN_DGTZ_TriggerMode_t TriggerMode;
    uint32_t Threshold;
    uint32_t DCOffset;
    uint32_t ChannelMask;
    CAEN_DGTZ_TriggerMode_t SelfTrigger;
    uint32_t MaxBLT;
};

void PrintParams(PLS_Parameters Params);

void EditParams(PLS_Parameters& Params);

int ConfigureDigitizer( int handle, PLS_Parameters params );
#endif