Preparatory work for the development of a data-taking algorithm for a prototype of the backward endcap calorimeter of the PANDA experiment

Bachelor thesis

written by

Pascal Lautz

Bachelor thesis in physics
Fachbereich Physik, Mathematik und Informatik (FB 08)
Johannes Gutenberg-Universität Mainz
12. June 2012

1. Reader: Prof. Dr. Frank Maas
2. Reader: Prof. Dr. Josef Pochodzalla

Vorarbeiten zur Entwicklung eines Datenaufnahmealgorithmus' für einen Prototypen der Rückwärtsendkappe des PANDA-Experiments

Bachelorarbeit

von

Pascal Lautz

Bachelorarbeit in Physik
Fachbereich Physik, Mathematik und Informatik (FB 08)
Johannes Gutenberg-Universität Mainz
12. Juni 2012

1. Gutachter: Prof. Dr. Frank Maas
2. Gutachter: Prof. Dr. Josef Pochodzalla

Inhaltsverzeichnis

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Mainz, den 12. Juni 2012

Pascal Lautz
Kaiserstraße 16
55116 Mainz
E-Mail: lautzp@gmail.com
Matrikelnummer: 2658260

1. Einleitung

...

Daß ich nicht mehr mit sauerm Schweiß,
Zu sagen brauche, was ich nicht weiß;
Daß ich erkenne, was die Welt
Im Innersten zusammenhält,

...

Johann Wolfgang von Goethe, Faust Teil I

Um immer weiter in die Tiefe der Materiestruktur vorzudringen, müssen physikalische Prozesse auf extremen Skalen erforscht werden. So können sowohl die Astrophysik als auch die Kern-und Teilchenphysik erstaunliche neue Erkenntnisse über den Aufbau der Materie liefern. Um heutzutage neue Erkenntnisse über Physik zu erhalten, die sich auf Skalen der Größenordnung von Atomkernen abspielt, ist ein großer technischer Aufwand notwendig. Um diesen hohen Aufwand an Kapital, Wissen und Technik schultern zu können, schließen sich Forschungsinstitute rund um den Globus zu Wissenschaftsallianzen zusammen.

Das Helmholtzzentrum für Schwerionenforschung in Darmstadt (GSI) ist eine Forschungseinrichtung einer solchen Wissenschaftsallianz. Derzeit wird dort ein neues Beschleunigerzentrum FAIR gebaut, das neue Erkenntnisse der Physik am kleinen Ende der uns bekannten Längenskala bringen wird. Das Institut für Kernphysik der Universität Mainz unterstützt den Bau des Beschleunigerzentrum mit dem Bau eines elektromagnetischen Kalorimeters für das PANDA-Experiment. Bevor jedoch das finale Kalorimeter gebaut werden kann, gilt es viele technische Herausforderungen zum Bau moderner Detektoren zu bestehen. Dies erfordert den Bau von Prototypen verschiedener Detektorkomponenten. Zurzeit wird ein Prototyp mit acht Bleiwolframat-Kristallen entwickelt. Um den Prototypen des Detektors auf Leistungsfähigkeit und charakteristisches Verhalten testen zu können, muss es eine Möglichkeit geben, Messdaten zu erfassen. Diese Daten sollen digital vorliegen, um eine Datenanalyse zu einem späteren Zeitpunkt zu vereinfachen.

Diese Bachelorarbeit befasst sich mit dem Erstellen und Testen eines Datenaufnahmealgorithmus für diesen Prototypen.

In Kapitel 2 wird zunächst ein Überblick geschaffen um zu zeigen, in welchem physikalischen Kontext das PANDA-Experiment und speziell die Rückwärtsendkappe stehen und worauf die Anforderungen an diesen Detektor begründet sind.
Danach wird in Kapitel 3 das nötige Grundwissen der Signalerzeugung im Detektor sowie die technische Umsetzung einer Signaldigitalisierung vermittelt. Um einen funktionierenden Datenaufnahmealgorithmus erstellen zu können, ist es von großer Wichtigkeit, den Weg vom einfallenden Teilchen zur gemessenen Spannung und weiter zu einem digitalen Wert nachvollziehen zu können.
In Kapitel 4 wird anschließend das Vorgehen zum Erstellen eines Datenaufnahmealgorithmus' detailliert beschrieben. Zuerst wird auf einen Aufbau eingegangen, der genutzt wurde, um Signale ähnlich einfallender Teilchen zu erzeugen. Darauf aufbauend wird der Aufbau und die Funktionsweise des Programmes zum Auslesen eines Analog-Digital-Konverters dokumentiert, das charakteristische Verhalten einzelner Bauteile der Signalaufnahme untersucht und eine Möglichkeit gezeigt, sich überlagernde Signale zu separieren.
Das letzte Kapitel fasst die Ergebnisse der neunwöchigen Arbeit zusammen und gibt einen Ausblick auf die verbleibenden Aufgaben, die nötig sind, um die Datenaufnahme zu optimieren.

2. Das elektromagnetische Kalorimeter des PANDA-Experiments

Dieses Kapitel beschäftigt sich mit den Grundlagen des PANDA-Experiments und zeigt, welche Anforderungen an den PANDA-Detektor gestellt werden. Quellen für dieses Kapitel sind der Technical Design Report[1] und der Technical Progress Report [2] des PANDA-Experiments, die hier kurz zusammengefasst werden.

2.1. FAIR

Auf der Suche nach neuer Physik und dem Wunsch, physikalische Theorien belegen zu können, haben sich mehrere Länder zusammengeschlossen, um ein internationales Beschleunigerzentrum zu bauen. Dieses trägt den Namen FAIR (**F**acility of **A**ntiproton and **I**on **R**esearch) und wird derzeit auf dem Gelände des Helmholtzzentrums für Schwerionenforschung (GSI) in Darmstadt gebaut. Dieses Zentrum ermöglicht neue Einsichten in Bereiche der Teilchen-, Kern-, Plasma- und Atomphysik, die bis dato experimentell unzugänglich waren. Antiproton- und Ionenstrahlen mit hohen Intensitäten erlauben es, eine Fülle neuartiger Experimente durchzuführen. Abbildung 2.1 verdeutlicht die Größe der neuen Beschleunigeranlage.

Das wissenschaftliche Programm von FAIR beruht auf vier Säulen: Im Bereich NUSTAR werden Untersuchen zur Kernstruktur durchgeführt mit dem Ziel, die Elementsynthese im Universum besser zu verstehen. Der Bereich APPA beschäftigt sich mit Fragen zur Plasma- und Atomphysik. In CBM wird der kritische Punkt des Phasendiagramms der starken Wechselwirkung (Quantenchromodynamik) untersucht, dazu werden Schwerionenexperimente durchgeführt. Das PANDA-Experiment beschäftigt sich mit der Hadronenspektroskopie und -struktur mithilfe von Proton-Antiproton-Annihilationsreaktionen.

2.2. PANDA

Eines der Experimente am Beschleunigerzentrum FAIR ist das PANDA-Experiment. PANDA steht für „Anti**P**roton **An**nihilations at **Da**rmstadt“. Protonen und Antiprotonen annihilieren und können dabei eine Vielzahl von Endzuständen erzeugen. Charakteristisch für dieses Experiment ist die sehr genau definierte Strahlenergie des Antiproton-Strahls, die auf ein fixiertes Target aus gefrorenen Wasserstoff-Pellets oder ein Wasserstoff-Clusterjet treffen.

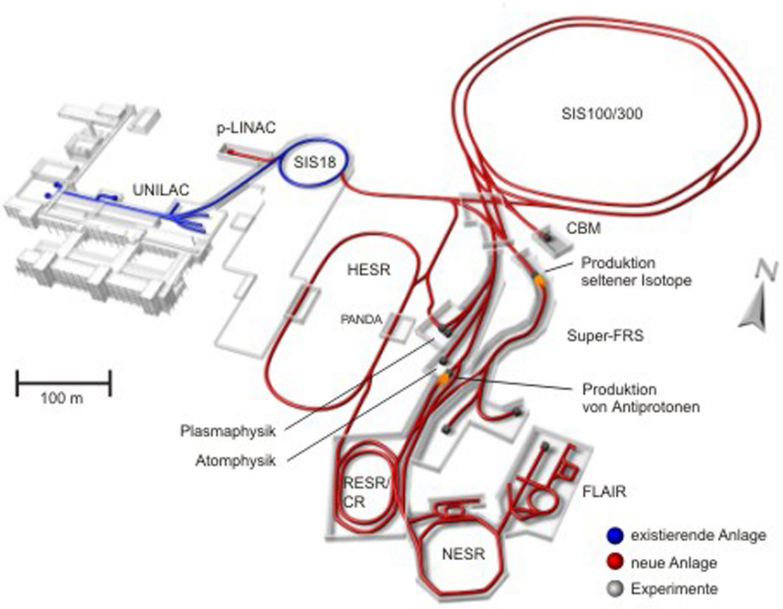


Abbildung 2.1.: Übersicht über den geplanten Bau des FAIR-Beschleunigers.

PANDA besteht thematisch aus vier Teilen: Hadronen-Spektroskopie, Hadronen in Materie, Nukleonstruktur und Hyperkerne. Unter anderem werden neue Erkenntnisse über Doppelhyperkerne, Hyperon-Hyperon-Wechselwirkung, Spektroskopie im Bereich des Charmoniums, exotische Hadronenzustände wie zum Beispiel sogenannte „Glueballs“ oder Hybridzuständen sowie neue Erkenntnisse über die Nukleonstruktur mithilfe von elektromagnetischen Prozessen im Inneren von Nukleonen erwartet. Diese Ansprüche bilden die Grundlage für die Entwicklung von Detektorkomponenten.

So sollen Energien und Massen bis auf 50-100 keV/c² genau über einen Impulsbereich von 100 MeV/c - 8 GeV/c gemessen werden können und nahezu der gesamte Raumwinkel abgedeckt werden. Die Impulsauflösung soll ca. 1% betragen und der Detektor bis zu 20 Millionen Ereignisse pro Sekunde verarbeiten können. Um eine solch hohe Rate an Ereignissen verarbeiten zu können, wird eine selbstgetriggerte Elektronik eingesetzt, die schon im Vorfeld wichtige Ereignisse herausfiltern kann. Der Detektor des PANDA-Experiments wurde speziell für diese physikalischen und technischen Herausforderungen konzipiert.[1]

Dieser Detektor besteht aus 2 Teilen:

1. Target-Spektrometer: Dieses Spektrometer misst einen Raumwinkelbereich von 22°-140°. Es ist ähnlich einer Zwiebel aufgebaut, wobei einzelne Schichten Detektoren darstellen. Das Spektrometer besteht innen um die Wechselwirkungszone herum aus einem Mikrovertex-Detektor (MVD), der versetzte Zerfalls-Vertices detektieren kann.

Dieser Detektor wird umhüllt von einem Gas-Spur-Detektor (auch Strawtubes genannt) zur 3-dimensionalen Rekonstruktion der Teilchenspur. Um diesen Detektor herum befindet sich ein abbildender Ring-Cherenkov-Detektor, der die Teilchen nach Art unterscheiden kann. Die nächste Schicht ist ein elektromagnetisches Kalorimeter mit 11360 Bleiwolframat-Kristallen, welches dazu dient, die Teilchenenergie zu messen. Das Kalorimeter wird umgeben von einem supraleitenden Solenoiden, der ein sehr homogenes Magnetfeld von bis zu 2 Tesla ($\pm$ 2%) aufbauen kann. Der Solenoid ist abschließend von einem Muon-Detektor (Driftröhren) umgeben. Für große Raumwinkel ($\approx$ 141°-170°) können Teilchen-Energien mithilfe eines elektromagnetischen Kalorimeters gemessen werden, das aus 592 Bleiwolframat-Kristallen besteht.[1][2]

2. Vorwärts-Spektrometer: Dieses Spektrometer ist darauf ausgelegt, Teilchen mit hohen Energien und Impulsen bis zu 8 GeV/c messen zu können. Das Vorwärts-Kalorimeter misst Partikel in einem Winkel von 5°-21°/10°-21° Raumwinkel vom einfallenden Strahl in vertikaler/horizontaler Ebene. Die Spurrekonstruktion wird über Mini-Driftkammern realisiert, ein abbildender Ring-Cherenkov-Detektor und ein elektromagnetisches Kalorimeter messen Teilchenart und Energie. Die Spurrekonstruktion erfolgt in einem starken Dipol-Magneten und einem Muondetektor.[1][2]

2.3. Die Rückwärts-Endkappe des elektromagnetischen Kalorimeters am PANDA-Experiment

Um im gesamten Raumwinkelbereich um das Target Teilchen detektieren zu können, wird das Target von mehreren Detektoren umschlossen. Das Institut für Kernphysik der Universität Mainz entwickelt die Rückwärtsendkappe des PANDA-Detektors. Es handelt sich bei diesem Detektor um ein homogenes, absorbierendes elektromagnetisches Kalorimeter zum Messen der Energien rückgestreuter Teilchen. So können in diesem Raumbereich Teilchen, die über die elektromagnetische Wechselwirkung interagieren (Photonen und Elektronen), detektiert werden.
Der Detektor besteht aus 592 Bleiwolframat-Kristallen, an denen jeweils 2 Large Area Avalanche-Photodioden (LAAPD) angebracht sind. Die Kristalle werden auf -25° C heruntergekühlt, um die Lichtausbeute um etwa einen Faktor 4 zu erhöhen. Um Elektronen mit Energien von ca. 10 MeV detektieren zu können, ist es sehr wichtig, die Lichtausbeute zu maximieren, da Teilchen dieser Energien nur sehr schwache Lichtimpulse erzeugen. Durch das Anbringen von 2 APD's an je einem Kristall vergrößert sich die aktive Fläche zum Detektieren von Szintillations-Photonen. Ein weiterer Vorteil der APD's ist die gute Quanteneffizienz, die bei

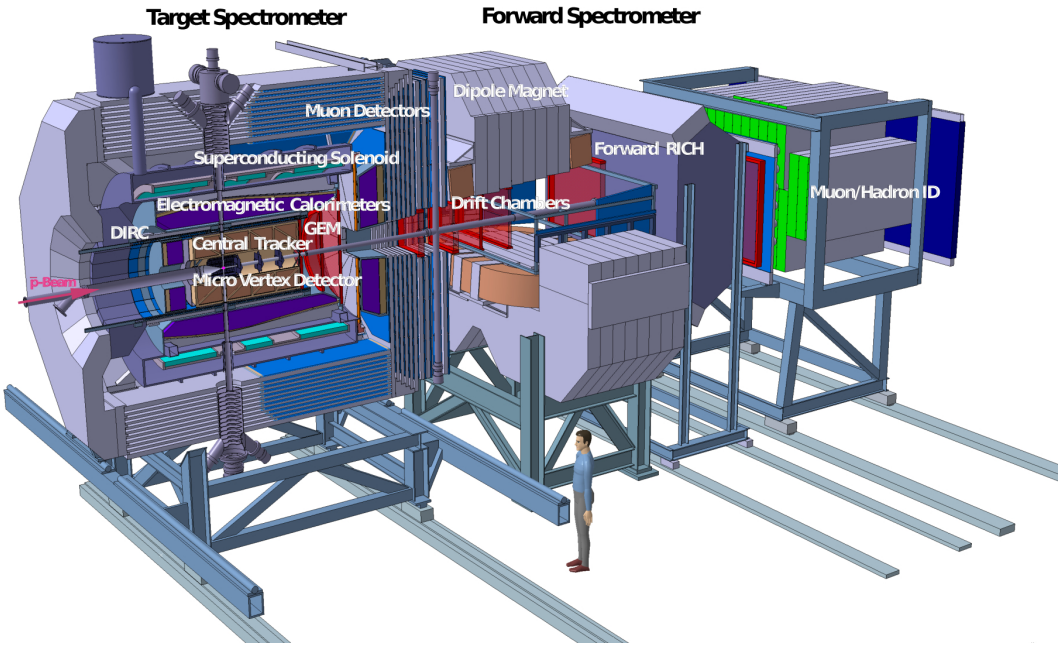


Abbildung 2.2.: Aufbau des PANDA-Detektors. In die Mitte des Target-Spektrometers wird ein Wasserstoff-Clusterjet oder ein Wasserstoff-Pellet-Target eingesetzt. Um dieses Target herum sind eine Vielzahl verschiedener Detektoren zum Messen von Teilcheneigenschaften angebracht. Für große Raumwinkel (>141°) existiert eine Rückwärts-Endkappe, ein elektromagnetisches Kalorimeter mit Bleiwolframat-Kristallen. Reaktionsteilchen mit kleinen Raumwinkeln (<22°) werden von einem Vorwärts-Spektrometer erfasst, das speziell auf hohe Eventraten ausgelegt ist.

ca. 70% liegt (im Vergleich zu ca. 20% bei Photomultipliern).[1]
Der genaue Aufbau der Endkappe wird derzeit durch den Bau von Prototypen erforscht und optimiert. Der detaillierte Aufbau eines Prototypen mit 8 Kristallen wird im folgenden Kapitel erläutert.

3. Der experimentelle Aufbau eines Prototyps mit 8 Kristallen

3.1. Anorganische Szintillatoren / Bleiwolframat-Kristalle

Um Teilchen zu detektieren, die über die elektromagnetische Wechselwirkung interagieren, werden in der Rückwärtsendkappe des PANDA-Experiments anorganische Szintillatoren genutzt. Szintillatoren haben eine große Sensibilität oberhalb einer gewissen Schwellenenergie und verhalten sich darüber nahezu linear zur einfallenden Teilchenenergie. Des Weiteren besitzen sie eine schnelle Ansprechzeit, was bedeutet, dass bei Experimenten, bei denen eine genaue Zeitauflösung entscheidend ist, kleinere Fehler erhalten werden als bei anderen Detektoren. Außerdem reduziert sich dadurch die Totzeit eines Detektors. Die Kristalle bestehen aus Bleiwolframat ($PbWO_4$) und wurden gegenüber dem CMS-Experiment am CERN, an dem ebenfalls Bleiwolframat-Kristalle genutzt werden, in Hinsicht auf Lichtausbeute bei tiefen Temperaturen verbessert. Diese Weiterentwicklung wird PWO-II genannt. Die Lichtausbeute der Kristalle vervierfacht sich beim Herunterkühlen von +25° C auf -25°C.[1]
Um auch bei niedrigen Teilchenenergien genaue Messungen zu erhalten, ist dies von großer Bedeutung. In Tabelle 3.1 sind einige wichtige charakteristische Größen für Bleiwolframat zusammengefasst.

Parameter	Wert
ρ	8,3 g/cm³
τ_{decay}	10 ns (s) /30 ns (l)
$\frac{dE}{dx}$ (MIP)	10,1 MeV/cm
λ_{max}	420 nm (s) /425 nm (l)
Lichtausbeute (-25°C)	500 Photonen/MeV

Tabelle 3.1.: Spezifische Größen von Bleiwolframat ($PbW0_4$). [1] [5]

s	schnelle Zerfallskomponente
l	langsame Zerfallskomponente
MIP	minimal ionisierende Partikel

Eine weitere Charakteristik des Szintillators sind die Zerfallszeiten eines angeregten Zustands. Die verwendeten Kristalle haben eine für anorganische Szintillatoren schnelle Zerfallszeit, so dass 95% der erzeugten Photonen in einem Zeitraum von 300-400 ns gemessen werden können.

3.2. Avalanche- Photodioden

Um kleinste Lichtintensitäten, die von einfallenden Teilchen in den Bleiwolframat-Kristallen erzeugt werden, messen zu können, werden an den Kopfflächen der Kristalle Photodioden angeklebt. Für die Rückwärtsendkappe werden großflächige Avalanche-Photodioden (LAAPD) genutzt, um einen großen Bereich der Kopfflächen abzudecken und so möglichst viele Photonen zu detektieren. Die genutzten Dioden des Herstellers Hamamatsu der Serie S8664 und des Fabrikats S8664-1010 (Datenblatt siehe [3], Maße abweichend 14x7 mm) sind aus Silicium gefertigt und besitzen eine Quanteneffizienz von ca. 70%. Avalanche-Photodioden (APD) sind im Gegensatz zu normalen PIN-Photodioden anders aufgebaut. Es wird ein Schichtaufbau mit $p^+ - i - p - n^+$ dotierten Schichten verwendet, wobei p^+ stark positiv und n^+ stark negativ dotiert ist. Wird eine hohe Spannung (300-350 Volt), auch Bias genannt, in Sperrrichtung angelegt (Minuspol an die p^+-Schicht und Pluspol an n^+-Schicht), so vergrößert sich die Raumladungszone. Fällt ein Photon mit einer Energie $E = h\nu$ ein, die größer ist als die Bandlücke, so regt das Photon ein Elektron in das Leitungsband an und erzeugt so ein Elektronen-Loch-Paar. Durch die hohe angelegte Spannung wird das Loch in Richtung der p^+-Schicht und das Elektron in Richtung der n^+-Schicht beschleunigt. Am $p - n^+$-Übergang ist das elektrische Feld besonders stark (siehe Abb. 3.1), wodurch das Elektron stark beschleunigt wird. Es entsteht eine Kettenreaktion: Sekundärelektronen werden ausgelöst und dadurch einen messbarer Strom erzeugt. Es muss sichergestellt werden, dass die APD auf Licht der Wellenlängen empfindlich ist, die der Bleiwolframat-Kristall emittiert (in diesem Fall ca. 420 nm, vgl. Tabelle 3.1, [1](S.43, Abb. 4.6) und [3](S.2)). Im Rahmen der Bachelorarbeit wurde mit APDs gearbeitet, die für diesen Bereich geeignet sind und schon auf ihren Verstärkungsfaktor bei verschiedenen Temperaturen durchgemessen wurden. Um den optimalen Verstärkungsfaktor zu definieren, muss beachtet werden, dass ein größerer Verstärkungsfaktor auch ein größeres Rauschen verursacht. Dieses Rauschen entsteht durch thermische Anregungen in der Raumladungszone. Positiv wirkt sich daher aus, dass die APDs im PANDA-Experiment bei -25℃ betrieben werden, da dies das Rauschen verkleinert. Details zu einzelnen APDs und ihren Parametern finden sich in den zugehörigen Datenblättern, die für jede APD einzeln bei Andrea Wilms (`a.wilms@gsi.de`) erhältlich sind.

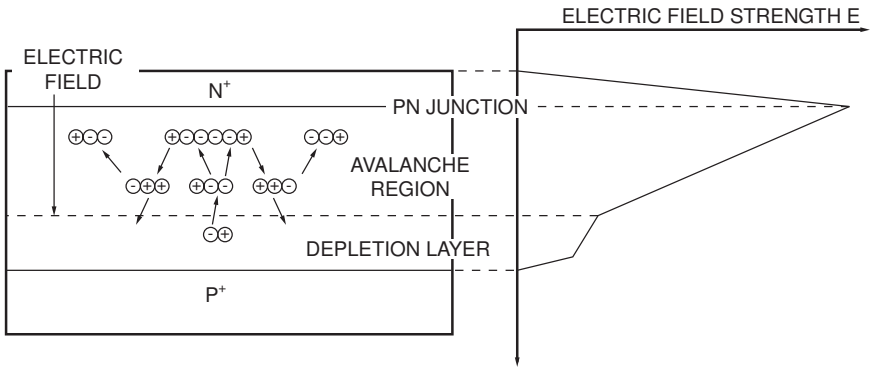


Abbildung 3.1.: Funktionsweise einer APD. Links in der Grafik ist die Bewegungsrichtung der Elektronen/Löcher und der Avalanche-Effekt bildlich dargestellt, rechts der Verlauf des elektrischen Feldes zwischen den verschieden dotierten Schichten in der APD.

3.3. Vorverstärker für die Avalanche-Photodioden

Um die Ladungsmenge, die von einer APD erzeugt wird zu detektieren, wird ein Vorverstärker an die APD angeschlossen. Der Vorverstärker ist ein speziell für das PANDA-Experiment gefertigter Verstärker der Universität Basel [4], der eine hohe Verstärkung bei niedrigem Rauschen garantiert. Der Vorverstärker ist ein ladungssensibler Verstärker, was bedeutet, dass er ein verstärktes Signal erzeugt, das proportional zur Anzahl der erzeugten Ladungen in der APD ist. Der Verstärker besitzt Anschlüsse für $\pm$ 6 Volt Versorgungsspannung, die APD, die Sperrspannung der APD (Reversed Bias) sowie für ein Signalkabel.

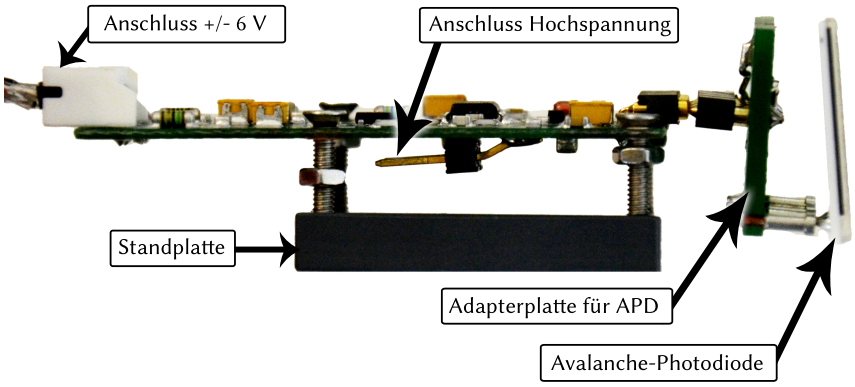


Abbildung 3.2.: Foto der Seitenansicht eines Vorverstärkers mit angeschlossener APD und Standplatte. Die Adapterplatte verlegt die Anschluss-Pins der APD, so dass dieser Aufbau weniger Platz benötigt.

Fällt nun ein Photon in die APD ein, so fließt ein elektrischer Strom. Dieser wird über einen Kondensator vom Bias getrennt (nur die AC-Komponente kann passieren) und in einem weiteren Kondensator gesammelt. Die Spannung des Kondensators wird über einen Operationsverstärker verstärkt. Parallel zu diesem Kondensator ist ein ohmscher Widerstand geschaltet, über den der Strom abfließen kann. Dieser Widerstand muss auf die Art der Signale optimiert werden, die gemessen werden sollen. Wird ein kleiner Widerstand gewählt, so fließt der Strom nicht vollständig in den Kondensator und es werden zu kleine Spannungen gemessen. Ist der Widerstand zu groß, verlängert sich das Entladen des Kondensator unnötig und es vergrößert sich damit das Risiko, dass beim nächsten einfallenden Signal der Kondensator noch eine Restladung besitzt.

Der Spannungsabfall des Kondensators wird durch eine Exponentialfunktion beschrieben:

$$U(t) = U_0 \cdot \exp\left(-\frac{t - t_0}{\tau}\right) . \tag{3.1}$$

τ entspricht der Abklingkonstante und ist eine Konstante des Vorverstärkers. Die typische Signalform ist in Abbildung 3.3 dargestellt.

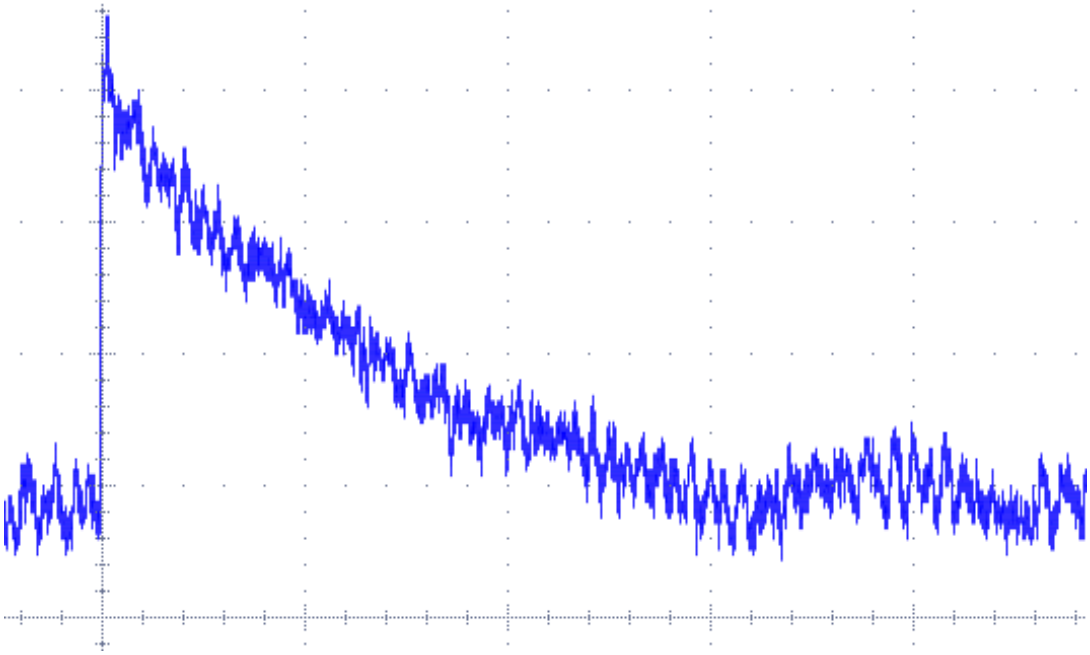


Abbildung 3.3.: Oszilloskopaufnahme eines typischen Signals des Vorverstärkers (Spannung: 20mV/div, Zeit:20 μs/div). Signal wurde in einem hochkant gestellten Kristall beim Raumtemperatur und einer APD mit einer Verstärkung von G$\approx$ 50 gemessen (Messung von kosmischer Höhenstrahlung).

Der Vorverstärker reagiert sensibel auf kleinste Ladungsmengen, jedoch macht ihn diese

große Sensibilität anfällig für elektromagnetische Strahlung aller Art sowie für Erschütterungen. Aus diesem Grund muss der Verstärker gut befestigt und optimimal abgeschirmt in einer geerdeten Metallbox betrieben werden.

3.4. Der aktuelle Aufbau des Prototypen

Derzeit wird ein Prototyp der Rückwärtsendkappe entwickelt, der wichtige Erkenntnisse zu Möglichkeiten des Aufbaus gibt, aber auch Schwierigkeiten und Probleme aufzeigt, die gelöst werden müssen. Zur Zeit der Erstellung dieser Arbeit befindet sich der experimentelle Aufbau in ständiger Veränderung, trotz dieses Umstandes soll jedoch ein Überblick über den aktuellen Stand geschaffen werden.

Das Kalorimeter wird im Rahmen einer größeren Kollaboration entwickelt. Fragen zur Auslesung der APDs sowie zur Auswahl und Charakterisierung der Kristalle wurden schon im Vorhinein genauestens untersucht. Dieser Prototyp soll spezielle Fragen, die aus den beengten räumlichen Verhältnissen der Rückwärtsendkappe resultieren, beantworten helfen.

Bei dem Prototyp handelt es sich um ein kleines elektromagnetisches Kalorimeter mit 8 Bleiwolframat-Kristallen. Diese sind in eine Reflektionsfolie eingepackt, um sie gegeneinander vor Photonen abzuschirmen. An einer von Folie unbedeckten Kopfseite sind zwei großflächige Avalanche-Photodioden aufgeklebt. Um messbare Signale zu erzeugen, wird das Signal jeder Photodiode durch einen Vorverstärker verstärkt. Es muss eine Möglichkeit gefunden werden, wie die Vorverstärker an die APD's angeschlossen werden. Derzeit stehen zwei Möglichkeiten zur Verfügung: die Vorverstärker können mithilfe einer Kunststoff-Kapsel, die am Ende des Kristalls sitzt, direkt auf die Photodiode aufgesteckt (siehe Abb. 3.4) und verschraubt werden oder aber es wird ein kleines Twisted-Pair-Kabel von der Diode zum Vorverstärker geführt und die Verstärker werden alle in einer zentralen Aufnahmevorrichtung befestigt. Vorteile der ersten Methode ist, dass keine Störsignale durch ein Kabel eingefangen werden können, Nachteil ist jedoch, dass die Vorverstärker bei Betrieb Wärme abgeben und sich deshalb möglichst weit entfernt von den gekühlten Kristallen befinden sollen. Welche der beiden Methoden sich schlussendlich durchsetzt, hängt von mehreren Faktoren ab. Da die Verstärker eine Leistungsaufnahme von 45-90 mW ([4]) besitzen, ist es vorteilhaft, sie möglichst weit entfernt vom Kristall zu plazieren, wobei sich in diesem Fall das Rauschen durch ein zwischengestecktes Kabel erhöhen könnte.

Die Kristalle werden in einer inneren, mit Kühlschläuchen umflossenen Aufnahmevorrichtung befestigt, die sich wiederum in einer mit einer Kopfplatte zu verschließenden Metallbox befindet. Die Kopfplatte ist mit Löchern versehen, durch die Kabel und Kühlschläuche geführt werden. Um zu verhindern, dass sich beim Kühlen Feuchtigkeit auf den elektroni-

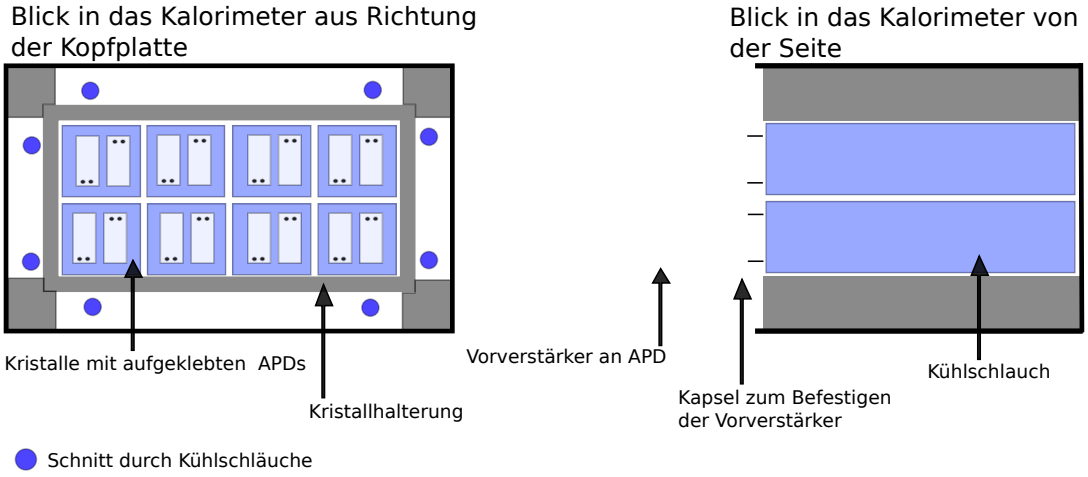


Abbildung 3.4.: Schematischer Aufbau des Prototypen mit 8 Kristallen. Links ein Blick aus Richtung der Kopfplatte in den Detektor, Vorverstärker und Kapsel sind nicht montiert. Rechts die Seitenansicht des Detektors mit angeschlossener Kapsel und Vorverstärker (Anschlüsse des Vorverstärkers nicht eingezeichnet.)

schen Bauteilen niederschlägt, ist es möglich, Stickstoff in die Box sprühen, um die Luftfeuchtigkeit zu verdrängen. Die Kühlung wird über einen externen Kühler realisiert, welcher mit 100%-igem Ethanol betrieben wird. Zur Isolierung der Box wird derzeit eine Hülle aus isolierendem Blauschaum genutzt, der mit Bauschaum verklebt wurde. Der gesamte Aufbau wird in einer lichtdichten Box betrieben, um Detektion von Umgebungslicht zu vermeiden, dass durch die Löcher in der Kopfplatte einfallen kann. Für eine detailliertere Beschreibung des Aufbaus wird hier auf die Bachelorarbeit von Christina Haberkorn (siehe [7]) verwiesen.

3.5. Der schnelle Analog-Digital-Wandler (FlashADC)

Um einkommende Signale des Vorverstärkers speichern zu können, müssen diese digitalisiert werden. Spannungen, die am Signalausgang des Vorverstärkers gemessen werden können, werden im PANDA-Experiment von schnellen Analog-Digital-Wandlern (Fast Sampling ADC, FlashADC oder auch FADC genannt) digitalisiert.

In einem Analog-Digital-Wandler werden einkommende Signale mit einem Referenz-Signal verglichen. Ein Komparator, ein elektronisches Bauteil, welches Spannungen vergleicht, gibt eine logische `1` oder `0` aus, je nachdem, ob der gemessene Wert kleiner oder größer als ein Referenzwert ist. Der Messbereich wird über die angelegte Referenzspannung definiert. Es gibt zwei verschiedene Typen der Realisierung. Es gibt ADCs, die das einkommende Signal

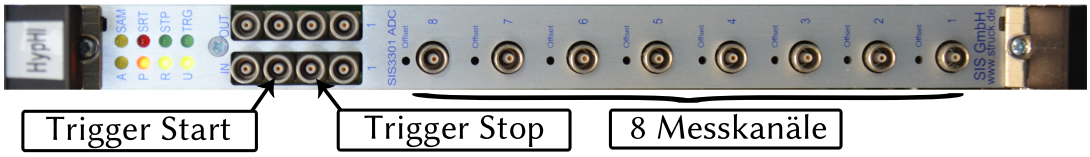


Abbildung 3.5.: Foto der Vorderansicht eines SIS 3301 FlashADC. Dies ist ein VME-Bus-Modul und kann softwarebasiert auf eine Abtast-Frequenz von 100 Mhz, 50 Mhz oder 25 Mhz eingestellt werden. Er speichert einen Spannungsbereich von maximal 5 Volt als 16-Bit-Wert ab und besitzt einen Speicher von 128kB pro Kanal. Eine Datenaufnahme von mehreren Messsignalen gleichzeitig ist möglich. Zum externen Triggern besitzt er zusätzliche Anschlüsse, über die die Datenaufnahme gestartet und gestoppt werden kann.

über einen Komparator mit mehreren Referenzwerten vergleichen und so mehrere Durchgänge zum Bestimmen eines Signals benötigen. Diese verursacht eine verhältnismäßig lange Zeit zur Digitalisierung. Eine andere Möglichkeit ist die Flash-Umsetzung in einem Flash-ADC, bei der für jedes Bit ein eigener Komparator bereitsteht; jeder dieser Komparatoren vergleicht das einkommende Signal mit nur einer Referenzspannung. Der Eingang für die Referenzspannung führt in diesem Fall zu einem linearen Spannungsteiler. So kann in einem Schritt eine Spannung digitalisiert werden, es werden jedoch für einen 16-bit-Wert $2^{16} - 1 = 65535$ Komparatoren benötigt, was sich in den Kosten eines Geräts wiederspiegelt. Die Genauigkeit des digitalisierten Wertes ist dann maximal durch diese Digitalisierung festgelegt, auch Binning genannt. Kann der FlashADC bespielsweise einen Messbereich von 5 Volt mit 16 Bit digitalisieren, so resultiert alleine aus der Digitalisierung eine Unsicherheit von

$$\Delta \mathrm{V} = \frac{5\,\mathrm{Volt}}{65536\,\mathrm{Bins}} = 76,3\,\mu\mathrm{V}. \tag{3.2}$$

Ein FlashADC tastet das einkommende Signal in einer vorher festgelegten Frequenz ab und speichert diese Werte in einem digitalen Register. Bei dem hier genutzten FlashADC SIS 3301 der Firma Struck kann eine Abtastrate von 100 Mhz (alle 10 ns), 50 Mhz (alle 20 ns) oder 25 Mhz (alle 40 ns) einstellt werden. So wird die Feinheit der Abtastung geändert, was sich allerdings auf die aufgenommene Datenmenge auswirkt. Ist nur die Signalhöhe interessant, reicht es normalerweise, einen ADC zu nutzen, der nur die Signalhöhe misst, einem sogenannten Peak Sensing ADC. Jedoch muss dann sichergestellt werden, dass es nicht zu einer Überlagerung von Signalen (sogenanntes Pile-Up) kommen kann, da mit Pile-Up-Signalhöhen keine Aussage mehr über die Höhe der Einzelsignale getroffen werden kann. Mit Shapern, elektronischen Geräten, die eine Signalform so verändern, dass es nicht zu Überlagerungen

kommen kann, könnte dies wiederum verhindert werden, jedoch lohnt es sich sowohl in finanzieller Hinsicht als auch aus Flexibilitätsgründen beim PANDA-Experiment FlashADCs zu nutzen. Durch Aufnahme des zeitlichen Verlaufs werden Pile-Up-Signale sichtbar, die mit unterschiedlichen Methoden separiert werden können.

Der FlashADC ist in ein VME-Bus-Crate eingebaut und wird über einen PC angesteuert. Der PC und der FlashADC kommunizieren über das VME-Bus-System (Versa Module Eurocard-Bus-System) miteinander. Dieses Bussystem ist für 19"-Einschubgeräte konzipiert. Standardmäßig besitzt es einen 16-Bit Datenbus und einen 24-Bit Adressbus, das Bus-System wurde im Laufe der Jahre jedoch erweitert.

Im aktuellen Aufbau zum Messen von Signalen des Prototyps wird der FlashADC über einen Einschub-PC gesteuert. Auf diesem PC ist Linux (Debian 6) installiert, es kann darauf über eine SSH-Verbindung zugegriffen werden. Des Weiteren ist auf diesem PC ein Treiber für den VME-Bus installiert, so dass es möglich ist, über den VME-Bus auf den FlashADC zuzugreifen.

Um einem FlashADC Befehle zu übermitteln, muss als Erstes der PC mit dem FlashADC verbunden werden. Es gibt zwei Möglichkeiten, die Verbindung herzustellen. Zum einen kann der Adressbereich des FlashADC direkt in den Adressbereich des PC integriert werden; der Nachteil davon ist, dass dem PC ein beträchtlicher Adressbereich verloren geht. Zum anderen ist es möglich, eine virtuelle Adresse zu erstellen, hinter der die Adressen des FlashADC als Subadressen anzusprechen sind. Für diese Arbeit wurde die zweite Variante gewählt. Realisiert wird diese Verbindung über den VME-Bus-Treiber, der mit einer übergebenen physikalischen Adresse den FlashADC in der PCI-Bus integriert und der FlashADC über eine virtuelle Adresse ansteuert wird. Mithilfe einer Adresstabelle für den FlashADC werden Pointer auf die Anfangsbits der Register gesetzt. Die meisten Register haben eine Größe von 32 Bit (long integer), einige aber nur von 16 Bit (integer). Die Größe des Registers ist ebenfalls in der Adresstabelle ersichtlich.

Register sind nach logischen Gruppen zusammengefasst, beispielsweise gibt es ein Register, um grundlegende Funktionen des FlashADC einzustellen. Wiederum ein anderes Register bietet die Möglichkeit, Aufnahmeparameter zu setzen. Außerdem gibt es Register, in welche die Messdaten geschrieben werden. Für die Register existieren verschiedene Wege, sie zu steuern. In einigen Registern können Funktionen über logische Bits (`1` / `0`) aktiviert/deaktiviert werden. Andere Register folgen einer Flipflop-Logik, was bedeutet, dass bestimmte Bits mit einer logischen 1 geschrieben werden müssen, um eine Funktion zu aktivieren, jedoch dann ein anderes Bit mit einer logischen 1 geschrieben werden muss, um diese wieder zu deaktivieren. Um den aktuellen Status (aktiviert/deaktiviert) einer Funktion zu lesen, muss gegebenenfalls wiederum ein anderes Bit oder eine Bitfolge gelesen werden. Außer-

dem gibt es Register, in die beliebige Bits geschrieben werden können, um eine Funktion zu aktivieren. Eine detaillierte Adresstabelle, in der alle Register und Funktionen aufgelistet sind, findet sich in [6].

Um nicht 32-stellige Bitcodes eingeben zu müssen, werden jeweils 4 Bit als eine Hexadezimal-Ziffer eingeben. Zwei Hexadezimal-Ziffern entsprechen folglich einem Byte. Soll ein Byte mit der Bitfolge `11111110` geschrieben werden, so entspricht dies einem `fe` in Hexadecimalcode. Dieser Code ist ein Stellenwertsystem zur Basis 16, besitzt die Ziffern 0-9,a-f und ist sehr verbreitet in der Informatik.

$$1111_{binär} = 2^3 + 2^2 + 2^1 + 2^0 = 15_{dezimal} = f_{hexadezimal}\,. \tag{3.3}$$

Dies erleichtert die Eingaben wesentlich und erhöht die Übersicht. Eine weitere Besonderheit ist das bitweise Verschieben. Dies wird nötig, wenn Werte ausgelesen werden, von denen nur einen Teil benötigt wird. Diese werden bitweise verschoben und anschließend mit einem logischen Operator so verknüpft, dass nur der gewünschte Ausdruck übrig bleibt. Dies ist zum Beispiel nutzvoll beim Auslesen der Messdaten des FlashADC, da im Datenregister gleichzeitig zwei 16-Bit-Werte für Messkanal 1 und Messkanal 2 geschrieben werden, diese aber separiert werden müssen.

Auf Einzelheiten in der Steuerung des FlashADC wird im folgenden Kapitel ausführlich eingegangen.

4. Entwicklung eines Datenaufnahmealgorithmus' für den FlashADC

Um bei dem Prototyp des Kalorimeters Daten aufzunehmen, werden in dieser Bachelorarbeit die Grundlagen dafür gelegt, den FlashADC initial einzurichten und ansprechen zu können, Daten auszulesen, zu speichern und gegebenenfalls sichtbar zu machen. Zusätzlich wurde eine Möglichkeit geschaffen, überlagerte Signale zu separieren.

Abschließend wurden einige Messungen mit dem Versuchsaufbau durchgeführt, um die Charakteristiken des Aufbaus zu ermitteln und den Datenaufnahmealgorithmus zu testen. In den folgenden Unterkapiteln wird auf die genannten Punkte eingegangen.

4.1. Aufbau einer Lichtquelle zur Simulation eines einfallenden Teilchens

Um schon jetzt eine Datenaufnahme des FlashADC durchführen zu können, obwohl der Prototyp zur Zeit der Erstellung dieser Arbeit noch keine Signale detektieren kann, wird eine APD mit angeschlossenem Vorverstärker in eine lichtdichte Box gesetzt. Nun können mithilfe einer lichtemittierenden Diode (LED) Photonen erzeugt und das Signal des Vorverstärkers gemessen werden. Dazu wird ein kurzer Strompuls an eine LED, die Licht im Bereich des sichtbaren blauen Spektrums erzeugt, gegeben. Diese emittiert Photonen zur APD, diese Photonen werden von der APD in elektrische Ladungen umgewandelt und der Vorverstärker erzeugt ein Signal einer Form ähnlich der in Abbildung 3.3. In Abbildung 4.1 ist schematisch die Verschaltung des Aufbaus der Lichtquelle nachzuvollziehen.

Das Ausgangssignal des Signalgenerators wird über einen 50 Ohm-Splitter aufgeteilt. Ein Teil dieses Signals läuft auf direktem Weg zum Trigger des FlashADC. Das andere Signal läuft über eine möglichst lange Kabelverbindung zu einem schnellen LED-Pulser, welcher mit $\pm$ 6 Volt DC Spannung versorgt wird und den Puls des Generators in einem für die LED brauchbaren Puls (mit konstantem Offset) umwandelt. Er arbeitet mit einer Verstärkung von 1:1 und steuert die LED an. Der Signalausgang des Vorverstärkers wird nun an einen Messeingang des FlashADC angeschlossen. Durch die längere Laufzeit des Signals zum LED-Pulser ist garantiert, dass der FlashADC das vollständige Signal aufnehmen kann, da das Triggersignal einige Nanosekunden vor dem Messsignal am FlashADC eintrifft. Die Einstellungen des Signalgenerators für einen einfachen Puls sind in der zweiten Spalte von

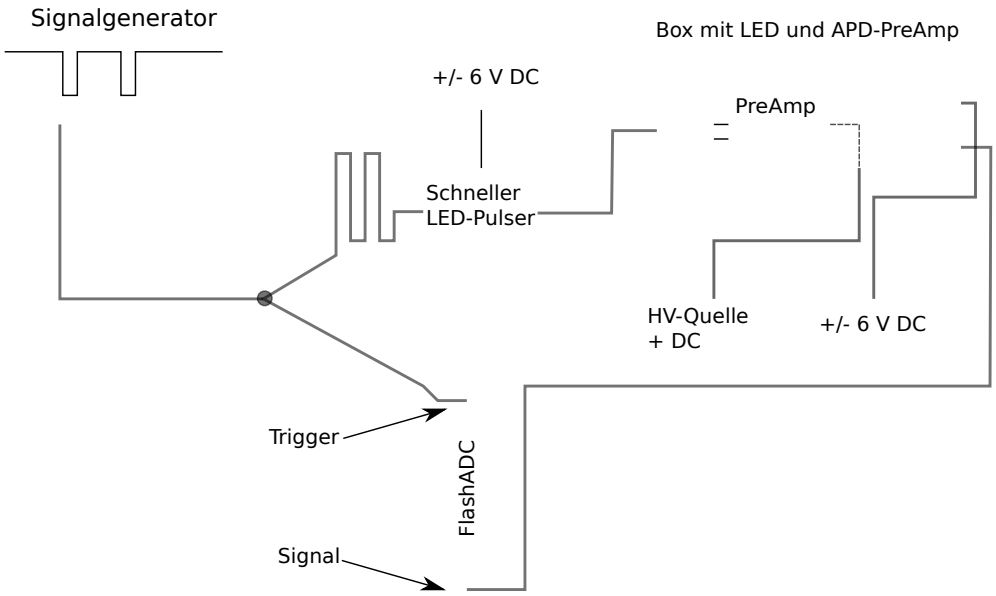


Abbildung 4.1.: Schematische Darstellung des Aufbaus zum Generieren eines Vorverstärker-Signals. Das Signal wird im Signalgenerator erzeugt und über einen 50-Ohm-Splitter aufgeteilt. Dieses Signal steuert mit einem Teil einen schnellen LED-Pulser, der andere Teil wird als Trigger zum Starten der Datenaufnahme am FlashADC verwendet. Das Signal des LED-Pulsers treibt eine LED, die Licht im Bereich des blauen Spektrums emittiert. Diese erzeugten Photonen werden von einer APD wiederum in Ladungen umgewandelt, vom Vorverstärker verstärkt und an einen Messkanal des FlashADCs geleitet.

Tabelle 4.1 einzusehen.

Um die Überlagerung der Signale zweier einfallender Teilchen zu simulieren, kann der Signalgenerator in einem Burst-Modus betrieben werden, in dem er zwei Signale in kurzer Zeit generiert und danach längere Zeit kein weiteres. Die genutzten Parameter für diesen Modus befinden sich in der dritten Spalte von Tabelle 4.1. In Abbildung 4.2 ist die Pulsform noch einmal graphisch dargestellt.

Für ein möglichst realistisches Testen des Signals ist es wichtig abzuschätzen, wie groß ein Signal bei einfallenden Teilchen zu erwarten ist. Da der Detektor für große Teilchenenergien ausgelegt ist, wird deshalb eine Rechnung durchgeführt, in der die Signalhöhe ausgerechnet wird, die ein kosmisches Muon erzeugt, wenn es den Kristall in waagerechter Lage durch-

	Einfachpuls	Doppelpuls
Modus	periodisch	Burst
Periode	100 ms	16 μs Abstand
Spannung	-4,346 V	-4,346 V
Pulsbreite	60 ns	60 ns
Intervall	-	100 ms

Tabelle 4.1.: Genutzte Parameter des Pulsgenerators fürs Erzeugen eines Pulses.

fliegt.

Ein Muon zählt in guter Näherung zu minimal ionisierenden Teilchen. Diese haben nach Tabelle 3.1 einen Energieverlust von 10,1 MeV/cm in Bleiwolframat. Desweiteren besitzt Bleiwolframat (PWO-II) eine Lichtausbeute von 20 Photoelektronen (phe) pro MeV bei einer Kristalltemperatur von +25℃. Sie erhöht sich bei einer Temperatur von -25℃ auf 90 Photoelektronen pro MeV ([1], S.54). Diese Angabe ist nicht sehr hilfreich, da sie an einen bestimmten Photomultiplier mit einer gewissen Quanteneffizienz (hier 18%) geknüpft ist. Jedoch lässt sich in [1] auf Seite 87 eine Anzahl erzeugter Photonen pro MeV ablesen: Bei einer Energieabgabe von 1 MeV eines Teilchens werden 500 Photonen an der Kopffläche des Kristalls bei einer Kristalltemperatur von -25℃ erhalten. Des Weiteren besitzt die APD eine Quanteneffizienz von ca. 70% bei einer Wellenlänge von $420\,\mathrm{nm}$ [3]. Diese Wellenlänge wird von der schnellen Szintillationskomponente des Kristalls erzeugt (siehe Tabelle 3.1). Es muss unter anderem die aktive Fläche berücksichtigt werden. Die APD besitzt eine Fläche von $7\,\mathrm{mm} \times 14\,\mathrm{mm} = 98\,\mathrm{mm}^2$, der Kristall eine Kopffläche von $24\,\mathrm{mm} \times 24\,\mathrm{mm} = 576\,\mathrm{mm}^2$, somit ist die aktive Fläche nur ein Anteil von $\frac{98\,\mathrm{mm}^2}{576\,\mathrm{mm}^2} = 17\%$, was bedeutet, dass nur 17% der Photonen überhaupt in die APD einfallen.

Diese Werte lassen sich nun zusammenziehen:

$$500\,\frac{\mathrm{ph}}{\mathrm{MeV}} \cdot \underbrace{17\%}_{\text{aktive Fläche}} \cdot \underbrace{0,7}_{\text{Quanteneffizienz}} \approx 60\,\frac{\mathrm{ph}}{\mathrm{MeV}}\,. \tag{4.1}$$

Wird davon ausgegangen, dass jedes Photon ein Elektron auslöst und eine Verstärkung von $G = 50$ durch den Avalanche-Effekt (abhängig von der Sperrspannung) erzeugt wird, so resultieren daraus $2975\,\frac{\mathrm{phe}}{\mathrm{MeV}}$. Kosmische Muonen hinterlassen in dem Kristall bei senkrechtem Einfall in einen waagerecht liegenden Kristall $2,4\,\mathrm{cm} \cdot 10,1\,\frac{\mathrm{MeV}}{\mathrm{cm}} = 24,2\,\mathrm{MeV}$ Energie.

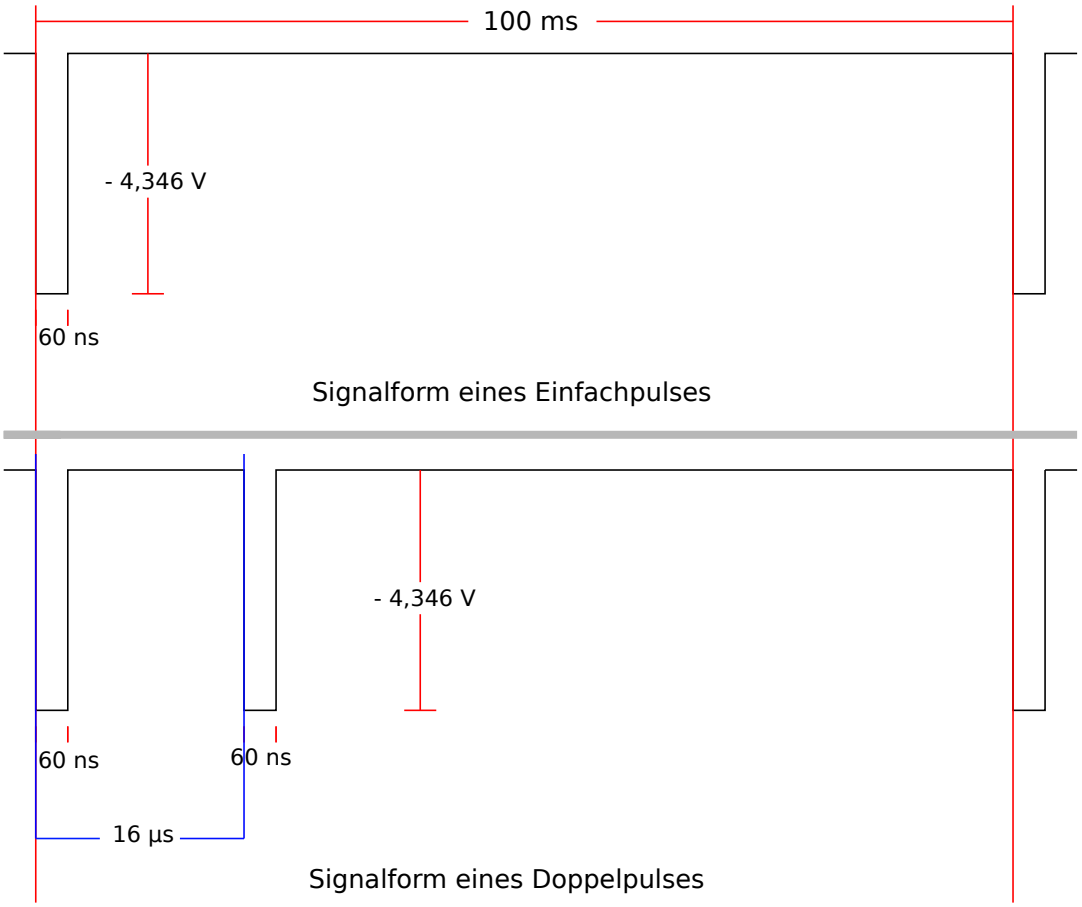


Abbildung 4.2.: Genutzte Pulsformen des Signalgenerators zum Ansteuern einer LED (LED-Pulser zwischengeschaltet).

Daraus resultieren folgende Ergebnisse bei einer Kristalltemperatur von -25℃:

$$2975\,\frac{\text{phe}}{\text{MeV}} \cdot 24,2\,\text{MeV} = 15730\,\text{phe}\,. \tag{4.2}$$

(4.3)

Dies entspricht einer Ladung von

$$15730\,\text{phe} \cdot 1,6021 \cdot 10^{-19}\,\frac{\text{C}}{\text{phe}} = 11,53 \cdot 10^{-15}\,\text{C} = 11,53\,\text{fC}\,. \tag{4.4}$$

Der Vorverstärker besitzt eine Ladungsempfindlichkeit von $\frac{0,5\,\text{V}}{\text{pC}}$ (vgl. [4], S. 4).Damit ergibt sich ein Ausgabesignal von

$$\frac{0,5\,\text{V}}{10^{-12}\,\text{C}} \cdot 11,53 \cdot 10^{-15}\text{C} = 5,76\,\text{mV}\,. \tag{4.5}$$

Wird davon ausgegangen, dass ein Muon längs durch den Kristall (200 mm Weglänge) fliegt, so ergibt sich eine Spannung von

$$2975 \frac{\text{phe}}{\text{MeV}} \cdot 202\,\text{MeV} = 600950\,\text{phe} \tag{4.6}$$

$$600950\,\text{phe} \cdot 1,6021 \cdot 10^{-19} \frac{\text{C}}{\text{phe}} = 96,28 \cdot 10^{-15}\,\text{C} \tag{4.7}$$

$$\frac{0,5\,V}{10^{-12}\,\text{C}} \cdot 96,28 \cdot 10^{-15}\text{C} = 48,14\,\text{mV}\,. \tag{4.8}$$

Dies stellt jedoch einen sehr langen Weg durch den Kristall dar und ist aufgrund des extremen Einfallwinkels ($\approx 0°$) für ein Muon relativ unwahrscheinlich. Die realen Weglängen werden sich im Bereich zwischen diesen beiden gerechneten Werten befinden.

Mit dem Wissen dieser Größen wurde der Signalgenerator auf eine Spannung eingestellt, so dass er ein Signal mit einer Höhe von ca. 25-40 mV im Vorverstärker erzeugt. Dies ist ein gutes Signal-Rausch-Verhältnis, um die Daten analysieren zu können, andererseits liegt die Spannung in ähnlicher Größenordnung wie bei realen Teilchen. Da die LED eine Schwellspannung besitzt und selbst mit hundertstel Volt Spannungsänderung große Intensitätssprünge verursacht, wurde auch der Verstärkungsfaktor der APD angepasst. In Tabelle 4.2 sind die gewählten Parameter zusammengefasst.

APD #	0A0331
PreAMP #	SP883a02
angelegte Hochspannung (HV)	(340 ± 1) V
Verstärkungsfaktor	$\approx$ 55 (bei +25℃)

Tabelle 4.2.: Parameter der verwendeten APD und des genutzten Vorverstärkers

4.2. Erstellung eines Programms zur Messdatenaufnahme mithilfe eines FlashADCs

Um den FlashADC steuern und auslesen zu können, wurde ein Programm in der Sprache C++ geschrieben. Für die Visualisierung und die Speicherung der Daten wurde das ROOT-Framework genutzt. Die genutzte Klasse „VMEBridge“ war bereits vorhanden und wurde unverändert zur Nutzung übernommen. Ebenso bestand bereits das Programm „CMake“ zur Verfügung, welches Verlinkung und Kompilierung delegiert. Dieses wurde entsprechend den Bedürfnissen angepasst. Das Datenaufnahmeprogramm ist in objektorientierter Weise aufgebaut und kann so in Zukunft für weitere FlashADCs und weitere Messeingänge beliebig

skaliert werden.Derzeitig kann Messkanal 1 ausgelesen werden. Das Programm besteht aus 3 Klassen, genannt

- FlashADC: Klasse zum Ansteuern des FlashADC
- DAQ: Klasse zum Aufnehmen, Visualisieren und Speichern von Daten
- Proto8: Hauptklasse zum Steuern der Datenaufnahme

Die Programmstruktur ist in Abbildung 4.3 noch einmal als UML-Diagramm dargestellt. Im Folgenden soll kurz auf die Funktionen der einzelnen Klassen eingegangen werden, um diese ohne Schwierigkeiten in Zukunft nutzen und verändern zu können.

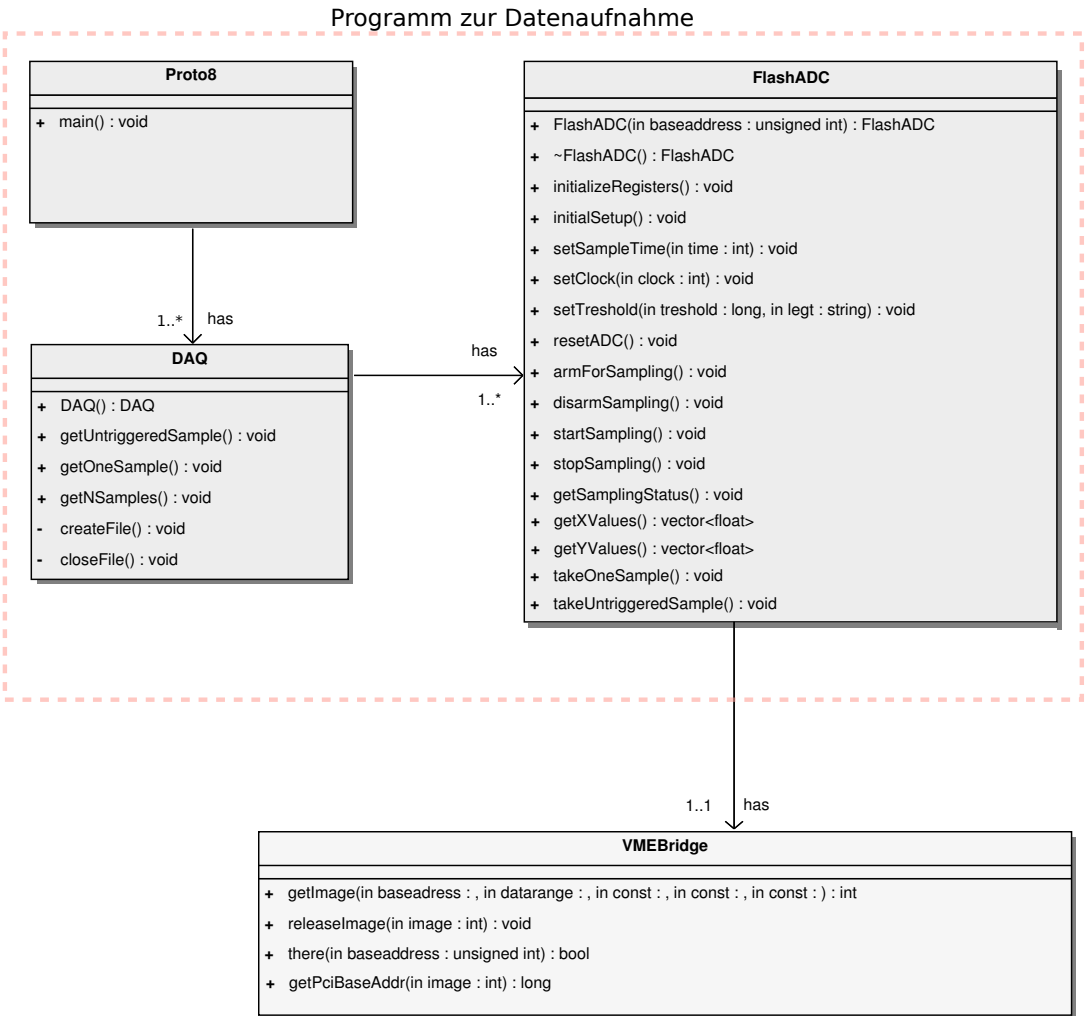


Abbildung 4.3.: UML-Diagramm des Datenaufnahmeprogramms. Die Klasse VMEBridge war bereits vorhanden und wurde ohne Änderungen verwendet.

4.2.1. Die Funktionen der Klasse FlashADC

- **FlashADC(unsigned int baseaddress)**: Dies ist der Konstruktor der Klasse FlashADC. Ihm wird die physikalische Adresse des FlashADC übergeben. Er erstellt ein Objekt der Klasse VMEBridge und bekommt mithilfe der Funktion VMEBridge::getImage() eine virtuelle Adresse für den FlashADC zurück. Diese wird für zukünftige Befehle genutzt.

- $\sim$ **FlashADC**: Der Destruktor der Klasse FlashADC sorgt dafür, dass der allokierte Speicherbereich des FlashADCs freigegeben und das Objekt der Klasse VMEBridge gelöscht wird.

- **initializeRegisters()**: Diese Funktion holt sich über das erzeugte Objekt der Klasse VMEBridge die virtuelle Adresse des FlashADC. Danach werden Pointer auf den Anfang aller später benötigten Register gesetzt. Dies soll an einem Beispiel erläutert werden: Die Adresse des Kontroll-Registers ist in [6] (S.16) zu erhalten und wird nun als Offset zu der virtuellen Adresse des FlashADC addiert.

  ```
  ph_FlashADC= vme->getPciBaseAddress(image);
  pl_control_reg =
  (long*) ((long)ph_FlashADC + (long)SIS3300_CONTROL_STATUS);
  ```

  Der Inhalt des Zeigers kann nun gelesen und geschrieben werden. Der Typ `long*` entspricht einer 32-Bit-Integer-Zahl und muss an die Größe des genutzten Registers angepasst werden. Ob das Register Schreib- und Lesezugriff hat oder nur eine Möglichkeit von beiden, ist aus [6] ersichtlich.

- **initialSetup()**: Setzt initiale Parameter, die für die derzeitige Nutzung wichtig sind. Der externe Trigger wird aktiviert, die Kontroll-LED angeschaltet, die interne Uhr zurückgesetzt und die interne Taktfrequenz zur Kontrolle am Trigger-Ausgang ausgegeben.

- **setSampleTime(int time)**: Errechnet aus der derzeitigen Taktrate und der gewünschten Messzeit die Anzahl der aufzunehmenden Datenwerte.

- **setClock(int clock)**: Setzt die interne Taktfrequenz der Datenaufnahme in Mhz. Mögliche Werte sind 100, 50 und 25.

- **setTreshold(long treshold, string legt)**: Diese Funktion legt den Schwellenwert für eine selbstgetriggerte Datenaufnahme fest. Es wird ein Wert im Bereich des ADC-Ausgabewerts übergeben und ob auf steigende (gt) oder fallende Flanke (le) getriggert werden soll.

- **resetADC()**: Zurücksetzen des FlashADC.
- **armForSampling()**: Aktiviert die Datenaufnahme des FlashADC. Mit dem folgenden Signal am Triggereingang startet der FlashADC die Datenaufnahme.
- **disarmSampling()**: Deaktiviert die Datenaufnahme des FlashADC.
- **startSampling()**: Startet die Datenaufnahme des FlashADC manuell.
- **stopSampling()**: Stoppt die Datenaufnahme.
- **getSamplingStatus()**: Gibt aus, ob der FlashADC im Moment Daten aufnimmt.
- **getXValues()**: Gibt die x-Werte der Messungen als n-dimensionalen Vektor aus.
- **getYValues()**: Gibt die y-Werte der Messungen als n-dimensionalen Vektor aus.
- **takeOneSample()**: Nimmt Datenwerte für eine bestimmte Zeit auf und schreibt sie in den x- und y-Vektor. Die Datenaufnahme wird durch ein externes Trigger-Signal gestartet.
- **takeUntriggeredSample()**: Funktioniert wie takeOneSample(), jedoch ohne einen Trigger. Somit ist diese Funktion beispielsweise zur Aufnahme von Rauschen geeignet.

4.2.2. Die Funktionen der Klasse DAQ

- **DAQ()**: Der Konstruktor der Klasse DAQ erzeugt ein Objekt der Klasse FlashADC, initialisiert die Register, setzt den FlashADC zurück und stellt das initiale Setup ein. Danach fragt dieser die gewünschte Abtastrate und die zu messende Zeit ab und übergibt sie an den FlashADC.
- **getUntriggeredSample()**: Nimmt ungetriggert Daten für eine vorher festgelegte Zeit auf und speichert sie als Histogramm und zusätzlich als zwei Vektoren in einer ROOT-Datei ab.
- **getOneSample()**: Nimmt eine Messung auf, die über einen externen Trigger gestartet wird. Gemessen wird für eine vorher festgelegte Zeit, danach wird die Messung als Histogramm und zusätzlich als zwei Vektoren in einer ROOT-Datei gespeichert.
- **getNSamples()**: Funktioniert wie getOneSample() mit dem Unterschied, dass mehrere Samples aufgenommen werden. Der Nutzer muss eine Anzahl an Messungen eingeben und festlegen, ob jede der Messungen als Histogramm und Vektoren oder lediglich als

zwei Vektoren gespeichert werden soll. Bei einer größeren Anzahl an Messungen (>10) ist es sinnvoll, keine Histogramme erstellen zu lassen, um Speicherplatz zu sparen und den PC nicht zu überlasten.

- **createFile()**: Legt eine ROOT-Datei an. Dabei wird die Datei in einen Unterordner names „rootfiles“ abgelegt. Jede Datei hat den Namen "Measurement-YYYY-MM-DD_hh-mm-ss.root".
- **closeFile()**: Schließt die ROOT-Datei und löscht das Objekt der Klasse FlashADC.

4.2.3. Die Funktion der Klasse Proto8

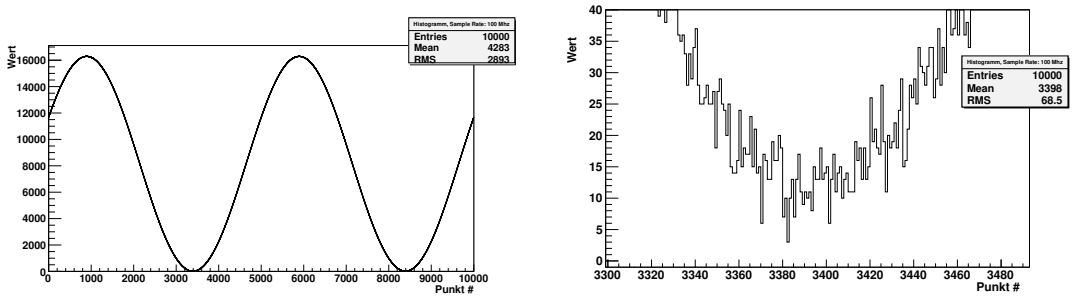
Diese Klasse enthält die main()-Funktion und steuert somit die Datenaufnahme. Hier wird eine Kommandozeilenauswahl realisiert, über die der Nutzer entscheiden kann, welche Art der Datenaufnahme er nutzen möchte.

4.3. Bestimmen des Messbereichs des FlashADC

Eine wichtige Größe zum späteren Erstellen von Histogrammen ist die Auflösung des FlashADC. Diese lässt sich mithilfe von Potentiometern und Jumpern einstellen. Da es für diesen ersten Schritt der Datenaufnahme nicht wichtig ist, einen speziellen Messbereich festzulegen, besteht die Aufgabe lediglich darin, den eingestellten Messbereich durch Testmessungen zu ermitteln. Dazu wird ein sinusförmiges Signal an den FlashADC angelegt und das einkommende Signal in kleinen Schritten soweit erhöht, bis Sättigung eintritt. Somit wird der maximale Messbereich ermittelt. Hierbei wurde unter anderem festgestellt, dass der FlashADC nur einen Teil der möglichen 65535 Werte nutzt, was bedeutet, dass er durch Limitieren des Messbereichs nicht genauer misst, sondern nur einen Teil des Wertebereichs nutzt (siehe [6]).

Es wurden Daten über einen Zeitbereich mehrerer Perioden eines Sinussignals (siehe Abbildung 4.4 a) aufgenommen, um sicherzustellen, dass mindestens ein Minimum und ein Maximum gemessen werden. Diese werden bis zur sichtbaren Sättigung um 0,001 V erhöht. Danach wird zur Analyse mithilfe von eines ROOT-Makros für jede Messung ein Maximum-zu-Minimum-Wert errechnet. Angegeben werden hier nur der höchste nichtgesättigte und der niedrigste gesättigte Wert, die Differenz entspricht dem Fehler dieser Messung.

Somit ergibt sich ein Messbereich von $2,143 \pm 0,001$ V, was in Ausgabewerten einem Bereich von 16298 ± 3 entspricht. Für spätere Histogramme wurde ein Messbereich von 16300 ± 5 angenommen. Die mangelnde Statistik rechtfertigt hier, einen größeren Fehler



(a) Verwendetes Signal zur Bestimmung des Messbereichs.

(b) Detail-Ansicht des Signals aus (a) nahe der Null.

Abbildung 4.4.: Verwendetes Sinus-Signal zum Bestimmen des Messbereichs des FlashADC. Durch die Periode von 50 Mikrosekunden wird garantiert, dass jeweils ein Minimum und ein Maximum gemessen werden können. Im rechten Bild ist zu sehen, wie klein der Abstand zwischen Signal und Nullpunkt ist und sich das Signal somit an der Grenze zur Sättigung befindet (Signalhöhe hier $\pm$ 1,071 V).

anzunehmen. Wenn final ein gewünschter Messbereich festgelegt ist, wird es nötig sein, diesen noch einmal mit einer größeren Anzahl an Messungen genaustens zu ermitteln.

Wird von einem Messbereich von 16300 ± 5 für eine Spannung von $(2,143 \pm 0,001)\,\mathrm{V}$ ausgegangen, so ergibt sich für einen Ausgabewert eine Spannungsdifferenz δ_k von

$$\delta_k = \frac{2,143\,\mathrm{V}}{16300\,\text{Ausgabewerte}} = 0,1315\,\frac{\mathrm{mV}}{\text{Ausgabewert}} \tag{4.9}$$

$$\Delta\delta_k = \left(\frac{0,001\,\mathrm{V}}{2,143\,\mathrm{V}} + \frac{5\,\text{Ausgabewerte}}{16300\,\text{Kanäle}}\right) \cdot 0,13\,\frac{\mathrm{mV}}{\text{Wert}} = 0,0001\ \frac{\mathrm{mV}}{\text{Wert}} \tag{4.10}$$

$$\delta_k = (0,1315 \pm 0,0001)\,\frac{\mathrm{mV}}{\text{Ausgabewert}} = (131,5 \pm 0,1)\,\frac{\mu\mathrm{V}}{\text{Ausgabewert}}\,. \tag{4.11}$$

Dies kann auch in Photoelektronen oder in MeV Energieverlust im Kristall für festgelegte Parameter (Kristalltemperatur, Verstärkungsfaktor der APD) gegeben werden. Dazu wird im folgenden die Abschätzung aus Kapitel 4.1 genutzt. Der Kristall ist auf -25℃ gekühlt, die APD wird bei einem Verstärkungsfaktor von 50 betrieben. Die aktive Fläche der APD an der Kopffläche beträgt 17%, die Quanteneffizienz der APD bei einer Wellenlänge von 420 nm 70%. Dann ergibt sich ein Energieverlust im Kristall in keV pro mV Verstärkerspannung oder in Photoelektronen (phe) pro mV Verstärkerspannung von

$$\frac{24,2\,\mathrm{MeV}}{5,76\,\mathrm{mV}} = 4,17\,\frac{\mathrm{MeV}}{\mathrm{mV}} = 12406\,\frac{\mathrm{phe}}{\mathrm{mV}}\,. \tag{4.12}$$

	Signalhöhe Peak zu Peak [V]	Ausgabewert
höchster nichtgesättigter Wert	2,142	16296
niedrigster gesättigter Wert	2,144	16301

Tabelle 4.3.: Bestimmung des Messbereichs des FlashADCs mithilfe eines sinusförmigen Signals. Dieses wurde mit dem FlashADC gemessen und in kleinen Schritten erhöht, bis Sättigung beim FlashADC eintritt und somit der maximale Messbereich erreicht ist. Angegeben sind nur der höchste nichtsättigende und der niedrigste sättigende Wert.

Es ergibt sich eine Unsicherheit durch Binning von

$$\delta_k = (131,5 \pm 0,1)\,\frac{\mu\mathrm{V}}{\text{Ausgabewert}} \tag{4.13}$$

$$= (548,4 \pm 0,4)\,\frac{\mathrm{keV}}{\text{Ausgabewert}} \tag{4.14}$$

$$= (1631 \pm 1)\,\frac{\mathrm{phe}}{\text{Ausgabewert}}\,. \tag{4.15}$$

Diese Werte werden im Folgenden verwendet, um das Schwanken des FlashADCs in Energieeinheiten- beziehungsweise Ladungseinheiten anzugeben. Da diese Schwankungen mindestens einen Ausgabewert betragen und die Energiewerte nur auf einer Schätzung beruhen, werden diese Größen im Folgenden ohne Fehler angegeben und gerundet auf

$$\delta_k = 0,5\,\frac{\mathrm{MeV}}{\text{Ausgabewert}} \tag{4.16}$$

$$= 1500\,\frac{\mathrm{phe}}{\text{Ausgabewert}}\,. \tag{4.17}$$

Die Genauigkeit des FlashADC ist im Moment um ca. den Faktor 7 kleiner als das Rauschen der Messeinrichtung (Vorverstärker mit APD), wenn ein Rauschen von ± 1 mV angenommen wird, das dem derzeit kleinstmöglichen erreichbaren Wert entpricht.

4.4. Charakteristisches Verhalten des Testaufbaus

In diesem Unterkapitel werden das Rauschen des Testaufbaus sowie die Nulllinie des FlashADC ermittelt. Dazu wurde nach und nach je eine weitere Komponente an den Aufbau angeschlossen und Messdaten aufgenommen. Diese werden anschließend mithilfe von ROOT visualisiert und analysiert.

4.4.1. Das interne Rauschen des FlashADC

Um das interne Rauschen aufzunehmen, wurden mit einer Frequenz von 25 Mhz 12500 Datenpunkte (Aufnahmezeit $500\,\mu$s) aufgenommen (Abbildung 4.5) und diese in ein Histogramm eingetragen. Dieses ist in Abbildung 4.6 abgebildet. Dort wird deutlich, dass das Rauschen einer Normalverteilung ähnelt. Der Mittelwert B_1 des Signals liegt bei 8667 und entspricht der Nulllinie des FlashADC, da kein Signal am Messeingang anliegt. Der Fehler wird berechnet zu

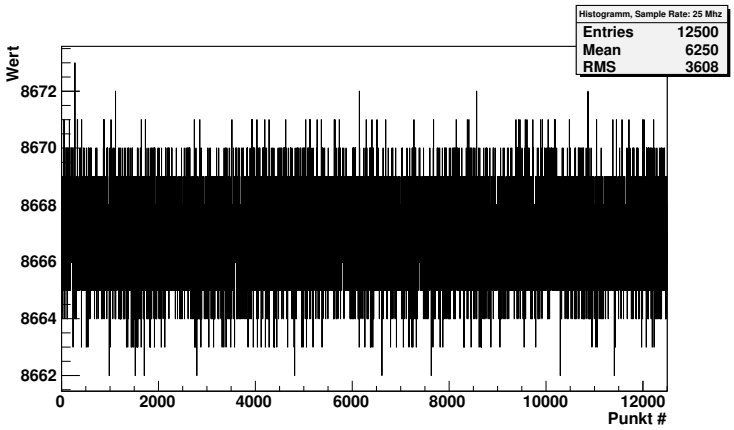


Abbildung 4.5.: Signal des inneren Rauschens des FlashADCs (kein Signal angelegt). Aufnahmefrequenz und -dauer: 25 Mhz über einen Zeitraum von 500 μs.

$$\Delta B_1 = \frac{\sigma}{\sqrt{n}}\,. \tag{4.18}$$

Der Parameter n entspricht der Anzahl der Messwerten, σ ist der Parameter für die Streuung um den Mittelwert bei einer Normalverteilung. Somit ergibt sich für das interne Rauschen ein Wert von

$$\Delta B_1 = 0,01 \tag{4.19}$$

$$B_1 = 8667 \pm 1\,. \tag{4.20}$$

Der Fehler wird aufgerundet, da die minimal zu messende Schwankung einem Wert entspricht. Somit ergibt sich bei einem auf -25℃ gekühlten Kristall und einem Verstärkungsfaktor von G = 50 der APD eine Energieunsicherheit des Mittelwerts von

$$\Delta B_1 = 1500\,\text{phe} = 0,5\,\text{MeV} \tag{4.21}$$

$$\tag{4.22}$$

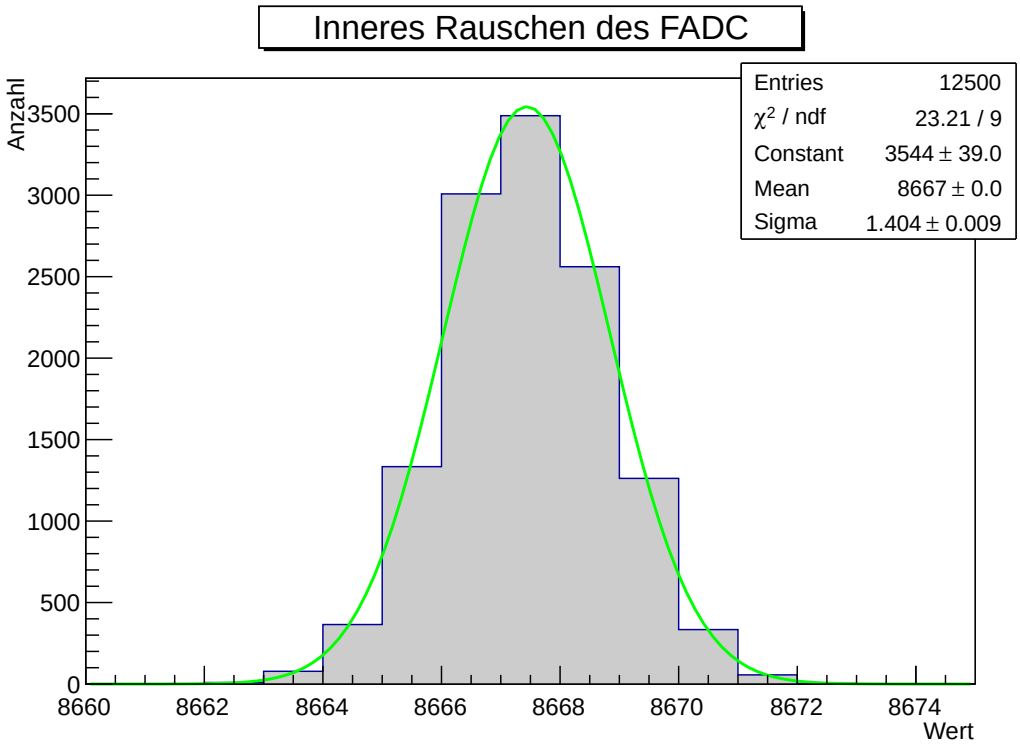


Abbildung 4.6.: Histogramm des inneren Rauschen des FlashADC (Abb. 4.5). Die Schwankung ΔB_1 um die Nulllinie beträgt einen Ausgabewert, das entspricht bei einem auf -25℃ gekühlten Kristall und einer APD mit Verstärkungsfaktor von 50 einer Ladungsschwankung von ca. 1500 Photoelektronen bzw. einer Energieschwankung von ca. 0,5 MeV.

4.4.2. Rauschen des Vorverstärkers, ohne Hochspannung an der APD

Der nächste Schritt besteht darin, den Aufbau wie in Kapitel 4.1 beschrieben an den FlashADC anzuschließen und die Versorgungsspannung von $\pm$ 6 V anzuschalten. Die Hochspannung ist derzeit noch nicht angeschlossen. Nun werden wiederum 12500 Messwerte mit einer Frequenz von 25 Mhz aufgenommen. In Abbildung 4.8 ist deutlich zu sehen, dass sich das Rauschen vergrößert hat. Außerdem hat sich die Nulllinie des Rauschens um 51 Ausgabewerte nach unten verschoben. Es ergibt sich ein Mittelwert von

$$\Delta B_2 = 0,18 \tag{4.23}$$

$$B_2 = 8616 \pm 1\,. \tag{4.24}$$

Somit ergibt sich bei einem auf -25℃ gekühlten Kristall und einem Verstärkungsfaktor von G = 50 der APD eine Energieunsicherheit des Mittelwerts von

$$\Delta B_2 = 1500\,\text{phe} = 0,5\,\text{MeV} \tag{4.25}$$

(4.26)

Die Kurve hat sich stark verbreitert, σ hat sich ca. um den Faktor 14 erhöht und somit entspricht σ einer Energieschwankung von 10,14 MeV bei einem Kristall bei -25℃ und einer APD mit Verstärkungsfaktor 50. Es ist zu sehen, dass das Rauschen des FlashADC außerdem eine gezackte Art besitzt, was für ein Rausch-Signal mit Überlagerungen einiger definierter Frequenzen spricht.

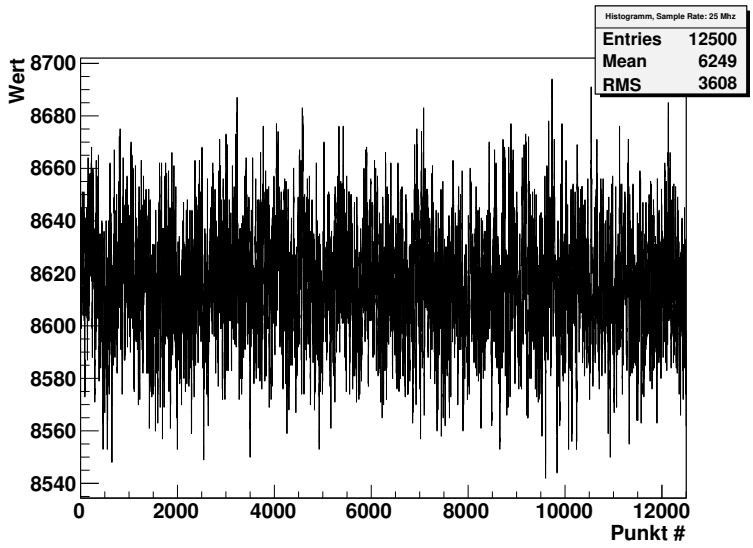


Abbildung 4.7.: Signal des Vorverstärkers mit angeschlossener APD, jedoch ohne angelegte Hochspannung an der APD. Aufnahmefrequenz und -dauer: 25 Mhz über einen Zeitraum von 500 μs.

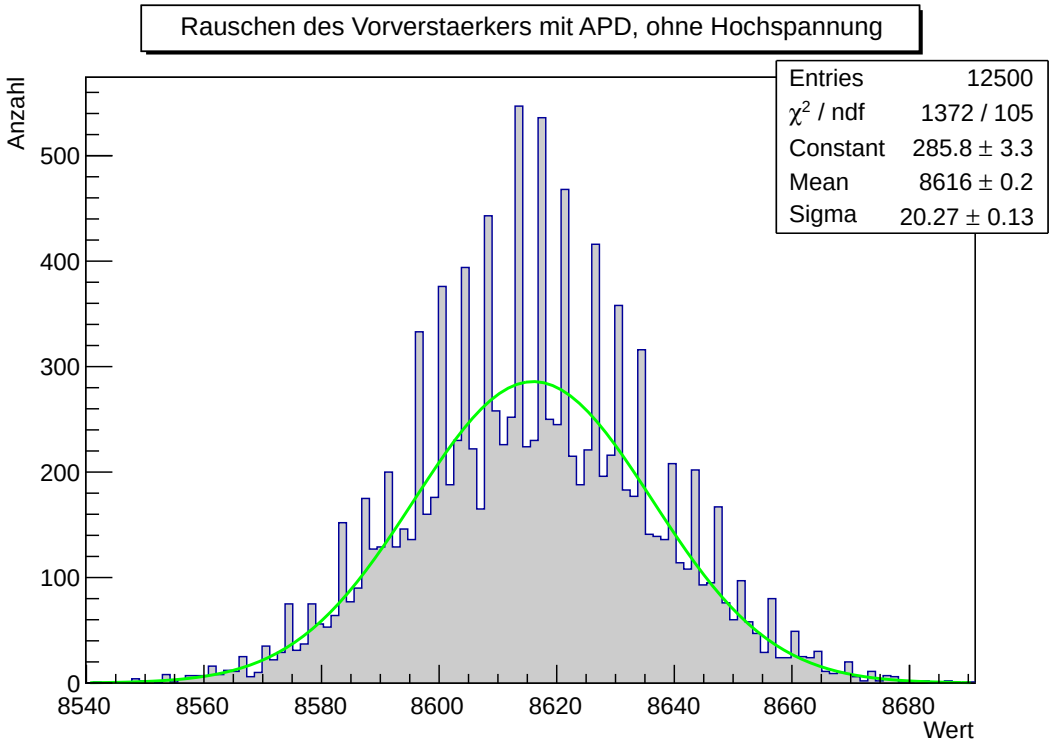


Abbildung 4.8.: Histogramm vom Signal (Abb. 4.7) des Vorverstärkers mit angeschlossener APD, jedoch ohne angelegte Hochspannung. Die Schwankung ΔB_2 um den Mittelwert beträgt einen Ausgabewert, das entspricht bei einem auf -25℃ gekühlten Kristall und einer APD mit Verstärkungsfaktor von 50 einer Ladungsschwankung von ca. 1500 Photoelektronen bzw. einer Energieschwankung von ca. 0,5 MeV.

4.4.3. Rauschen des Vorverstärker, mit Hochspannung an der APD

Als weiterer Schritt wird der Aufbau nun zusätzlich mit Hochspannung versorgt. Danach wird erneut eine Messung wie in Kapitel 4.4.2 durchgeführt. In Abbildung 4.10 ist diese Messung dargestellt. Hier wird deutlich, dass der Vorverstärker offensichtlich nur mit applizierter Hochspannung optimal austariert ist. Das Rauschen hat nun eine Größe von 9 Kanälen, was einem Rauschen von ca. 1 mV entspricht. Dies ist derzeit der minimal zu erreichende Wert.

Außerdem ist zu sehen, dass die Nulllinie sich wieder bis auf einen Wert genau an der internen Nulllinie des FlashADC befindet. Dies zeigt, dass der Vorverstärker mit Versorgungsspannung, jedoch ohne Hochspannung ein negatives Offset besitzt, was ein weiterer Indiz für ein nicht austariertes Messinstrument ist, da es bei der Messung mit applizierter

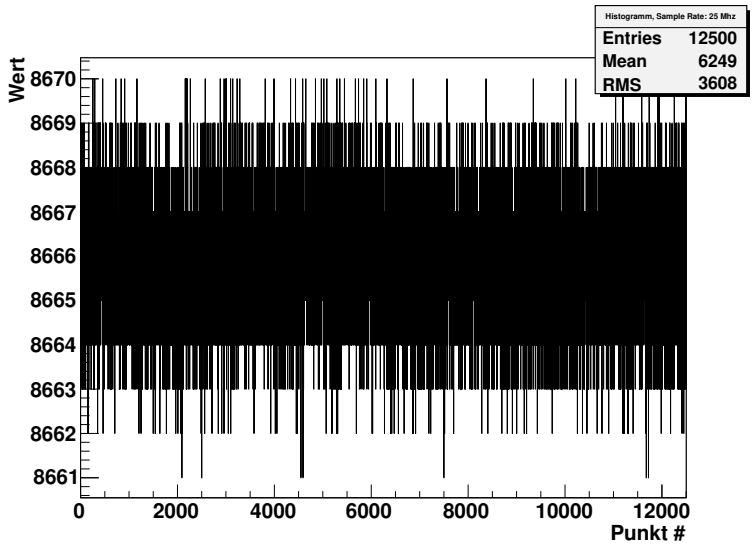


Abbildung 4.9.: Signal des Vorverstärkers mit angeschlossener APD, mit angelegter Hochspannung an der APD. Aufnahmefrequenz und -dauer: 25 Mhz über einen Zeitraum von 500 μs.

Hochspannung wieder verschwindet. Dies scheint verständlich, da der Vorverstärker speziell für die Nutzung mit einer Avalanche-Photodiode entwickelt wurde. Es ergibt sich für den Mittelwert:

$$\Delta B_3 = 0,01 \tag{4.27}$$

$$B_3 = 8666 \pm 1\,. \tag{4.28}$$

Es ergibt sich bei einem auf -25℃ gekühlten Kristall und einem Verstärkungsfaktor von G = 50 der APD eine Energieunsicherheit des Mittelwerts von

$$\Delta B_3 = 1500\,\text{phe} = 0,5\,\text{MeV}\,. \tag{4.29}$$

$$\tag{4.30}$$

Abschließend sind die ermittelten Parameter noch einmal in Tabelle 4.4 dargestellt.

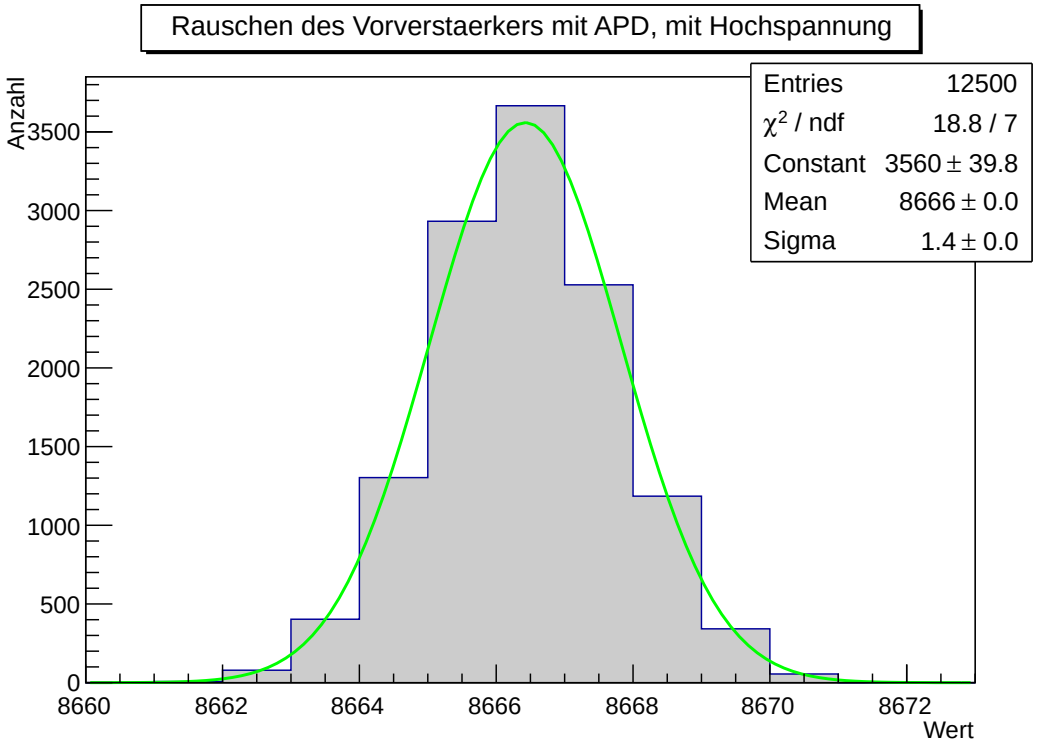


Abbildung 4.10.: Histogramm vom Signal (Abb. 4.9) des Vorverstärkers mit angeschlossener APD und mit angelegter Hochspannung. Aufnahmefrequenz und -dauer: 25 Mhz über einen Zeitraum von 500 μs. Die Schwankung ΔB_3 um den Mittelwert beträgt einen Ausgabewert, das entspricht bei einem auf -25℃ gekühlten Kristall und einer APD mit Verstärkungsfaktor von 50 einer Ladungsschwankung von ca. 1500 Photoelektronen bzw. einer Energieschwankung von ca. 0,5 MeV.

Messung	B		in Ausgabewerten	in MeV	in phe	in μV
Inneres Rauschen	8667	ΔB_1	1	0,5	1500	$(131,5 \pm 0,1)$
des FlashADC		σ_1	1,4	0,7	2100	$(184,1 \pm 0,7)$
Rauschen mit	8616	ΔB_2	1	0,5	1500	$(131,5 \pm 0,1)$
APD, ohne HV		σ_2	20,27	10,14	30405	$(2665,5 \pm 2,0)$
Rauschen mit	8666	ΔB_3	1	0,5	1500	$(131,5 \pm 0,1)$
APD, mit HV		σ_3	1,4	0,7	2100	$(184,1 \pm 0,7)$

Tabelle 4.4.: Zusammenfassung der ermittelten Werte bei den Messungen des inneren Rauschens des FlashADCs ohne angeschlossenes Signal, mit angeschlossenem Vorverstärker und APD, jedoch ohne Hochspannung an der APD sowie mit angeschlossenem Vorverstärker, mit Hochspannung an der APD. Für die Angabe von MeV und Photoelektronen (phe) wurde angenommen, dass ein Kristall bei -25℃ und eine APD mit einer Verstärkung von 50 genutzt werden.

4.4.4. Verteilung von mehreren Signalen

Die Energie eines einfallenden Teilchens ist proportional zu der Spannung, die am Vorverstärker zu messen ist. Um die Schwankungen von Messungen identischer Signale zu messen, wurde die LED mit Pulsen konstanter Höhe angesteuert und 100 LED-Pulse gemessen. Einer der aufgenommenen Pulse ist in Abbildung 4.11 dargestellt. Anschließend wurde mithilfe eines ROOT-Makros das Spannungsmaximum jeder Messung ermittelt und histogrammiert. Die Verteilung der Pulse ist in Abbildung 4.12 abgebildet. Es wird deutlich, dass keinesfalls konstant das gleiche Signal gemessen wird, sondern eine Schwankung von 90 Ausgabewerten existiert. Die Standardabweichung beträgt 18,15 Ausgabewerte, was einer Energieunsicherheit von ca. 9 MeV oder ca. 27200 Photoelektronen entspricht, wenn von einem Aufbau mit einem Kristall bei -25℃ und einer APD mit Verstärkungsfaktor 50 ausgegangen wird. Ein Teil der Schwankung wird vom Rauschen verursacht, doch liegt hier eine zusätzliche Schwankung der LED vor. Diese ist nicht in der Lage, eine scharf definierte Menge an Photonen zu emittieren, der Vorverstärker jedoch ist äußerst sensibel und kann diese Schwankungen detektieren.

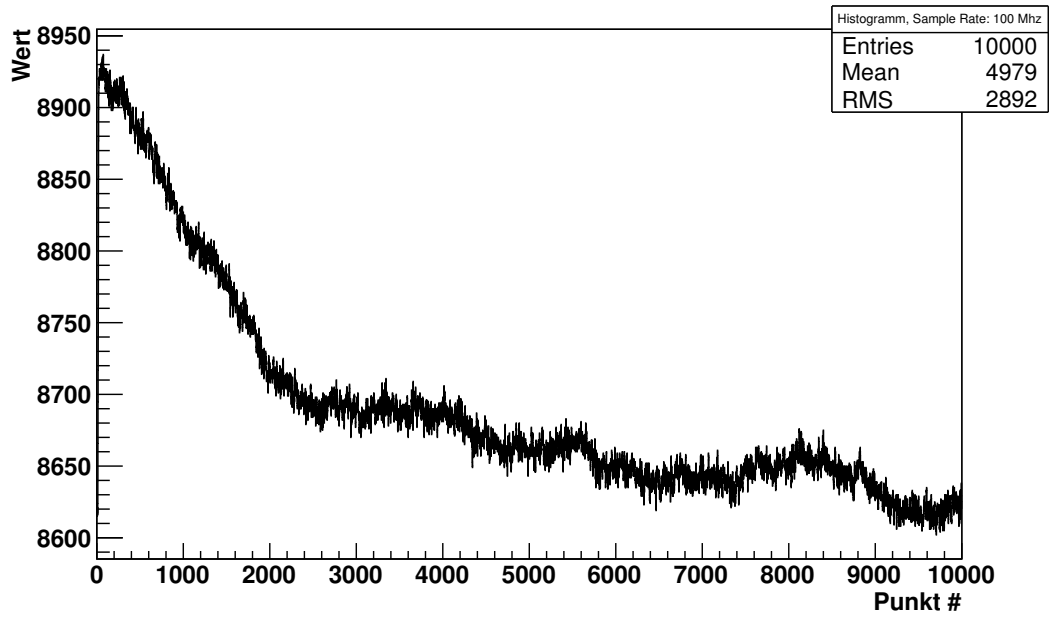


Abbildung 4.11.: Signal des Vorverstärkers bei konstantem LED-Eingangssignal (Einfachpuls). Aufnahmefrequenz und -dauer: 100 Mhz über einen Zeitraum von 100 μs. Es liegen 340 Volt Spannung an der APD an, das entspricht einem Verstärkungfaktor der APD von ca. 55.

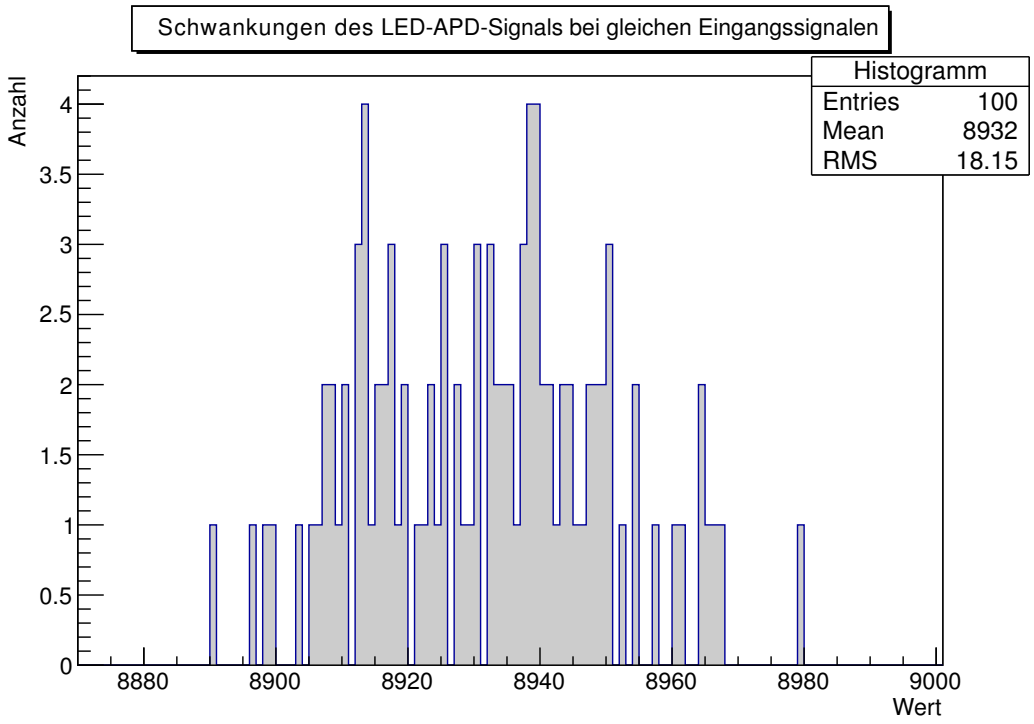


Abbildung 4.12.: Histogramm der Maximas von Signalen des Vorverstärkers bei konstantem Eingangssignal (Einfachpuls). Die Signalform ist in Abbildung 4.11 abgebildet. Die Standardabweichung entspricht einer Energieschwankung von ca. 9 MeV oder einer Ladungsschwankung von ca. 27200 phe, wenn von einem Kristall bei -25℃ und einer APD-Verstärkung von 50 ausgegangen wird. Verursacht wird diese Schwankung durch eine nicht konstante Photonenemission der LED.

4.5. Separieren eines Pile-Up-Signals

Beim PANDA-Experiment wird mit einer hohen Luminosität gearbeitet, woraus eine hohe Eventrate folgt. Um diese Masse an Daten fehlerfrei aufnehmen zu können, muss die Elektronik dementsprechend schnell arbeiten. Dennoch benötigen die Messinstrumente eine gewisse Zeit, um bereit für die nächste Messung zu sein. Der langsamste Teil dieser Elektronik ist der Vorverstärker. Die Notwendigkeit eines Separations-Algorithmus' im finalen Aufbau soll anhand einer Abschätzung deutlich werden. Ausgewählte Parameter sind dabei das Signal des Vorverstärkers, welches mit 30 mV angenommen wird (vgl. mit Kapitel 4.1) und eine Eventrate pro Kristall von 1 kHz ([1], S.128). Die Abklingkonstante τ des Vorverstärkers ist in [4] zu finden, sie beträgt 25 μs. Nun kann die Zeit berechnet werden, die ein Signal

braucht, um von 30 mV auf 1 mV abzufallen. Dieses wäre bei dem derzeitigen Rauschen von ca. 1-2 mV nicht mehr messbar.

Damit ergibt sich nach Formel 3.1:

$$1\,\mathrm{mV} = 30\,\mathrm{mV} \cdot \exp\left(-\frac{t}{25\,\mu\mathrm{s}}\right) \tag{4.31}$$

$$\Leftrightarrow t = -\ln\left(\frac{1\,\mathrm{mV}}{30\,\mathrm{mV}}\right) \cdot 25\,\mu s \tag{4.32}$$

$$t \approx 85\,\mu\mathrm{s}\,. \tag{4.33}$$

Nun wird mit diesem Wert ausgerechnet, mit welcher Wahrscheinlichkeit ein Pile-Up-Signal auftritt. Dazu wird oben genannte Eventrate genutzt. Die Verteilung entspricht der Poisson-Verteilung, da es sich um diskrete Ereignisse handelt.

$$1\,\mathrm{kHz} = 1 \cdot 10^3\,\mathrm{s}^{-1} = \frac{0,085}{85\mu\mathrm{s}} \tag{4.34}$$

$$P_{0,085}\,(2) = \frac{0,085^2}{2!} \cdot \exp(-0,085) \tag{4.35}$$

$$= 3,32 \cdot 10^{-3}\,. \tag{4.36}$$

Das heißt, dass bei einer Eventrate von einem Kilohertz pro Kristall und einer Signalhöhe von 30 mV pro Sekunde statistisch gesehen drei Pile-Up-Signale zu beobachten sind. Pro Minute sind dies mehr als 180 Pile-Up-Signale, somit ist dieser Algorithmus ein wichtiger Teil der Datenaufnahme.

In Abbildung 4.13 sind die Signalformen des Pile-Up-Signals sowie der separierten Signale in je einem Histogramm dargestellt. Um das Signal zu separieren, liest ein Makro (Anhang A.6) das gespeicherte Signal ein. Es wird eine Nulllinie von 8666 (in Kapitel 4.4.3 ermittelt) subtrahiert. Danach wird das Objekt „TSpectrum“ des ROOT-Frameworks genutzt. Dieses sucht nach beliebig vielen Peak-Positionen in einem Histogramm und speichert diese in einem Array. Um verlässlich Peaks finden zu können, muss der Klasse ein Sigma und einen Treshold in Prozent des Signal übergeben werden, diese wurden durch mehrmaliges Testen ermittelt. Um zu erkennen, an welcher Position ein Peak gefunden wird, wird die Stelle des erkannten Peaks mit einem kleinen roten Dreieck gekennzeichnet. Nun wird an das letzte gefundene Signal eine Exponential-Funktion gefittet. Wichtig ist dabei, den Fitbereich so einzugrenzen, dass ein gewisser Abstand (1% der x-Achse) zu den Randstellen des zu separienden Signals eingehalten wird, da der Signalverlauf unter Umständen dort keiner Exponentialfunktion entspricht und so ein Konvergieren des Fits verhindern würde. Die Parameter dieses ersten Fits werden nun für alle weiteren Fits verwendet, da diese unvollständig sind und sich somit nur schlecht anfitten lassen. Für jedes weitere unvollständige Signal wird nun

eine weitere Fitkurve erstellt, die mit einer Ausnahme einer Verschiebung in x-Richtung (t_0) der ersten Fitkurve entspricht. Diese Verschiebung in x-Richtung wird auch wiederum angefittet. Danach werden alle separierten Signale in ein separates Histogramm gezeichnet. Das zeitlich später einfallende Signal muss von dem Signalanteil des ersten Signals getrennt werden, dazu werden die Werte der Fitfunktion des ersten Signals in dem Bereich vom zweiten Signal abgezogen, wo sie sich überlagern. Das erste einfallende Signal, dass nur unvollständig vorliegt, wird mit Werten der Fitfunktion ergänzt. Somit ergibt sich in diesem Histogramm ab einem gewissen Punkt eine glatte Kurve, die rauschfrei zu sein scheint, da dies Werte der Fitfunktion sind. Dass dieses Signal in dieser Art separiert werden kann, soll eine kurze Rechnung verdeutlichen:

$$Signal_{gesamt} = S_1 + S_2 = \exp\left(-\frac{t}{\tau}\right)(U_1 + U_2 \exp\left(-\frac{t_0}{\tau}\right)) \tag{4.37}$$

$$Signal_1 = U_1 \cdot \exp\left(-\frac{t}{\tau}\right) \tag{4.38}$$

$$Signal_2 = U_2 \cdot \exp\left(-\frac{t}{\tau}\right) \cdot \exp\left(-\frac{t_0}{\tau}\right) \tag{4.39}$$

$$= U_2 \cdot \exp\left(-\frac{t - t_0}{\tau}\right) . \tag{4.40}$$

Eine zweite Möglichkeit besteht darin, nur die Maximas zu errechnen und diese zu speichern. Welche Art der Separation bevorzugt wird, ist deshalb abhängig von der Anzahl der zu berechnenden Pile-Up-Signale. Softwarebasierend wäre es jedoch ebenfalls möglich, einen softwarebasierenden Filter zu programmieren, der das Signal digital formt und so einen Pile-Up schon im Vorhinein verhindern könnte.

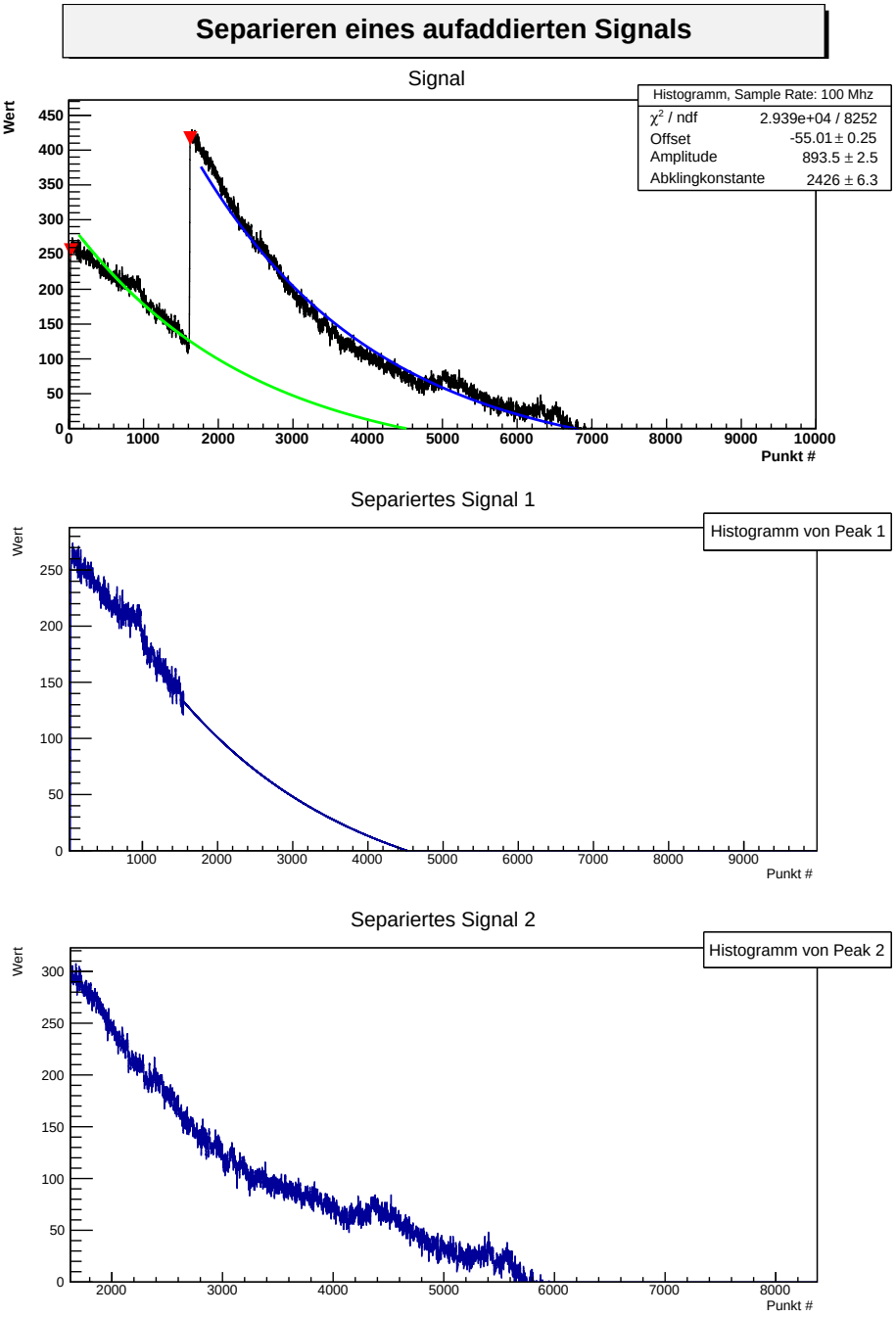


Abbildung 4.13.: Histogramm eines Pile-Up-Signals sowie der beiden separierten Signale. Aufnahmefrequenz 100 Mhz, Aufnahmezeit 100 μs. Im ersten Histogramm ist ein Pile-Up-Signal dargestellt. Die roten Dreiecke markieren die detektierten Peaks, die blaue Kurve ist ein Fit an das zweite Signal. Die grüne Kurve entspricht einer verschobenen blauen Kurve, so dass sie dem erwarteten Verlauf des unvollständigen Signals entspricht. Im zweiten Histogramm ist das erste separierte Signal dargestellt, im dritten Histogramm das zweite separierte Signal.

4.5.1. Verteilung von mehreren Pile-Up-Signalen

Der Pulsgenerator erzeugt im Burstmodus in einem zeitlichen Abstand von 16 μs zwei Pulse, die ein überlagertes Vorverstärkersignal verursachen. Abbildung 4.13 (Histogramm 1) zeigt die Form eines solchen Signals. Mithilfe eines Separationsalgorithmus' können Pile-Up-Signale separiert und untersucht werden. In diesem Fall wird eine andere Möglichkeit genutzt, die Signale zu separieren. Dazu werden wie in Kapitel 4.5 beschrieben Signalspitzen ermittelt, dann wird jedoch die die Differenz der Werte 10 ns vor vor dem Signalanstieg sowie 10 ns nach nach dem Signalanstieg ermittelt, da die Signalform in diesem Fall nicht relevant ist. Um die Peaks vergleichen zu können, muss die Nulllinie des FlashADC (8666) entfernt werden. Die Signalhöhen werden dann nach erstem und zweiten Signal getrennt in Histogramme eingetragen (siehe Abbildung 4.14).

Es wird deutlich, dass das zweite Signal trotz konstant gleichem Eingangssignal im Mittel größer ist als das erste. So ergibt sich beim ersten Signal ein mittlere Signalhöhe von $(262,7 \pm 1,9)$ Ausgabewerten, beim zweiten eine mittlere Signalhöhe von $(311,8 \pm 2,2)$ Ausgabewerten. Die Standardabweichungen der beiden separierten Signale sind ähnlich groß wie die Standardabweichung bei einem Einfachpuls, die Signale verhalten sich einzeln betrachtet bezüglich ihrer Schwankung wie jeweils ein Einzelpuls. Vermutet wird derzeit, dass dieses Verhalten an der LED liegt, die eine Abklingkurve besitzt und sich dort beim zweiten Puls somit ein abklingender Strom mit dem erneut vom Pulsgenerator applizierten Strom überlagert.

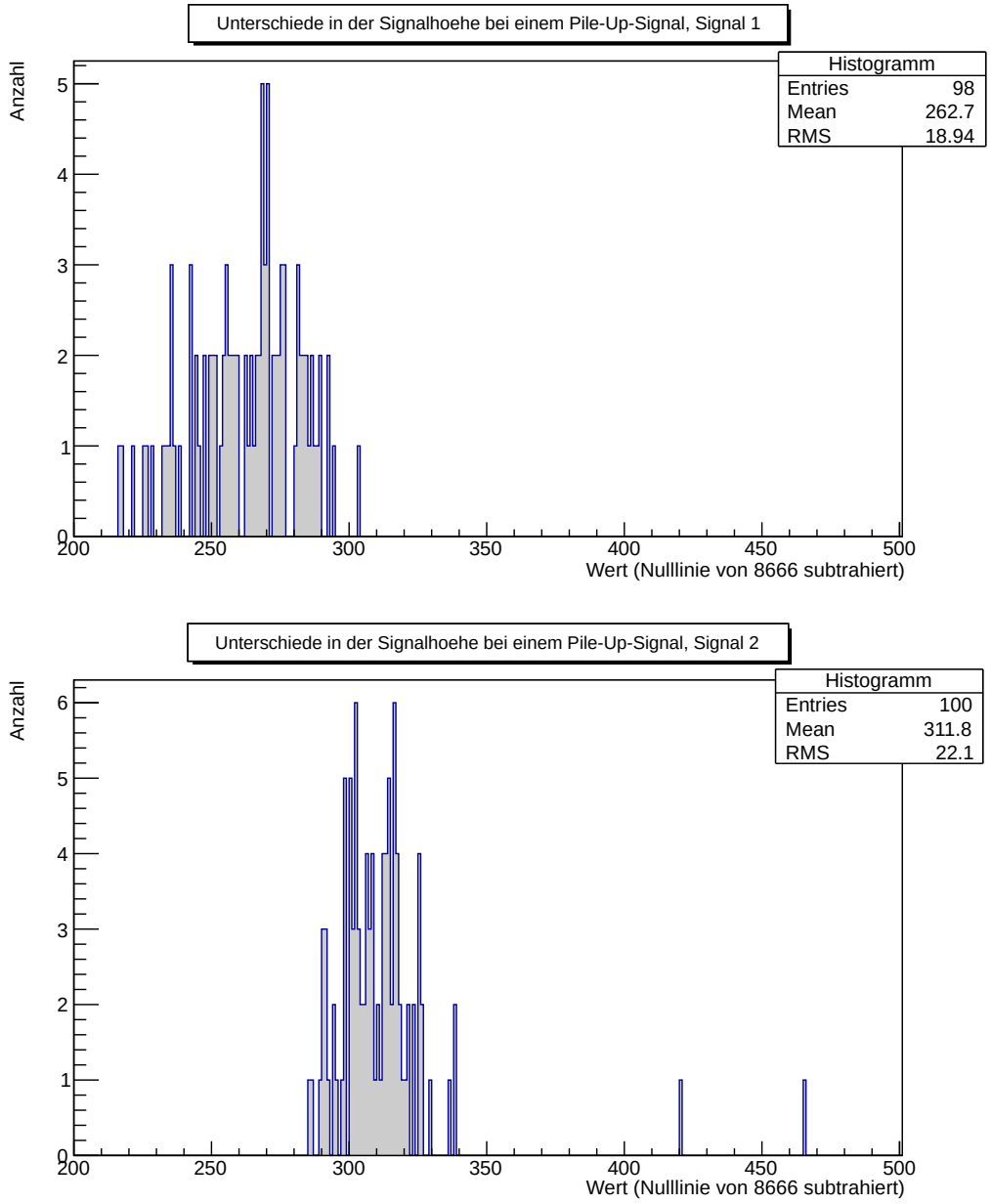


Abbildung 4.14.: Histogramm der Maximas von Signalen des Vorverstärkers bei konstantem LED-Eingangssignal (Doppelpuls). Die Signalform ist in Abbildung 4.13 abgebildet. Es ist zu erkennen, dass das zweite Signal im Mittel deutlich größer ist als das erste. Vermutet wird ein systematischer Fehler, verursacht durch die LED. Die Standardabweichungen der Signale liegen in der gleichen Größenordnung wie in Abbildung 4.12. Sie entsprechen einer Energieschwankung von ca. 9 MeV bei Histogramm 1 und 11 MeV bei Histogramm 2, wenn von einem Kristall bei -25℃ und einer APD mit Verstärkungsfaktor 50 ausgegangen wird.

5. Fazit und Ausblick

Anhand dieser Arbeit wurde eine Möglichkeit aufgezeigt, wie ein Datenaufnahmealgorithmus für den Prototypen der Rückwärtsendkappe aussehen kann und wie sich ein Testaufbau mit Vorverstärker und APD bei Detektion von Photonen verhält.

In Kapitel 2 wurde dazu erläutert, in welchem physikalischen Rahmen diese Bachelorarbeit angefertigt wurde und welche Anforderungen die Rückwärtsendkappe des PANDA-Experiments erfüllen muss. Dies ist in erster Linie die Detektion von Teilchenenergien ab einer Energie von wenigen MeV.

In Kapitel 3 wurde auf den experimentellen Aufbau eines Prototypen mit acht Kristallen eingegangen, der zurzeit entwickelt und aufgebaut wird. Dazu wurde erst auf die Eigenschaften der Kristalle und der Avalanche-Photodioden eingegangen und die Charakteristika dieser Komponenten aufgezeigt. Mithilfe von Bleiwolframat-Kristallen und APDs können Teilchenenergien von wenigen MeV detektiert werden. Um diese Signale auszulesen, wird ein speziell entwickelter Vorverstärker der Universität Basel verwendet. Die Arbeitsweise eines ladungsintegrierenden Verstärkers sowie die Signalform eines solchen wurde dargestellt, der aktuelle Aufbau des Prototypen mit acht Kristallen erläutert und anschließend auf die Digitalisierung eines Messsignals, in diesem Fall mithilfe eines FlashADCs, eingegangen. Dieser ist in der Lage, mit einer Abtastfrequenz von 25-100 Mhz Signale aufzunehmen und als 16-Bit-Werte abzuspeichern.

Anschließend wurde in Kapitel 4 die Entwicklung eines Datenaufnahmealgorithmus' für den FlashADC dokumentiert. Dazu wurde der Testaufbau erläutert, mithilfe dessen Lichtpulse erzeugt werden können. Die erwartete Vorverstärker-Signalhöhe für den Einfall von Muonen in einen auf -25℃ gekühlten Kristall wurde abgeschätzt, um das Testsignal in ähnlicher Höhe zu erzeugen. Die Funktionen des selbst erstellten Programms zum Auslesen des FlashADC wurden erläutert. In den darauf folgenden Unterkapiteln wurde das charakteristische Verhalten des FlashADC analysiert. So stellt sich heraus, dass die Nulllinie des FlashADCs ohne Anschluss eines Messsignals bei 8667 Ausgabewerten liegt und eine Unsicherheit von einem Ausgabewert besitzt, das entspricht einer Energieunsicherheit von ca. 0,5 MeV bei einem Kristall, der auf -25℃ gekühlt ist und einer angeschlossenen APD mit einem Verstärkungsfaktor von 50. Schließt man den Testaufbau ohne Hochspannung an der APD an, so verschiebt sich der mittlere Ausgabewert um 51 Kanäle nach unten auf 8616.

Das Rauschen besitzt nun eine Standardabweichung von 20,27 Ausgabewerten, was einer Energieunsicherheit von ca. 10 MeV bei oben genannten Parametern entspricht. Legt man zusätzlich Hochspannung an die APD an, verkleinert sich die Standardabweichung des Rauschens wieder auf den Wert des inneren Rauschens und die Nulllinie wandert wieder auf einen Ausgabewert von 8666. Anschließend wurden 100 Signale der LED gemessen. Hierbei stellt sich heraus, dass die Verteilung der Signale eine Standardabweichung von 18,15 Ausgabewerten besitzt, was einer Energieunsicherheit von ca. 9 MeV bei oben genannten Kristall- und APD-Parametern entspricht. Dies ist der LED geschuldet, da diese keine fest definierte Anzahl an Photonen emittieren kann.

In Kapitel 4.5 wurde ein Algorithmus zur Separation von Pile-Up-Signalen vorgestellt und an einem Beispiel erläutert. Anschließend wurden 100 Pile-Up-Signale aufgenommen, separiert und histogrammiert. Dabei wurde deutlich, dass das zweite Signal im Mittel eine größere Signalhöhe als das erste aufweist. Es wird vermutet, dass sich in der LED ein abklingender Strom mit einem neuen Strompuls überlagert und so mehr Photonen emittiert werden.

Da mit dieser Arbeit nur der Grundstein für eine digitale Datenaufnahme gelegt werden konnte, soll nun auf einige Dinge eingegangen werden, die im zeitlichen Rahmen der Bachelorarbeit nicht umgesetzt werden konnten.

So sind für die Datenaufnahme eines funktionierenden Prototyps 16 Messkanäle nötig, um jeden der Kristalle mit je zwei APDs aufnehmen zu können. Die Aufnahme weiterer Kanäle muss in einem nächsten Schritt implementiert werden. Falls es möglich sein soll, kontinuierlich über einen längeren Zeitraum Daten aufzunehmen, muss die Datenaufnahme angepasst werden, so dass während der Aufnahme immer wieder Daten aus dem FlashADC ausgelesen werden, um zu vermeiden, dass dieser aufgrund eines vollgeschriebenen Speichers die Datenaufnahme unterbricht. Da der Einschub-PC nur einen Gesamtspeicherplatz von ca. 16 GB besitzt, muss ein Datenstreaming auf einen anderen Server realisiert werden, auf dem die Messdaten dauerhaft gespeichert werden können. Das ROOT-Makro zum Separieren der Pile-Up-Signale muss in den Aufnahmealgorithmus integriert und darauf getestet werden, ob eine Überlagerung von mehr als zwei Signalen exakt separiert werden kann. Alternativ könnte ein völlig neuer Algorithmus implementiert werden, der softwarebasiert einen Shaper simuliert und somit einen Pile-Up verhindert. Die Ursache der unterschiedlichen Signalhöhen bei einem Pile-Up-Signal muss untersucht werden. Dazu könnte das Zeitintervall variiert werden, in der das zweite überlagernde Signal einfällt. Auch ließe sich eine zweite LED in den Aufbau integrieren, um die Vermutung, dass sich dort ein abklingender Strom mit einem neuen Strompuls überlagert, zu bestätigen.

Literaturverzeichnis

[1] Gesellschaft für Schwerionenforschung Darmstadt / PANDA Collaboration, PANDA Technical Design Report, 2008.

[2] Gesellschaft für Schwerionenforschung Darmstadt / PANDA Collaboration, Technical Progress Report.

[3] Hamamatsu, Si APD, S8664 Series, Short wavelength type APD, Sept. 2005.

[4] Universität Basel, Discrete Preamplifier for APD readout, LNP Preamplifier Version SP 883A02, Version 2.8, 31.8.2008.

[5] Particle Data Group, Section: Particle Detectors, S. 18, 2011, `http://pdg.lbl.gov/2011/reviews/rpp2011-rev-particle-detectors-accel.pdf`.

[6] SIS GmbH,SIS 3300/3301 65/100 Mhz VME FADCs User Manual, Version 3.00, 27.05.2002.

[7] Christina Haberkorn, Bachelorarbeit: Vorarbeiten zum Aufbau eines Prototypen für die Rückwärtsendkappe des elektromagnetischen Kalorimeters am PANDA-Experiment, Institut für Kernphysik der Universität Mainz, 2012

Abbildungsverzeichnis

A. Quellcode

A.1. Header-Datei der Klasse FlashADC

Listing A.1:

```
1  /*******************************************************
2   *
3  * Header File for Flash ADC SIS 3301 GAMMA
4  * written by Pascal Lautz
5  */



9  #ifndef FLASH_ADC_H_
10 #define FLASH_ADC_H_

12 #include<iostream>
13 #include<string>
14 #include<sstream>
15 #include<vector>

17 #include "vmelib.h"

19 #define FLASH_ADC_NAME  "SIS 3301 GAMMA"
20 #define PHYSICAL_ADDRESS 0x10000000

22 #define SIS3300_CONTROL_STATUS  0x00000000
23 #define SIS3300_MODID    0x00000004
24 #define SIS3300_ADC_MEMORY_PAGE_REGISTER  0x00000034
25 #define SIS3300_ADC1_OFFSET   0x04000000
26 #define SIS3300_EVENT_CONFIG_ADC12 0x00200000
27 #define SIS3300_END_ADDRESS_THRESHOLD_ADC12 0x02000004
28 #define SIS3300_PRETRIGGER_DELAY_TRIGGERGATE_LENGTH_ADC12 0x02000008
29 #define SIS3300_ACTUAL_SAMPLE_VALUE_ADC12 0x02000020
30 #define SIS3300_TRIGGER_SETUP_ADC1 0x02000030
31 #define SIS3300_TRIGGER_TRESHOLD_ADC1 0x02000034
32 #define SIS3300_RAW_DATA_BUFFER_CONFIG_ADC12 0x0200000C
33 #define SIS3300_ACQUISITION_CONTROL 0x00000010
34 #define SIS3300_GENERAL_RESET 0x00000020
35 #define SIS3300_EXTERNAL_STOP_DELAY 0x00000018
36 #define SIS3300_VME_START_SAMPLING 0x00000030
37 #define SIS3300_VME_STOP_SAMPLING 0x00000034
38 #define SIS3300_TRIGGER_THRESHOLD 0x00100004
39 #define SIS3300_TRIGGER_THRESHOLD_12 0x00200004
```

```
40 #define SIS3300_EVENT_CONFIGURATION 0x00100000
41 #define SIS3300_EVENT_CONFIGURATION_12 0x00200000
42 #define SIS3300_MAX_EVENT 0x0010002c
43 #define SIS3300_MEMBASE_BANK1_12 0x00400000
44 #define SIS3300_BANK1_ADD_CNT_12 0x00200008
45 #define SIS3300_BANK1_MEM_CLEAR 0x00000048

47 using namespace std;

49 class FlashADC //: DAQMODULE
50 {
51  public:
52   FlashADC (unsigned int baseaddress);
53   virtual ~FlashADC();

55   void InitializeRegisters();
56   void initialSetup();
57   void GetModuleInfo();
58   void SetLedStatus(int input);
59   long GetLedStatus();
60   long GetSampleData();
61   void setControlRegister(long input);
62   void setAcquisitionControlRegister(long input);
63   void setSampleTime(int time);
64   void setClock(int clock=10);
65   void setTreshold(long threshold=2000, string legt="gt");
66   void resetADC();
67   void startSampling();
68   void stopSampling();
69   void getSamplingStatus();
70   void getGraph();
71   void armForSampling();
72   void disarmSampling();
73   void takeOneSample();
74   void takeUntriggeredSample();
75   vector<float> getXValues();
76   vector<float> getYValues();
77   int getSamples();

79  private:
80   VMEBridge *vme;
81   int samples;
82   int time;
83   int image;
84   vector<float> xValue;
85   vector<float> yValue;
86   long startpointer;
87   long currentpointer;
88   long volatile *ph_FlashADC;
89   long volatile *pl_control_reg;
90   long volatile *pl_firmware;
```

```
91    long volatile *pl_pageRegister;
92    long volatile *pl_eventBufferADC1;
93    long volatile *pl_actual_sample_data;
94    long volatile *pl_trigger_setup_ADC1;
95    long volatile *pl_trigger_treshold_ADC1;
96    long volatile *pl_trigger_pretrigger_ADC12;
97    long volatile *pl_trigger_raw_data_buffer_config_ADC12;
98    long volatile *pl_aquisition_control;
99    long volatile *pl_general_reset;
100   long volatile *pl_external_stop_delay;
101   long volatile *pl_vme_start_sampling;
102   long volatile *pl_vme_stop_sampling;
103   long volatile *pl_trigger_threshold;
104   long volatile *pl_trigger_threshold_12;
105   long volatile *pl_event_configuration;
106   long volatile *pl_event_configuration_12;
107   long volatile *pl_max_event;
108   long volatile *pl_membase_bank1_12;
109   long volatile *pl_bank1_add_cnt_12;
110   long volatile *pl_bank1_mem_clear;

112 };

114 #endif
```

A.2. C++-Datei der Klasse FlashADC

Listing A.2:

```
1  /*Class to control the FlashADC SIS3301 Model
2   written by Pascal Lautz
3  May 2012
4  */



7  #include "FlashADC.h"
8  #include "stdio.h"
9  #include "stdlib.h"
10 #include<iostream>
11 #include<string>
12 #include<sstream>
13 #include <unistd.h>
14 #include <vector>



17 FlashADC::FlashADC(unsigned int baseaddress)
18 {
19   vme = new VMEBridge();

21   cout << " FlashADC:: Get Image" << endl;
```

```
23   int success=0;
24   unsigned int slot=0;

26   image = vme->getImage(baseaddress, 0x1000000, A32, D32, MASTER);

28   if (vme->there(baseaddress)) // is someone responding?
29     {
30       success=1;
31     }
32   else{
33     vme->releaseImage(image);
34   }

36   if (success)
37     cout << "Found FlashADC SIS 3302 in base address:"
38    << hex << "0x"<<baseaddress << endl;
39   else
40     cout << "No FlashADC SIS 3302 was found..." << endl;

42 }

44 FlashADC::~FlashADC()
45 {
46   cout << " FlashADC: releasing image..." << endl;
47   vme->releaseImage(image);
48   delete(vme);

50 }

52 void FlashADC::InitializeRegisters()
53 {
54   printf("\n***************************************") ;
55   printf("\n*    InitializeRegisters  *") ;
56   printf("\n***************************************\n") ;

58 //Adresses of the registers routed by the virtual address of the ADC
59   ph_FlashADC = (long *) vme->getPciBaseAddr(image);

61   pl_control_reg = (long*)((long)ph_FlashADC + (long)SIS3300_CONTROL_STATUS);
62   pl_firmware= (long*) ((long)ph_FlashADC + (long)SIS3300_MODID);

64   pl_eventBufferADC1=(long*)((long)ph_FlashADC + (long)
         SIS3300_ADC_MEMORY_PAGE_REGISTER);

66   pl_actual_sample_data= (long *) ((long)ph_FlashADC + (long)
         SIS3300_ACTUAL_SAMPLE_VALUE_ADC12);

68   pl_trigger_setup_ADC1= (long*) ((long)ph_FlashADC + (long)
         SIS3300_TRIGGER_SETUP_ADC1);

70   pl_trigger_treshold_ADC1= (long*) ((long)ph_FlashADC + (long)
```

```
        SIS3300_TRIGGER_TRESHOLD_ADC1);

72    pl_trigger_pretrigger_ADC12= (long*) ((long)ph_FlashADC + (long)
        SIS3300_PRETRIGGER_DELAY_TRIGGERGATE_LENGTH_ADC12);

74    pl_trigger_raw_data_buffer_config_ADC12= (long*) ((long)ph_FlashADC + (long)
        SIS3300_RAW_DATA_BUFFER_CONFIG_ADC12);

76    pl_aquisition_control= (long*)((long)ph_FlashADC + (long)
        SIS3300_ACQUISITION_CONTROL);

78    pl_general_reset= (long*)((long)ph_FlashADC + (long)SIS3300_GENERAL_RESET);

80    pl_external_stop_delay= (long*)((long)ph_FlashADC + (long)
        SIS3300_EXTERNAL_STOP_DELAY);

82    pl_vme_start_sampling= (long*)((long)ph_FlashADC + (long)
        SIS3300_VME_START_SAMPLING);

84    pl_vme_stop_sampling= (long*)((long)ph_FlashADC + (long)SIS3300_VME_STOP_SAMPLING
        );

86    pl_trigger_threshold= (long*)((long)ph_FlashADC + (long)SIS3300_TRIGGER_THRESHOLD
        );

88    pl_trigger_threshold_12= (long*)((long)ph_FlashADC + (long)
        SIS3300_TRIGGER_THRESHOLD_12);

90    pl_event_configuration= (long*)((long)ph_FlashADC + (long)
        SIS3300_EVENT_CONFIGURATION);

92    pl_event_configuration_12= (long*)((long)ph_FlashADC + (long)
        SIS3300_EVENT_CONFIGURATION_12);

94    pl_max_event= (long*)((long)ph_FlashADC + (long)SIS3300_MAX_EVENT);

96    pl_membase_bank1_12= (long*)((long)ph_FlashADC + (long)SIS3300_MEMBASE_BANK1_12);

98    pl_bank1_add_cnt_12= (long*)((long)ph_FlashADC + (long)SIS3300_BANK1_ADD_CNT_12);

100   pl_bank1_mem_clear= (long*)((long)ph_FlashADC + (long)SIS3300_BANK1_MEM_CLEAR);
101 }

103 void FlashADC::initialSetup(){


106   *pl_control_reg= 0x00000001; //enable LED
107   *pl_control_reg= 0x00100000; //non-invertable trigger out
108   *pl_control_reg= 0x00000004; //enable trigger out
109   *pl_aquisition_control= 0x70000000; //clear clock
110   *pl_aquisition_control= 0x00000100; //enable LEMO start/stop
```

```
112 }


115 //Calculate Time in Microseconds to Amount of Samples
116 void FlashADC::setSampleTime(int time){
117   int clock = (*pl_aquisition_control & 0x00007000)>>12;
118   if (clock ==0){//50 MHz
119     samples= (int) 100 * time;
120   }

122       if (clock ==1){//50 MHz
123     samples= (int) 50 * time;
124   }
125   else if (clock == 2){
126     samples= (int) 25*time;
127   }

129 }

131 int FlashADC::getSamples(){
132 return samples;
133 }

135 void FlashADC::setClock(int clock){
136   //Set the intern Clock of the ADC
137   switch (clock) {
138     case 100:
139       *pl_aquisition_control= 0x00000000;
140     break;

142     case 50:
143       *pl_aquisition_control= 0x00001000;
144     break;

146     case 25:
147       *pl_aquisition_control= 0x00002000;
148     break;

150     default:
151       *pl_aquisition_control= 0x00000000;
152     break;
153   }
154 }

156 //Treshold for a self-triggered DAQ, not used in the current setup
157 void FlashADC::setTreshold(long threshold, string legt){
158   //default=2000,
159   long value;
160   value = (threshold&0xfff)<<16 | threshold&0xfff;
161   //greater than
```

```
163   if (legt==”gt”) value = value | 0x00000000;
164   else if (legt==”le”) value = value | 0x80000000;

166   *pl_trigger_threshold = value; //setup threshold

168 }

170 void FlashADC::resetADC(){
171   *pl_general_reset= 0x00000001; // reset
172 }

174 void FlashADC::armForSampling(){
175  *pl_aquisition_control= 0x00000001; //start data sampling of bank1
176 }

178 void FlashADC::disarmSampling(){
179  *pl_aquisition_control= 0x00010000; //stop data sampling of bank1
180 }

182 void FlashADC::startSampling(){
183 *pl_vme_start_sampling= 0x00000001;
184 }

186 void FlashADC::stopSampling(){
187 *pl_vme_stop_sampling= 0x00000001;
188 }

190 void FlashADC::getSamplingStatus(){
191  int status= (*pl_aquisition_control & 0x000f0000)>>16;
192  if (status==1) cout<<”ADC is currently sampling”<<endl;
193  else if (status==0) cout<<”ADC is currently NOT sampling”<<endl;
194 }

196 vector<float> FlashADC::getXValues(){
197   return xValue;
198 }

200 vector<float> FlashADC::getYValues(){
201   return yValue;
202 }

204 void FlashADC::takeOneSample(){

206   xValue.clear();
207   yValue.clear();
208   startpointer=*pl_bank1_add_cnt_12 & 0x0000ffff;
209   //cout<<”Startpointer ”<<dec<<(startpointer)<<endl;
210   this->armForSampling();
211   currentpointer=*pl_bank1_add_cnt_12 & 0x0000ffff;
212   //this->getSamplingStatus();
```

```
213   while ((currentpointer - startpointer)/4 < samples){
214    currentpointer=*pl_bank1_add_cnt_12 & 0x0000ffff;
215   }

217   this->stopSampling();
218   //cout<<"Stoppointer: "<<dec<<currentpointer<<endl;

220   this->disarmSampling();

222   //Clear the pointer, important for not getting waste data
223   currentpointer=NULL;
224   startpointer=NULL;

226   cout<<"Taken following number of points for this sample: "<<dec<<samples<<endl;

228   for (int i=0;i<samples;i++){
229    float tmp = (*(pl_membase_bank1_12 +startpointer +i)>>16&0x3fff);
230    yValue.push_back(tmp);
231    xValue.push_back(i);
232   }


235 }

237 void FlashADC::takeUntriggeredSample(){

239   xValue.clear();
240   yValue.clear();
241   startpointer=*pl_bank1_add_cnt_12 & 0x0000ffff;
242   currentpointer=*pl_bank1_add_cnt_12 & 0x0000ffff;
243   //cout<<"Startpointer "<<dec<<(startpointer)<<endl;
244   this->armForSampling();
245   this->startSampling();

247   this->getSamplingStatus();
248   while ((currentpointer - startpointer)/4 < samples){
249    currentpointer=*pl_bank1_add_cnt_12 & 0x0000ffff;
250   }

252       this->stopSampling();
253   //cout<<"Stoppointer: "<<dec<<currentpointer<<endl;

255   this->disarmSampling();

257   //Clear the pointer, important for not getting waste data

259       currentpointer=NULL;
260   startpointer=NULL;

262   cout<<"Taken following number of points for this sample: "<<dec<<samples<<endl;
```

```
264   for (int i=0;i<samples;i++){
265     float tmp = (*(pl_membase_bank1_12 +startpointer +i)>>16&0x3fff);
266     yValue.push_back(tmp);
267     xValue.push_back(i);
268   }

270 }
```

A.3. Header-Datei der Klasse DAQ

Listing A.3:

```
1  #ifndef SYMBDAQ_H
2  #define SYMBDAQ_H

4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <iostream>
7  #include <TStyle.h>
8  #include <TDatime.h>
9  #include <TTree.h>
10 #include <TFile.h>
11 #include <TGraph.h>
12 #include <TFrame.h>
13 #include <TString.h>
14 #include <TH1.h>
15 #include <TH1F.h>
16 #include <TCanvas.h>
17 #include <TApplication.h>
18 #include <vector>
19 #include <string>
20 #include <sstream>
21 #include "FlashADC.h"
22 #include <signal.h>
23 #include <time.h>
24 #include "TSystem.h"
25 #include "TROOT.h"
26 #include "TAxis.h"

28 using namespace std;

30 class DAQ //: DAQMODULE
31 {
32  public:
33   DAQ();
34   void getUntriggeredSample();
35   void getOneSample();
36   void getNSamples();
37   virtual ~DAQ();

39  private:
```

```
40    int samplerate;
41    TString rate;
42    FlashADC *adc;
43    TFile *RootFile;
44    void createFile();
45    void closeFile();
46  };

48  #endif
```

A.4. C++-Datei der Klasse DAQ

Listing A.4:

```
1   #include "DAQ.h"




7   //Constructor for the Data Acquisition
8    DAQ::DAQ(){
9         adc = new FlashADC(PHYSICAL_ADDRESS);
10   adc->InitializeRegisters();
11   adc->resetADC();
12   adc->initialSetup();
13        cout<<"Which Sample Rate do you want to use? (100, 50, 25)"<<endl;
14        char input[20];
15        cin.getline(input,20);
16        rate=input;
17        samplerate = atoi(input);
18        adc->setClock(samplerate);
19        cout<<"Which Time do you want to sample (Microseconds)?"<<endl;
20        cin.getline(input,20);
21        int sampletime = atoi(input);
22   adc->setSampleTime(sampletime);
23        //For using vectors also in ROOT
24        gROOT->ProcessLine("#include <vector>");
25  }

27  void DAQ::getUntriggeredSample(){
28        adc->takeUntriggeredSample();
29        this->createFile();
30        TTree *myTree = new TTree("Tree","Data Container");
31        vector<float> xval= adc->getXValues();
32   vector<float> yval= adc->getYValues();
33   TApplication * App = new TApplication("App", 0, 0);
34        TCanvas *c1 = new TCanvas("Canvas Untriggered Sample","Untriggered Sample"
              ,10,10,700,500);
35        c1->cd();
```

```
36        TH1F *hist = new TH1F("Histogramm, Sample Rate: "+rate+" Mhz","Untriggered
              Sample;Sample #; Value",xval.size(),0,xval.size());

38    for (int i=0; i<xval.size();i++){
39     hist->Fill(i,yval[i]);
40    }

42        hist->Draw();
43        c1->Update();
44        cout<<"Press Key to continue and to save the Data"<<endl;
45        int key;
46        cin>>key;

48         myTree->Branch("Histogramm","TH1F",&hist,32000,0);
49         myTree->Branch("XWert",&xval);
50         myTree->Branch("YWert",&yval);
51         myTree->Fill();
52         this->closeFile();

54  }


57  //For reading one sample and saving it into a ROOT file
58  void DAQ::getOneSample(){
59        adc->takeOneSample();
60        this->createFile();
61        TTree *myTree = new TTree("Tree","Data Container");
62        vector<float> xval= adc->getXValues();
63    vector<float> yval= adc->getYValues();
64        TApplication * App = new TApplication("App", 0, 0);
65        TCanvas *c1 = new TCanvas("Canvas Untriggered Sample","Untriggered Sample"
              ,10,10,700,500);
66        c1->cd();

68        TH1F *hist = new TH1F("Histogramm, Sample Rate: "+rate+" Mhz","Signal;Sample
               #; Value",xval.size(),0,xval.size());

70    for (int i=0; i<xval.size();i++){
71     hist->Fill(i,yval[i]);
72    }

74        hist->Draw();
75        c1->Update();
76        cout<<"Press Key to continue and to save the Data"<<endl;
77        int key;
78        cin>>key;

80         myTree->Branch("Histogramm","TH1F",&hist,32000,0);
81         myTree->Branch("XWert",&xval);
82         myTree->Branch("YWert",&yval);
83         myTree->Fill();
```

```
84          this->closeFile();

86  }

88  //For reading N samples given by the user
89  void DAQ::getNSamples(){
90     this->createFile();
91     TTree *myTree = new TTree("Tree","Data Container");
92     vector<float> xval;
93     vector<float> yval;
94     cout<<"How many Samples do you want to take?"<<endl;
95     int input;
96     cin>>input;
97     cout<<"Do you want to plot Histograms to every Sample? (y/n)"<<endl;
98     char createSample[10];
99     cin.ignore(1);
100    cin.getline(createSample,10);
101    bool histograms=false;

103    if (strcmp(createSample,"y")==0) {
104       cout<<"Create also Histograms!"<<endl;
105       histograms=true;

107    }

109    for (int i=0; i<input;i++){
110       adc->takeOneSample();
111       xval= adc->getXValues();
112       yval= adc->getYValues();
113       stringstream ss;
114       ss<< (i+1);
115       string strI=ss.str();
116       TString name= strI;

118       if (histograms) {
119          TH1F *hist = new TH1F("Histogramm"+name+", Sample Rate: "+rate+" Mhz","
              Signal;Sample #; Value",xval.size(),0,xval.size());
120          for (int i=0; i<xval.size();i++){
121                hist->Fill(i,yval[i]);
122          }

124          myTree->Branch("Histogramm"+name,"TH1F",&hist,32000,0);
125       }
126       myTree->Branch("XWert"+name,&xval);
127       myTree->Branch("YWert"+name,&yval);
128       myTree->Fill();

130    }
131    this->closeFile();
132 }
```

```
135 void DAQ::createFile(){
136       time_t rawtime;
137       struct tm * myTime;
138       time ( &rawtime );

140       myTime = localtime ( &rawtime );
141       char date[40];
142   TString name="rootfiles/Measurement";
143       strftime(date,40,"%Y-%m-%d_%H-%M-%S",myTime);
144       TString datestr=date;
145   name += datestr;
146   name+=".root";
147       RootFile =new TFile(name,"RECREATE","Measurement of FADC");
148 }

150 void DAQ::closeFile(){
151    RootFile->Write();
152    RootFile->Close();
153    delete adc;
154 }

156  DAQ::~DAQ(){

158  }
```

A.5. C++-Datei der Klasse Proto8

Listing A.5:

```
1  /*
2   * File: proto8.cpp
3   * Author: Pascal Lautz
4   *
5   * Created on 09 May 2012, 10:02
6   */

8  #include <cstdlib>
9  #include <signal.h>
10 #include "DAQ.h"
11 #include <iostream>

13 using namespace std;
14  bool shutdown=false;

16 // stop data taking in case ctr+c were pressed
17 void ctrl_c(int)
18 {
19   printf("\n CTRL-C pressed \n\r");
20   shutdown = true; // tell main loop that it has to exit
21 }
```

```
24 int main(int argc, char *argv[])
25 {
26     //Function to stop data Taking
27   signal(SIGINT, ctrl_c);
28   int input;
29    DAQ *myDAQ= new DAQ();


32       cout<<"What do you want to do?"<<endl;
33       cout<<"1: Take one sample"<<endl;
34       cout<<"2: Take N Samples"<<endl;
35       cout<<"3: Take untriggered Samples"<<endl;
36       cout<<"4: Exit the program"<<endl;

38       cin>>input;

40       if (input == 1){
41        cout<<"Taking one Sample"<<endl;
42          myDAQ->getOneSample();


45       }

47       else if (input == 2){
48        myDAQ->getNSamples();

50       }

52       else if (input == 3){
53          myDAQ->getUntriggeredSample();
54       }

56        else if (input == 4){
57           exit(1);
58       }




64 }
```

A.6. C-Makro zum Separieren eines Pile-Up-Signals

Listing A.6:

```
1 /*Macro to fit the decay curve of the PreAMP, to convolute the signals if the is a
      pile-up
2 *written by Pascal Lautz
```

```
3  May 2012
4  */

6  #include "TH1.h"
7  #include "TMath.h"
8  #include "TF1.h"
9  #include "TLegend.h"
10 #include "TCanvas.h"
11 #include "TGraph.h"
12 #include <TFile>
13 #include "TAttLine.h"
14 #include "TObjArray.h"
15 #include <sstream>
16 #include "TPaveLabel.h"
17 //offset of the FlashADC
18 float offset=8666;

20 //Fit Function

22 Double_t exp_decay(Double_t *x, Double_t *par){
23   Double_t exponent=TMath::Exp(-x[0]/par[2]);
24   return par[0]+par[1]*exponent;
25 }

27 Double_t exp_decay_shifted(Double_t *x, Double_t *par){
28   Double_t exponent=TMath::Exp(-(x[0]-par[3])/par[2]);
29   return par[0]+par[1]*exponent;

31 }

33 void deconvolute_signals(TString filename) {

35 TFile *myfile= new TFile(filename,"OPEN","Root-Daten");
36 TFile *newfile= new TFile("deconvoluted_"+filename,"RECREATE","Root-Daten");
37 TH1F *h1=(TH1F*)myfile->Get("Histogramm, Sample Rate: 100 Mhz;1");
38 h1->SetMinimum(0);
39 h1->GetYaxis()->SetTitle("Value");
40 //h1->SetMaximum(2000);

42 int nbins=h1->GetNbinsX();

44 float *source= new float[10000];
45 float *dest = new float[10000];
46 for (int i = 0; i < nbins; i++) source[i]=h1->GetBinContent(i + 1);
47 TCanvas *c1= new TCanvas("c1","Graph Canvas",10,10,700,1000);
48 TPaveLabel *title = new TPaveLabel(0.1,0.96,0.9,1,"Separieren eines aufaddierten
       Signals");
49 title->Draw();

51 //Find peaks with TSpectrum
52 TSpectrum *spec = new TSpectrum();
```

```
53 int nfound;
54 nfound = spec->SearchHighRes(source, dest, nbins, 4, 97, kTRUE, 2, kFALSE, 3);
55 cout<<"Found Number of peaks:"<<nfound<<endl;


58 TObjArray *fitfunctions= new TObjArray(nfound);
59 TObjArray *histograms = new TObjArray(nfound);

61 c1->Divide(1,nfound+1);
62 c1->cd(1);



66 h1->SetMaximum((h1->GetBinContent(h1->GetMaximumBin())-offset)*1.1);

68 //Extract the offset
69 for (int i=0;i < nbins;i++){
70 h1->SetBinContent(i,h1->GetBinContent(i)-offset);
71 }




76 //Just for setting a marker to the peaks
77 Float_t *xpeaks = spec->GetPositionX();
78 Double_t a;
79 Float_t fPositionX[100];
80 Float_t fPositionY[100];
81 Int_t bin;

83 for (i = 0; i < nfound; i++) {
84  a=xpeaks[i];
85   cout<<xpeaks[i]<<endl;
86  bin = 1 + Int_t(a + 0.5);
87  fPositionX[i] = h1->GetBinCenter(bin);
88  fPositionY[i] = h1->GetBinContent(bin);
89 }
90 TPolyMarker * pm = (TPolyMarker*)h1->GetListOfFunctions()->FindObject("TPolyMarker
       ");
91 if (pm) {

93 h1->GetListOfFunctions()->Remove(pm);
94 delete pm;
95 }
96 pm = new TPolyMarker(nfound, fPositionX, fPositionY);
97 h1->GetListOfFunctions()->Add(pm);
98 pm->SetMarkerStyle(23);
99 pm->SetMarkerColor(kRed);
100 pm->SetMarkerSize(1.3);
```

```
103 //Fit the last signal.... for use in former fits
104 TF1 *fitfunc= new TF1("FitFunction",exp_decay,fPositionX[0]+(nbins/100),nbins,3);
105 fitfunc->SetLineColor(kBlue);
106 fitfunc->SetParameters(0,800,2400);

108 h1->Fit(fitfunc,"R");


111 //Fit the incomplete second signal, Parameter 2 fixed
112 int count=0;
113 TF1 *currentFit;

115 //Draw Fitfunctions in first Pad
116 for (int p=nfound-1;p>0;p--){
117   fitfunctions->Add(new TF1("FitFunction",exp_decay_shifted,fPositionX[p]+(nbins
          /100),fPositionX[p-1]-(nbins/100),4));
118   currentFit=(TF1*) fitfunctions->Last();
119   currentFit->SetLineColor(kGreen);

121   Double_t parameters[4];

123   fitfunc->GetParameters(&parameters[0]);

125   currentFit->SetParameters(parameters[0],parameters[1],parameters[2], (fPositionX
          [0]-fPositionX[1]));
126   currentFit->SetParLimits(0,parameters[0],parameters[0]);
127   currentFit->SetParLimits(1,parameters[1],parameters[1]);
128   currentFit->SetParLimits(2,parameters[2],parameters[2]);

130   h1->Fit(currentFit,"R+");

132   //h2->Draw();
133   h1->Draw();
134   currentFit->SetRange(fPositionX[p]+(nbins/100),nbins);
135   currentFit->Draw("SAME");
136   c1->Update();
137 }


140 TH1F *currentHisto;
141 gStyle->SetOptStat("n");

143 //Now make Pads with deconvoluted Signals
144 for (int c=1;c<=nfound;c++){
145   c1->cd(c+1);
146    stringstream ss;
147        ss<< (c);
148   TString counter=ss.str();
149   histograms->Add(new TH1F("Histogramm of Peak "+counter,"Deconvoluted Signal;Point
           no;Value",nbins - xpeaks[nfound-c],xpeaks[nfound-c],(nbins - xpeaks[nfound-c
          ])));
```

```
150   cout<<histograms->Last()<<endl;
151   currentHisto=(TH1F*) histograms->Last();
152   currentHisto->SetMinimum(0);

154   if (c<= fitfunctions->GetEntries()){
155    currentFit=(TF1*) fitfunctions->At(c-1);
156   }

158   if (c==1){

160    for (i=0;i<nbins;i++){

162     if (i >= (xpeaks[0]-(nbins/100))){
163      currentHisto->SetBinContent(i, currentFit->Eval(i,0,0,0));
164     }

166     else if (i < xpeaks[0]) {
167      currentHisto->SetBinContent(i, h1->GetBinContent(i));
168     }
169    }
170   }

172   else {
173    for (i=xpeaks[0];i<nbins;i++){


176     if (currentFit->Eval(i,0,0,0) > 0){
177      float val=(h1->GetBinContent(i) - currentFit->Eval(i,0,0,0));
178      currentHisto->SetBinContent(i-xpeaks[0],val);
179     }

181     else{
182     currentHisto->SetBinContent(i-xpeaks[0],h1->GetBinContent(i));
183     }
184    }

186   }

188   currentHisto->Draw();
189   }


192 c1->Update();
193 c1->Write();
194 }
```

Danksagung

An dieser Stelle möchte ich mich bei einigen Menschen bedanken:

Prof. Dr. Frank Maas, der mich während der Anfertigung der Thesis sehr gut betreute und mir immer wieder mit Tipps und Anregungen hilfreich zur Seite stand.

Dr. Yue Ma, der mir mit großer Motivation einen flüssigen Einstieg in das Thema meiner Bachelorarbeit verschaffte und mich in den ersten Wochen mit seinem Optimismus und seinem Wissen bei vielen schwierigen Aufgaben begleitete.

Prof. Dr. Josef Pochodzalla, der sich als Zweitkorrektor dieser Thesis zur Verfügung stellte.

Christina Haberkorn und Elvar Steinn Kjartansson, die diese Arbeit korrekturgelesen und mich mein gesamtes Studium hindurch als Freunde und Kommilitonen begleitet haben.

Danken möchte ich auch meinen Eltern und meiner Freundin Jana, die mich während der Studienzeit immer unterstützt und so einen maßgeblichen Anteil zum erfolgreichen Abschluss dieses Studiums beigetragen haben.