Aufbau und Inbetriebnahme des Silizium Detektor-Balls am QCLAM Spektrometer

Assembly and Initiation of the Silicon Detector Ball at QCLAM Spectrometer

Bachelor-Thesis von Henno Lauinger

November 2012

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Physik
Institut für Kernphysik

Aufbau und Inbetriebnahme des Silizium Detektor-Balls am QCLAM Spektrometer
Assembly and Initiation of the Silicon Detector Ball at QCLAM Spectrometer

Vorgelegte Bachelor-Thesis von Henno Lauinger

1. Gutachten: Prof. Dr. Peter von Neumann-Cosel
2. Gutachten: Dipl.-Phys. Jonny Birkhan

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 5. November 2012

(Henno Lauinger)

Inhaltsverzeichnis

1 Experiment

Der Silizium PIN-Detektor Ball (Si-Ball) ist ein Detektorsystem zur Energiemessung von geladenen Teilchen im vollen Raumwinkel.

Im speziellen soll er für Koinzidenzmessungen bei Aufspaltungen von ^{3}He, ^{12}C und ^{16}O Targets zur Detektion der abgespaltenen ^{4}He beziehungsweise ^{1}H und ^{2}H Kerne zum Einsatz kommen, kurz $X(e,e'x)Y$. Hierbei wird das entsprechende Isotop mit Hilfe des Elektronenstrahls aus dem S-DALINAC in jeden beliebigen Zustand angeregt. Darunter befindet sich auch ein Zustand bei dem vermutet wird, dass sich eines oder mehrere der erwarteten Ejektile in einem Gebundenen Zustand um einen Kern niedrigerer Massenzahl befindet.

Die Energien der am Target gestreuten Elektronen werden im QCLAM Spektrometer gemessen, welches seinen Namen aus den muschelförmigen Quadrupolmagneten hat, welche Elektronen unterschiedlicher Energien auf unterschiedliche Kreisbahnen ablenken. Diese passieren die Driftkammern, in welchen mehrere Schichten von Drähten die abgelenkte Bahn der Elektronen rekonstruieren können, wodurch deren Energie nach dem Streuvorgang bestimmt werden kann.

Über die Energie des Elektronenstrahls, der gestreuten Elektronen und der Ejektile, die mit dem Si-Ball gemessen wird kann nun die Energie des im Target angeregten Zustands bestimmt werden. Durch eine Flugzeit Messung von Projektil und Ejektil kann nun ermittelt werden, ob bei einem gemessenen Elektronen- und Si-Ball-Signal eine Koinzidenz vorliegt. Ist dies nicht der Fall kann dieses Ereignis keinem bestimmten Zustand des Targets zugeordnet werden, das gemessene Elektron muss nicht unbedingt das gemessene Ejektil aus dem Target gelöst haben, es könnte sich auch um einen statistischen Zerfall handeln.

Der komplett aufgebaute Si-Ball enthält Detektoren rund um das Target mit denen etwa 36,5% der Gesamtfläche des Balls abgedeckt werden kann. Dadurch lässt sich auch zeigen, dass die Ejektile in eine bestimmte Richtung emittiert werden und dort gegebenenfalls von Detektoren mit besserer Auflösung gemessen werden können.

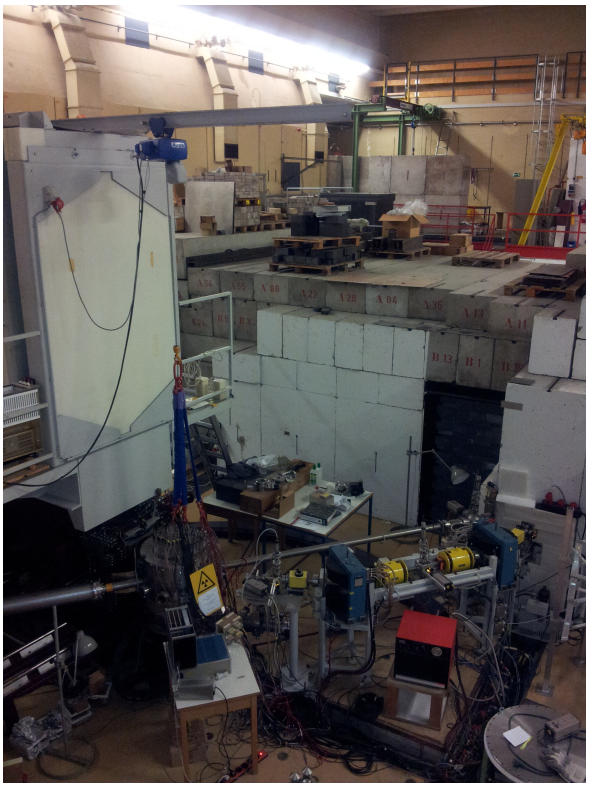


Abbildung 1.: QCLAM Aufbau in der Messhalle am IKP. Der Elektronenstrahl kommt durch das Rohr von rechts in die Streukammer und wird dort am Target ins QCLAM-Spektrometer (weißer Kasten links) gestreut.

Als Beispiel ist der ^{12}C Kern (X) zu nennen, bei dem in besagtem Zustand ein ^{4}He Kern (x) um einen ^{8}Be Kern (Y) kreist. Das Koinzidenz-Streuexperiment lässt sich mit ^{12}C$(e,e'\ ^{4}$He$)^{8}$Be beschreiben. Wird dabei eine Koinzidenz von Elektronen und Helium Kernen gemessen lässt sich die Anregungsenergie des

vorhergesagten Kohlenstoff Zustands nun aus der bekannten Strahlenergie, der in den Driftkammern gemessenen Elektronen Energie und der mit dem Si-Ball gemessenen Helium Energie bestimmen.

Mit dem Si-Ball lässt sich nur diese Art von Zuständen messen, jedoch keine komplette Aufspaltung von zum Beispiel ^{12}C in drei ^{4}He Kerne, da deren Energie im Bereich der von den Elektronen im Silizium deponierten Energie (etwa 100 keV bei 20 kHz) liegt und somit nicht von den Elektronen unterschieden werden kann. Die erwartete Energie bei Emission eines einzelnen ^{4}He Kerns liegt jedoch bei 5 MeV, die eines emittierten Protons bei 10 MeV mit einer Rate von jeweils etwa 1 Hz.

2 Halbleiter Detektoren

Halbleiter Detektoren nutzen in Sperrichtung betriebene Halbleiter Dioden als Kondensatoren. Durchdringt ein geladenes Teilchen den Halbleiter deponiert es seine Energie in diesem indem es den Kondensator entlädt. Dieses Spannungssignal kann mit der richtigen Elektronik aufgenommen werden um die deponierte Energie des geladenen Teilchens zu ermitteln.

2.1 Halbleiter

In Bezug auf die Leitfähigkeit unterteilt man Materialien in drei Kategorien: Leiter, Nicht-Leiter und Halbleiter. Weiterhin gibt es noch den supraleitenden Zustand, welchen man durch herabkühlen eines Materials dieser drei Typen erhalten kann. *Leiter* sind Metalle, bei denen sich bestimmte Elektronen innerhalb der Gitterstruktur frei bewegen können. Sie sind die Ladungsträger, die einen Stromfluss ermöglichen. Bei *Nicht-Leitern* gibt es keine freien Elektronen, sie können also keinen Strom leiten. Bei tiefen Temperaturen ($4 - 77\,\mathrm{K}$) können Materialien zu *Supraleitern* werden, das heißt, dass sie ihren Ohmschen Widerstand verlieren, also verlustlos Strom leiten können, und magnetische Felder aus ihrem inneren verdrängen. Ihre ideale Leitfähigkeit macht Supraleiter zu einem nützlichen Material für starke Magnete, da sie sich bei starken Strömen nicht erhitzen. So steht zum Beispiel das S in S-DALINAC für supraleitend.

Halbleiter sind bestimmte Nicht-Leiter, die durch einbringen von Fremdatomen (dotieren) leitfähig werden. [1]

2.1.1 Bändermodell

Werden n Atome nah zusammen gebracht spalten sich ihre diskreten Zustände in n dicht beieinanderliegende Unterzustände auf, deren Energieunterschied bei einer hohen Anzahl von Atomen (zum Beispiel 10^{24} Molekülen in makroskopischen Stoffmengen) so gering wird, dass man sie als ein kontinuierliches Energieband ansehen kann. Elektronen füllen diese Bänder von unten – das heißt von niedrigen zu hohen Energien – auf und können sich dann durch das Gitter bewegen, wenn sie durch ein angelegtes Feld (zum Beispiel von einer angelegten Spannung) in einen höheren Zustand übergehen können. Man nennt daher das niedrigste Band mit freien Zuständen das *Leitungsband*. Ist also das höchste von Elektronen besetzte Band – man spricht hier vom *Valenzband* – nicht komplett mit Elektronen gefüllt können diese sich bewegen und besser Strom leiten, je mehr Zustände im Valenzband noch frei sind. Es liegt also ein Leiter vor, wenn das Valenzband gleich dem Leitungsband ist.

Sind alle Zustände des Valenzbands mit Elektronen besetzt und der Abstand zum nächsten Band groß liegt ein Isolator vor.

Ist die Energielücke zwischen voll besetztem Valenz- und leerem Leitungsband jedoch gering kann ein niedriges elektrisches Feld schon ausreichen, um Elektronen aus dem Valenzband in das Leitungsband anzuheben und das Material somit leitfähig zu machen.

[1] Für Abschnitt 2.1 vgl. [1, S. 987ff].

2.1.2 Dotierung

Reicht die thermische Energie nicht aus, um Elektronen aus dem Valenzband in das Leitungsband zu heben, oder will man die Leitfähigkeit von Halbleitern verbessern, kann man dies durch einbringen von Fremdatomen erreichen, durch sogenanntes *dotieren*.

Dotiert man einen Halbleiter mit Fremdatomen, welche ein Valenzelektron mehr haben als der Halbleiter, ist dieses nur schwach an die Gitterstruktur gebunden, das heißt sein Niveau befindet sich nahe unterhalb des Leitungsbands. Da die Leitfähigkeit durch dieses zusätzliche, im Leitungsband bewegliche Elektron erreicht wird spricht man hier von einem *Donator*. Desweiteren nennt man den Halbleiter aufgrund des negativen Ladungsträgers *n-dotiert*.

Hat das Fremdatom jedoch ein Elektron weniger, entsteht ein Elektronen Loch im Gitter, oder anders gesagt gibt es weitere leere Energieniveaus nahe über dem gefüllten Valenzband. Einzelne Elektronen können dieses Loch füllen, wodurch hingegen ein weiteres Loch entsteht, da wo sich das Elektron vorher befand. Man nennt das Fremdatom hier *Akzeptor* und den Halbleiter *p-dotiert*, da man sich den Leitvorgang als ein Wandern des positiv geladenen Elektronenlochs vorstellen kann.

Durch das Einfügen von Fremdatomen wird desweiteren die Leitfähigkeit von Halbleitern verbessert, da die Wahrscheinlichkeit, dass ein Elektron mit einem Loch Rekombiniert vergrößert wird. Wegen der Energie- und Impulserhaltung kann dies nämlich nur bei einem Elektron-Loch Paar geschehen, deren Eigenschaften genau angepasst sind. Kann kein Elektron mit einem Loch rekombinieren ist dieses auch kein beweglicher Ladungsträger und trägt somit nicht zum Stromfluss bei. Durch die Dotierung werden jedoch weiter Energieniveaus eingefügt, die bei der Rekombination als Zwischenschritte verwendet werden können, wodurch die Beweglichkeit und somit auch die Leitfähigkeit erhöht wird. [2]

2.2 Halbleiter Dioden

2.2.1 np Übergang

Bringt man einen n-dotierten Halbleiter mit einem p-dotierten Halbleiter in Kontakt entsteht durch die gößere Anzahl an Elektronen im n-dotierten Teil und die niedrigere Anzahl an Elektronen im p-dotierten Teil eine Potenzialdifferenz. Es wandern sich nahe der Kontaktfläche befindliche freie Elektronen aus dem n Bereich in den p Bereich um dort die Löcher zu füllen. Der p Bereich hat nun also einen leichten Elektronenüberfluss, er ist negativ geladen, der n Bereich einen Elektronenmangel und ist somit positiv geladen. Es entsteht somit eine Art Kondensator mit einer Ladungsträger freien Verarmungszone im np Kontaktbereich. Da Halbleiter ansich keine guten Leiter sind können sich die Ladungsträger jedoch nicht weit vom np Übergang weg bewegen.

2.2.2 Strom in Durchlassrichtung

Legt man nun ein positives Potenzial an den negativ geladenen p Bereich – niedriges Potenzial – an, und ein niedrigeres Potenzial an den positiv geladenen n Bereich – hohes Potenzial –, kann man einen Stromfluss messen. Die Elektronen können aus dem p Bereich über die Anode abfließen, aus der Kathode wandern neue Elektronen in den n Bereich, aus dem Elektronen nahe des Übergangs wieder in den p Bereich wandern können um dort die Löcher zu füllen, und so weiter.

[2] Für Abschnitte 2.1.2 und 2.2 vgl. [2, S. 211ff].

2.2.3 Strom in Sperrrichtung

Polt man die angelegte Spannung nun um wird die durch den np Übergang entstandene Potenzialdifferenz weiter vergrößert. Aus dem n-dotierten Teil wandern die freien Elektronen ab, aus dem p-dotierten Teil die Elektronenlöcher. Wird die Spannung groß genug gewählt und ist die Diode nicht zu dick, kann man die in Sperrrichtung betriebene Diode als Plattenkondensator mit Plattenabstand der Dicke der Diode ansehen.

2.3 Halbleiter Detektortypen

Als einfachste Form und um das Prinzip von Halbleiterdetektoren zu verstehen wird zunächst eine in Sperrrichtung betriebene Halbleiterdiode betrachtet. Wie bereits erwähnt gibt es in dem Übergangsbereich zwischen n- und p-Teil keine freien Ladungsträger, man nennt ihn daher auch die *Verarmungszone*.

Kommt nun ionisierende Strahlung in diesen Bereich gibt sie Energie ab indem sie dort ein Elektron-Loch Paar erzeugt. Dieses wird von dem elektrischen Feld jedoch sofort aus der Verarmungszone gezogen und man kann an der angelegten Spannung einen Abfall messen. Dieser Spannungsabfall ist proportional zur von der ionisierenden Strahlung deponierten Energie. [3]

Um diese Teilchendetektoren zu optimieren werden verschiedene Herstellungsmethoden bzw. Detektortypen verwendet.

2.3.1 PIN Detektor

Ein PIN Detektor besteht aus einem p-, einem i-, und einem n-dotierten Teil. Der mittlere Teil ist mit gleich vielen Donator- wie Akzeptorteilchen dotiert, wodurch sich deren Leitungsverbessernden Eigenschaften wieder kompensieren. Die wichtigste Eigenschaft dieses Teils ist sein hoher Widerstand [2, S. 215]. Diese Anordnung ermöglicht es Detektoren von größerer Dicke herzustellen, wodurch Teilchen in ihnen beim durchdringen mehr Energie deponieren können, das heißt dass auch höher energetische Teilchen mit ihrer vollen Energie detektiert werden können.

2.3.2 Streifendetektor

Ein Streifendetektor basiert auch auf dem Diodenprinzip. Im Gegensatz zu einem PIN Detektor werden hierbei jedoch nicht nur zwei Flächen von einem n- beziehungsweise p-dotiertem Halbleiter aneinander gepresst, er besteht vielmehr aus einer beispielsweise n-dotierten Rückseite auf der mehrere p-dotierte Streifen angebracht sind. Auf diese weise kann man eine bessere Ortsauflösung erreichen als mit Detektoren größerer Fläche.

Die geringe Größe der effektiven Fläche bringt jedoch auch Nachteile mit sich. So haben Streifendetektoren im Vergleich zu Detektoren mit einer einzelnen, großen fläche im Verhältnis mehr Rand pro Fläche, wodurch sich Randeffekte mehr bemerkbar machen.

2.3.3 Guardring

An einige Detektoren wird ein Guardring um die Detektorfläche installiert. Dieser soll Randefekte vermindern, welche durch elektrische Felder aus positiven Oxiden, welche im Randbereich entstehen kön-

[3] Vgl. [2, S. 215f].

nen verursacht werden. Durch ein regelbares, entsprechend entgegengesetztes Feld, welches mithilfe des Guardrings erzeugt wird sollen diese störenden Felder aufgehoben werden.

Es hat sich jedoch gezeigt, dass auch durch einen Guardring ungewollte Effekte auftreten können [3]. So können auch dort Ionisationen durch geladene Teilchen auftreten, die einen Peak im Spektrum erzeugen können. Aufgrund der anderen physikalischen Eigenschaften im Vergleich zum eigentlichen Diodenübergang kann dieser Peak bei höheren oder niedrigeren Energien liegen.

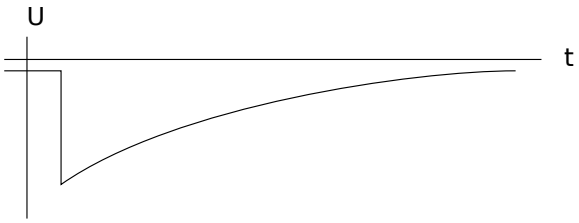
2.4 Messprinzip

Durchdringen geladene Teilchen eine Halbleiterdiode deponieren sie dort Energie in Form von Elektron-Loch-Paaren. Die so erzeugten Ladungsträger Paare sind proportional zur deponierten Energie. Sie verhalten sich in der Halbleiterdiode wie in einem Kondensator: Sie bewegen sich entsprechend des Potenzials zu entgegengesetzten Seiten der Diode. Die so auf dem Kondensator gesammelten Ladungsträger fließen dann langsam über die angelegte Spannung ab. Da die Spannung in einem Kondensator über die Kapazität proportional zur gesammelten Ladung ist

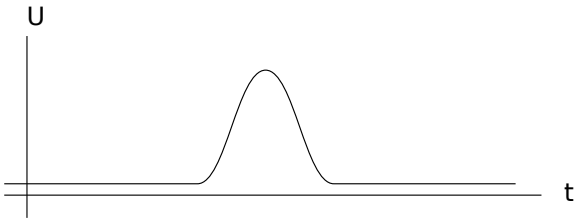
$$Q = CU \tag{1}$$

ist die Amplitude des entstehenden Spannungssignals auch proportional zur Energie. Abbildung 2(a) zeigt den Spannungsabfall eines solchen Energiesignals, wie es auch nach der Verstärkung durch einen Vorverstärker zu sehen ist.

Um das Energiesignal mit einem ADC (analog to digital converter) digitalisieren zu können muss man es zunächst in eine für den ADC verwendbare Form bringen. Dazu wird ein Spektroskopie Verstärker (SPA – spectroscopy amplifier) verwendet, welcher das Signal weiter verstärkt und in eine Gaußform bringt, dessen Amplitude wieder proportional zur Spannungsamplitude des Energiesignals und somit auch zur Energie selbst ist. Neben der Verstärkung kann hier noch eine shaping time geregelt werden, welche angibt über welche Zeit das Eingangssignal analysiert wird und unter Umständen die Form des Gaußsignals sowohl in Zeit- als auch in Spannungsrichtung strecken oder stauchen kann.



(a) Energiesignal eines Halbleiterdetektors.



(b) Gaußförmiges Signal nach einem SPA.

Abbildung 2.: Schematische Signalformen.

Am ADC wird nun die Amplitude des analogen Signals in einen digitalen Wert umgewandelt. Dabei werden kontinuierliche Spannungsbereiche diskreten Bins oder Kanälen zugeordnet, zum Beispiel erhält ein Spannungssignal, welches sich zwischen einer Spannung von 0 V und 1 V befindet den digitalen Wert 1, wenn es sich zwischen 1 V und 2 V befindet wird es zu einer 2 und so weiter.

Der ADC benötigt jedoch noch ein weiteres Signal, welches die Datenaufnahme startet und stoppt, das so genannte Gate. Nur während dieses Rechtecksignals analysiert der ADC das analoge Eingangssignal, um es nach Ende des Gates in einen digitalen Wert umzuwandeln. Das Gate wird üblicherweise von einem Gate Generator (GG) erzeugt, der einen logischen Trigger als Eingangssignal erhält, zu diesem Zeitpunkt je nach Einstellungen das Gate mit einer bestimmten Amplitude öffnet und nach einer bestimmten Zeit wieder schließt.

Das logische Trigger Signal kann durch einen Pulsgenerator erzeugt werden, in den meisten Fällen soll das Gate aber mit dem Energiesignal korrelieren, das heißt es wird ein anderes Signal verwendet, welches zeitnah zum Energiesignal ist. Dies könnte zum Beispiel bei einem Streuexperiment ein von dem gestreuten Elektron erzeugtes Signal sein, oder aber das Timing Signal des Vorverstärkers, welches nicht unbedingt porportional zu der im Detektor deponierten Energie sein muss, dafür jedoch schon etwas vor dem Energiesignal gemessen werden kann.

Um aus einem beliebig geformten Signal ein logisches Signal zu erhalten wird ein CFD (constant fraction discriminator) verwendet.

Diese vielen digitalen Energiesignale können anschließend per Software am Computer zu einem Spektrum verarbeitet werden, das heißt es wird die Anzahl aller Ereignisse für jede Energie aufsummiert und über der Energie als Histogramm aufgetragen.

2.4.1 Koinzidenzmessung

Bei einer Koinzidenzmessung werden zwei unterschiedliche, zeitnahe Signale aufgenommen und deren zeitlicher Abstand bestimmt. Alle Ereignisse, die im Rahmen einer gewissen Unsicherheit denselben zeitlichen Abstand haben werden als koinzident interpretiert, es wird also ein kausaler Zusammenhang zwischen den beiden Ereignissen erwartet. Neben den koinzidenten Ereignissen können auch zwei zufällig zeitgleiche Signale gemessen werden. Da deren zeitlicher Abstand jedoch statistisch verteilt ist wird er in einem Graphen, der die Anzahl an gemessenen Ereignissen über dem zeitlichen Abstand aufträgt die Baseline nach oben verschieben.

Analog zum ADC kann hierfür ein TDC (time to digital converter) verwendet werden. Er weist der Zeitspanne zwischen Start des Gates und Beginn des Zeitsignals (common start) bzw. dem Beginn des Zeitsignals und dem Ende des Gates oder eines Stop-Triggers (common stop) einen digitalen Wert zu.

Ein Schaltplan für eine Koinzidenzmessung am QCLAM-Spektrometer ist in Abbildung 3 dargestellt.

Das wichtigste bei einer Koinzidenzmessung, was jedoch auch bei einer Single-Spektren Messung nicht zu vernachlässigen ist, ist das zeitliche abpassen aller Signale. So sollte man zum Beispiel sicher gehen, dass der Stop-Trigger immer nach den Zeitsignalen kommt, falls man den TDC im common stop Modus betreibt oder dass das Energiesignal auch im Gate liegt und nicht erst danach. Die beiden Signale der koinzidenten Ereignisse können auch zeitlich weit auseinander liegen, da zum Beispiel bei einem Streuexperiment von Elektronen an einem Target die am Target separierten Teilchen direkt gemessen werden, die Elektronen jedoch erst noch einen weiteren Weg von einigen Metern zurück legen müssen. Außerdem sollte man darauf achten, dass durch die Schaltung keine Selbstkoinidenz erzeugt wird, das heißt, dass zum Beispiel der TDC durch ein Signal gestartet und durch die Verzögerung dieses Signals wieder gestoppt wird.

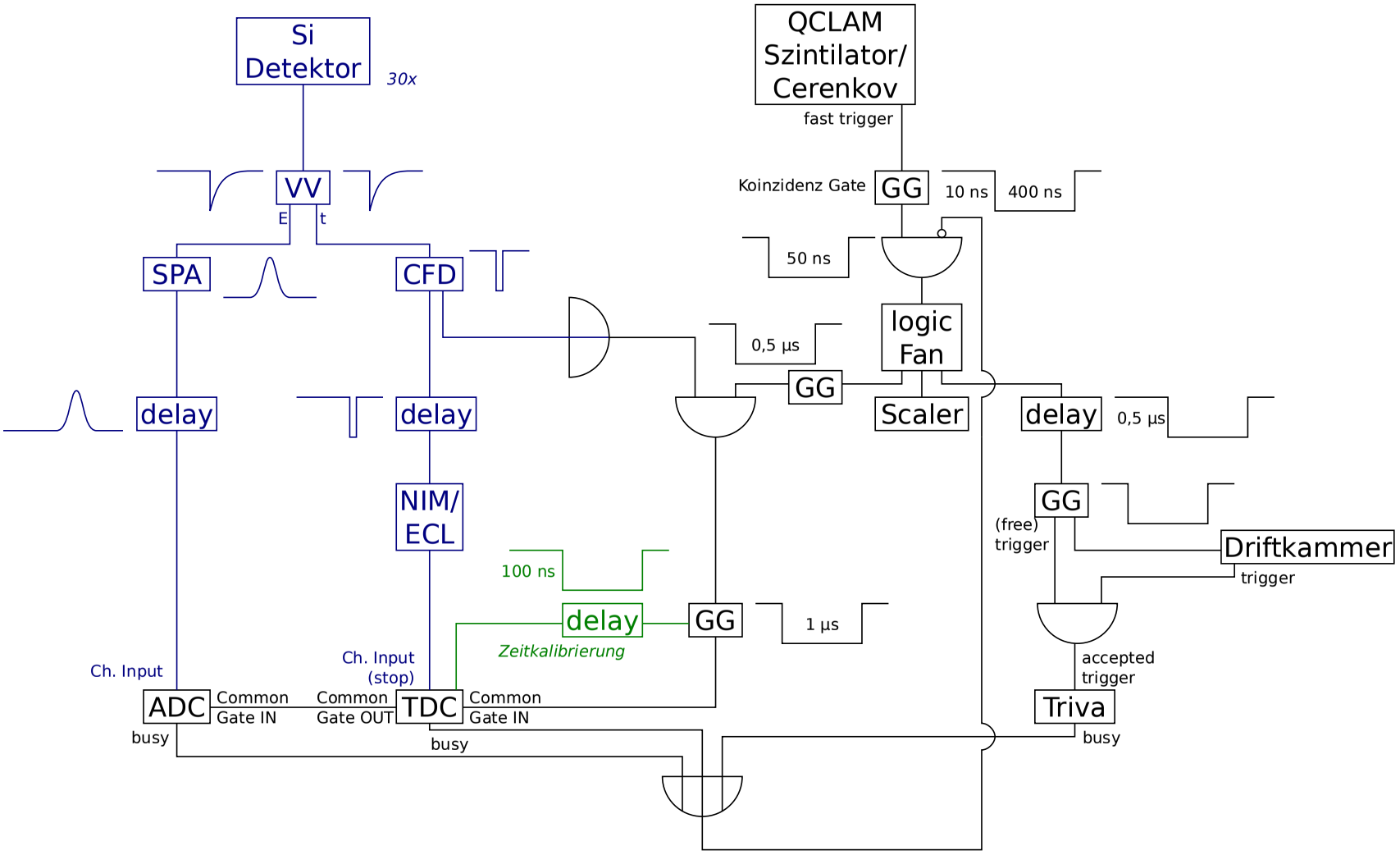


Abbildung 3.: Schaltplan für eine Koinzidenzmessung zwischen dem QCLAM-Spektrometer und dem Si-Ball, entwickelt mit Hilfe von [4]. Der blaue Abschnitt ist für jeden Detektor am Si-Ball vorgesehen, derzeit sind das mit dem Streifendetektor 35. Delays und Gatelängen sind grobe Schätzungen. Das Verwerfen einer Messung durch die Driftkammer muss noch durch eine logische Verschaltung ermöglicht werden.

3 Aufbau

Im Ramen der Bachelor Arbeit wurden die drei vorhandenen Detektorarme mit je zehn Silizium PIN Detektoren bestückt. Diese wurden mit entsprechenden Kabeln versehen und das ganze zusammen mit einem Silizium Streifendetektor an einer eigens dafür entworfenen Halterung befestigt, welche in die Streukammer am QCLAM-Spektrometer gestellt werden kann. Die Signale der Detektoren werden in den Messraum geleitet, in dem sie von der Ausleseelektronik verarbeitet und durch eine Software ausgelesen wird, um sie dann an einem Computer als Spektren darstellen zu können.

3.1 Verwendete Detektoren

Für das Detektorsystem sind Silizium PIN Detektoren vorgesehen, die sich in einem Ball angeordnet rund um das Target befinden, daher der Name *Si-Ball*. Sie sollen ortsaufgelöst die Energie von im Target separierten Protonen, Deuteronen und α-Teilchen messen, und aus dem zeitlichen Abstand dieser Signale das Ereignis auf eine Koinzidenz mit der Streuung eines Elektrons überprüfen.

Der Ball ist aufgeteilt in Abschnitte, welche aus drei Armen bestehen, auf denen jeweils bis zu zwölf Detektoren in drei zueinander gebogenen Segmenten mit je vier Detektoren Platz haben. Für die erste Inbetriebnahme wurde zunächst ein Abschnitt mit 30 Detektoren bestückt. Auf den jeweils zwei äußersten Plätzen befinden sich keine Detektoren.

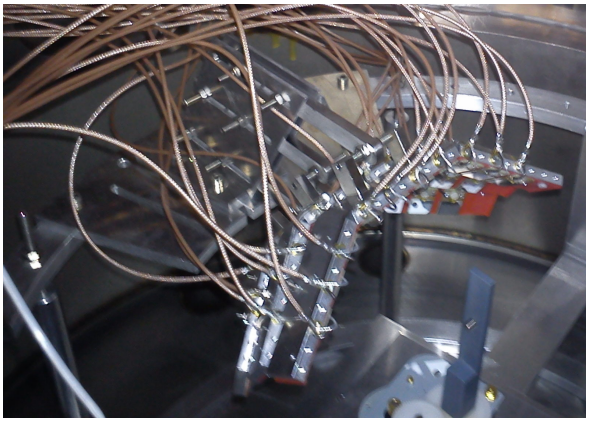


Abbildung 4.: Verkabelte PIN Detektoren (nummeriert von rechts nach links) befestigt an drei Detektorarmen (nummeriert von oben nach unten). Unten rechts ist die Halterung für die α-Quelle zu sehen.

Desweiteren sind zwei Silizium Streifendetektoren vorhanden um die Möglichkeit und Notwendigkeit einer besseren Ortsauflösung zu ermitteln.

3.1.1 Silizium PIN Detektoren

Die PIN Dioden haben eine effektive Fläche von $(89,54 \pm 0,98)\,\mathrm{mm}^2$ und eine Dicke von etwa $300\,\mu\mathrm{m}$. Daraus ergibt sich nach

$$C(d) = \frac{100\,\mathrm{pf}}{mm}\frac{A}{d} \tag{2}$$

für Silizium Dioden mit hoher Sperrspannung [5, S. 15] eine Kapazität von $(29,8 \pm 0,4)\,\mathrm{pF}$.

Mit Hilfe der Formel

$$U = \frac{Q}{C} \tag{3}$$
$$= \frac{n_e e}{C} \tag{4}$$
$$= \frac{E}{E_i}\frac{e}{C} \tag{5}$$

kann die erwartete Spannung am Vorverstärker berechnet werden, wobei n_e die Zahl der erzeugten Elektron-Loch-Paare ist, e die Elementarladung, E die vom Teilchen im Detektor deponierte Energie und $E_i = 3,6\,\text{eV}$ [6, S. 296] die Ionisierungsenergie von Silizium. In Tabelle 1 sind die verwendeten und ermittelten Größen dargestellt. Durch die Verstärkum am Vorverstärker ist also mit einer Spannung von einigen 10 mV zu rechnen.

Teilchen	E / MeV	Rate	Q / e	U / mV
Elektronen	0,1	20kHz	$2,78 \times 10^4$	0,15
α	5	1 Hz	$1,39 \times 10^6$	7,47
Proton	10	1 Hz	$2,78 \times 10^6$	14,95

Tabelle 1.: Erwartete Spannung, die am VV verstärkt wird. Da es sich bei deponierter Energie und Rate um Schätzwerte handelt wurde auf eine Unsicherheitsbetrachtung verzichtet.

An den Detektoren sind Koaxial Kabel mit dem für positive Spannung vorgesehen Innenleiter am n-dotierten Teil und dem für die Erdung bestimmten Außenleiter am p-dotierten Teil fest gelötet. Für eine gemeinsame Erdung sind die Außenleiter jeweils mit einer Brücke an dem Metallteil des Detektorarms befestigt, die für Kontakt sorgende Klemme dient gleichzeitig als Zugentlastung für die Kabel. Abbildung 4 zeigt ein Foto des verkabelten Aufbaus in der Streukammer.

In vorherigen Messungen wurde für alle verwendeten Detektoren eine optimale Betriebsspannung von 150 V ermittelt. Dabei wurde an einigen Detektoren ein Strom von bis zu 3 μA nach wenigen Stunden Betrieb festgestellt. Erwartete Ströme liegen im Bereich von 40 nA.

3.1.2 Silizium Streifendetektoren

Die Streifendetektoren bestehen aus einer p-dotierten Rückseite auf der acht n-dotierte streifen angebracht sind. Um die Detektorfläche herum befindet sich ein ebenfalls n-dotierter Guardring.

Der Detektor wird über acht Koaxialkabel mit Spannung versorgt. Dabei liegt an den miteinander kurzgeschlossenen Innenadern eine poitive Spannung an, die individuellen Außenleiter liefern das Signal der jeweiligen Streifen. Die Kabel sind über einen Pfostenstecker mit dem Detektor verbunden. Die Pinbelegung ist in Abbildung 6 dargestellt.

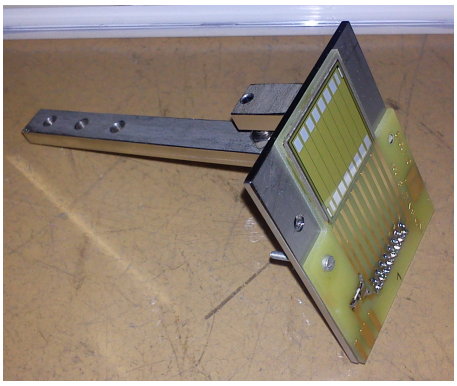


Abbildung 5.: Der Quelle bzw. dem Target zugewandte Seite des Streifendetektors.

Damit der Streifendetektor so auf der Halterungsplatte angebracht werden kann, dass die Detektorfläche zur Quelle bzw. zum Target zeigt wurde eine 30 mm × 10 mm Aussparung für den Pfostenstecker auf der Rückseite des Detektors in die Platte gefräst.

Genau wie die PIN Detektoren werden können die Streifendetektoren somit mit einer positiven Spannung von 150 V versorgt werden.

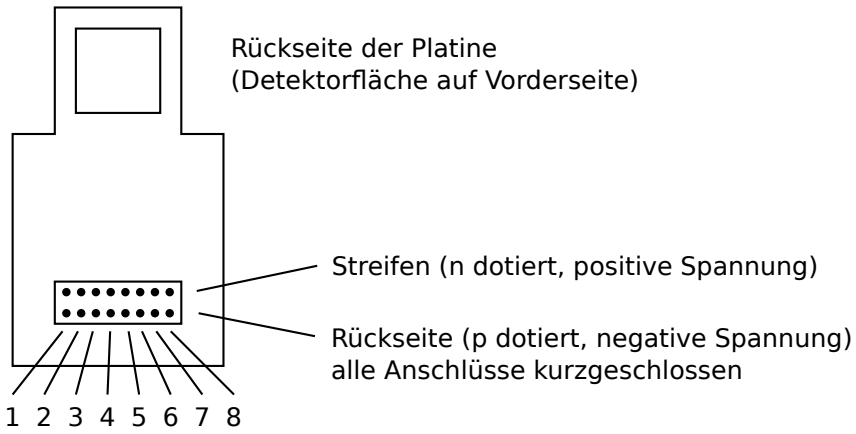


Abbildung 6.: Pinbelegung der Streifendetektoren. Die Brücke zum Guardring wurde abgetrennt, da dieser nicht verwendet wird.

3.2 Verwendete Vorverstärker

Abbildung 7.: CSTA2 Vorverstärker. Die linke Seite wird direkt auf den Deckel der Streukammer gesteckt. Rechts befinden sich HV-BNC Anschluss für die HV-Versorgung und Pins für Signale und die Spannungsversorgung des VVs.

Als Vorverstärker kommen CSTA2 Modelle mit etwa zweifacher Verstärkung zum Einsatz. Sie leiten die über einen HV-BNC Stecker angelegte Hochspannung von 150 V an die Detektoren weiter. Über den separaten Pin-Anschluss werden sie selbst mit Spannung versorgt und geben Energie und Timing Signal aus. Desweiteren besteht die Möglichkeit ein Detektorsignal zu simulieren indem an den Testeingang ein kurzer Rechteckpuls angelegt wird.

Die Amplitude des Energiesignals ist proportional zu der im Detektor deponierten Energie. Das schnellere Timing Signal wird bereits 60 ns vor dem Energie Signal ausgegeben und kann als Trigger verwendet werden.

Die Vorverstärker wurden jeweils dem Detektor zugeordnet bei dem bei vorherigen Tests [7] eine bestmögliche Funktionalität ermittlet wurde. Abschnitt A.1 im Anhang stellt diese Zuordnung inklusive der gemessenen Ströme in einer Tabelle dar.

3.3 Befestigung in der Streukammer

Auf den drei bereits vorhandenen Detektorarmen sind jeweils zehn Silizium PIN Detektoren befestigt. Sowohl Detektoren als auch Detektorarme sind nummeriert, die Kabel am Deckel nach Schema in Abbildung 10 befestigt.

Die drei Detektorarme befinden sich zusammen mit einem Streifendetektor und einer Halterung für eine α-Quelle auf einem Aluminium-Ring, dessen Maße dem Ring im Goniometer nachempfunden sind, auf dem die Detektoren bei Messungen mit Elektronenstrahl befestigt werden sollen. Der obere Ring des Gestells hat zwei gegenüberliegende, jeweils 5 cm breite Aussparungen, so dass auch diese Halterung im Betrieb mit dem Elektronenstrahl verwendet werden kann, ohne dass unerwünschte Sekundärstrahlung entsteht. Die Höhe des oberen Rings kann über Muttern angepasst werden, konzipiert wurde die Höhe so, dass die Oberseite der Halterungsplatten der Detektorarme sich auf halber Höhe der Streukammer befindet. Trotz dieser fixen Befestigung haben die Kabel der Detektoren eine ausreichende Länge, so dass sie auch mit dem Goniometer verwendet werden können, wenn dieses sich um bis zu 50° in beide Richtungen dreht. Abbildungen 8 und 9 zeigen ein Bild und die technischen Zeichnungen dieser Halterung.

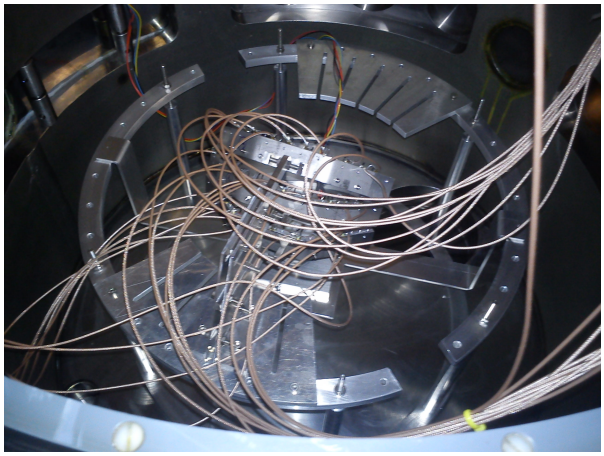


Abbildung 8.: Teil des Si-Balls befestigt an der im Rahmen dieser Bachelor Thesis entwickelten Halterung in der Streukammer am QCLAM-Spektrometer.

Die Kabel werden jeweils über BNC Durchführungen durch den Deckel der Streukammer geleitet, an dessen anderem Ende direkt die Vorverstärker befestigt sind.

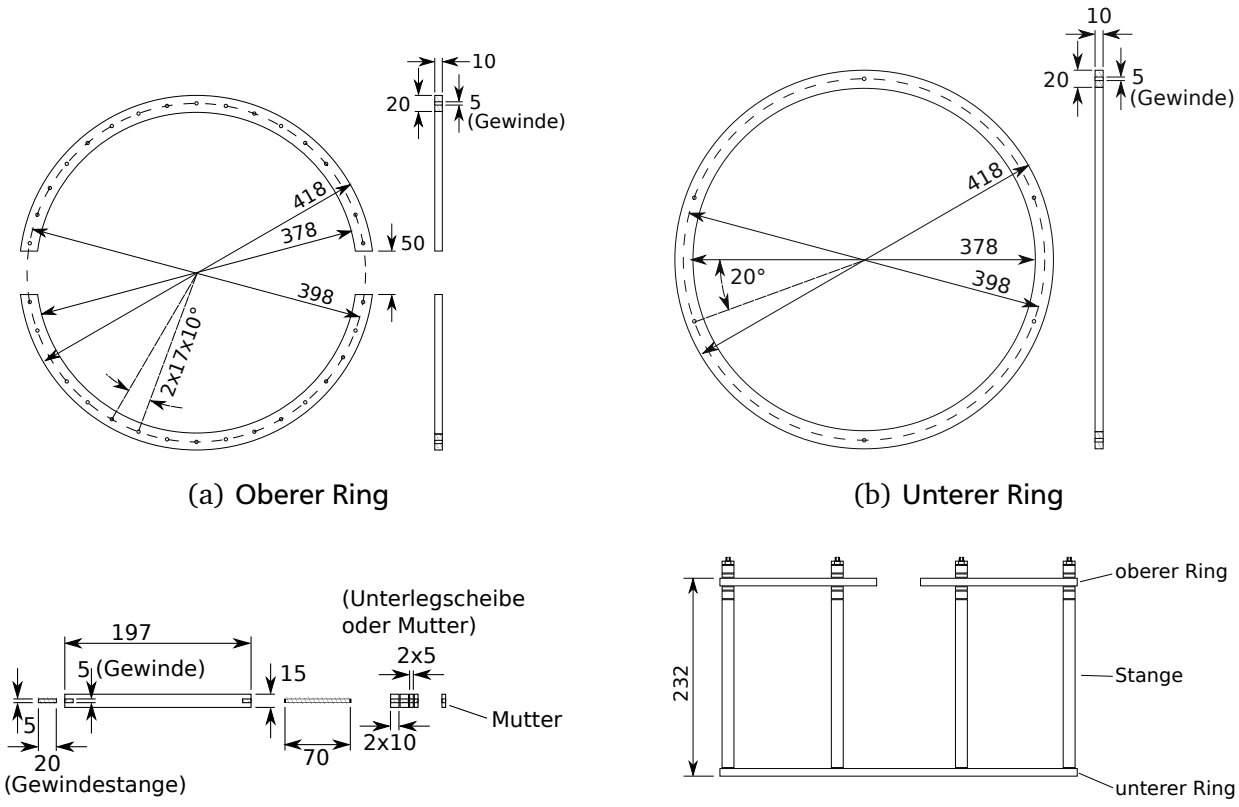
3.3.1 ^{241}Am Quelle für die Testmessung

Um mit den Detektoren Spektren aufnehmen zu können ohne dafür einen Elektronenstrahl und das gesamte Koinzidenzexperiment zu benötigen wird ein ^{241}Am α-Quelle verwendet. Die am häufigsten auftretende α-Strahlung dieses Isotops hat Energien von $(5485,56 \pm 0,12)$ keV mit $(84,8 \pm 0,5)$ % Anteil, $(5442,80 \pm 0,13)$ keV mit Häufigkeit von $(13,1 \pm 0,3)$ % und 5388,56 keV mit $(1,660 \pm 0,020)$ %, außerdem wird Röntgen- beziehungsweise γ-Strahlung mit Energien von 13,9 keV zu (37 ± 3) % und $(59,5409 \pm 0,0001)$ keV mit $(35,9 \pm 0,4)$ % emittiert [8].

Die Quelle befindet sich in einem Messingzylinder, welcher ein M4 Gewinde auf der Rückseite hat. Darüber ist er an einer mit einem Motor drehbaren Halterung befestigt. Diese kann über ein mit Processing geschriebenes Programm gesteuert werden, welches im Rahmen einer Miniforschung entwickelt wurde [9]. Der Schrittmotor kann die Quelle mit $0,67\,^{\circ}/_{\text{Schritt}}$ relativ zur aktuellen oder zur Startposition gegen den Uhrzeigersinn rotieren. Dadurch können alle drei Segmente der Detektorarme direkt mit α-Partikeln bestrahlt werden.

3.4 Verkabelung in der Messhalle

Das Detektorsystem befindet sich wie für das spätere Experiment vorgesehen in der Streukammer am QCLAM-Spektrometer in der Messhalle des Elektronenbeschleunigers, welche auf Abbildung 11(a) zu



(a) Oberer Ring

(b) Unterer Ring

(c) Stange (6x)

(d) Gesamt

Abbildung 9.: Technische Zeichnung der Halterung des Si-Balls. Größenangaben in mm. Nicht maßstabsgetreu.

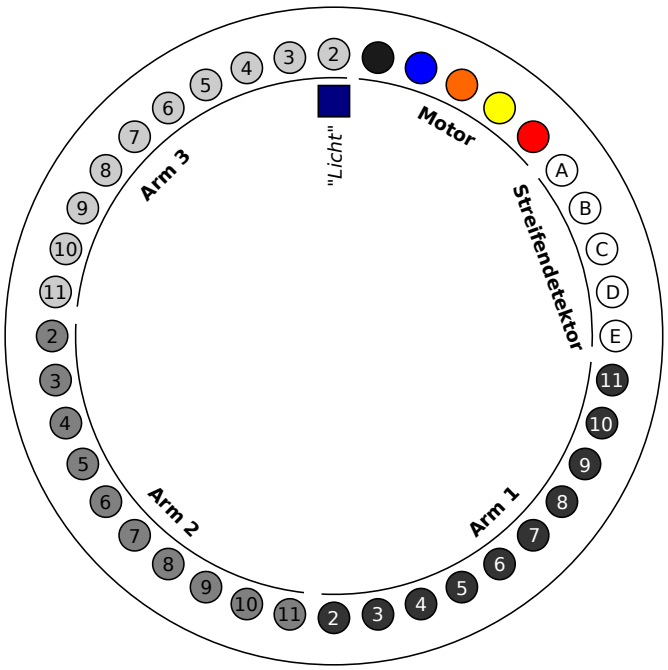
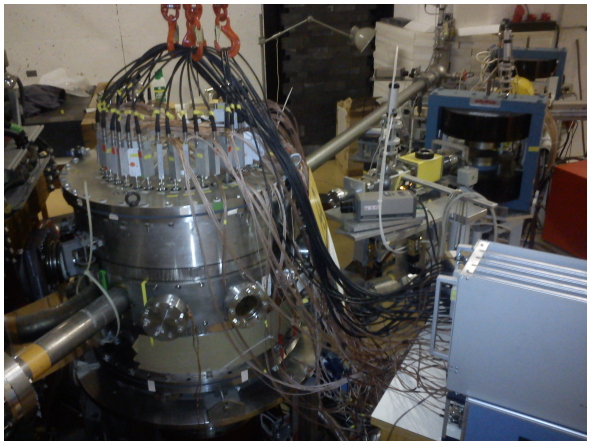
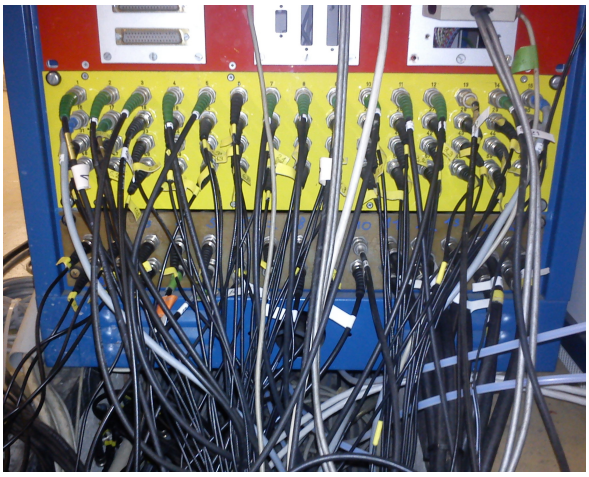


Abbildung 10.: Zuordnung der Deckel-Durchführungen zu Detektoren.

sehen ist. Für die Verkabelung wurden im Wesentlichen bereits vorhandene Kabel verwendet, welche in einem Kabelkanal von der Streukammer zur Hochspannungsversorgung (HV-Versorgung) und zu den Durchführungen zum Messraum liegen.



(a) Verkabelung an der Streukammer. Auf dem Deckel sind die Vorverstärker zu sehen, welche von den Spannungsquellen unten links im Bild versorgt werden. Die schwarzen Kabel führen zur HV-Versorgung und sind zwecks Zugentlastung am Kran aufgehängt.



(b) Gelbes und braunes Shielding der Durchführung aus der Messhalle zum Messraum.

Abbildung 11.: Streukammer und Durchführung in der Messhalle.

Die direkt auf den Durchführungen im Deckel der Streukammer angebrachten Vorverstärker werden über zwei Spannungsquellen versorgt, welche sich auf einem Tisch neben der Streukammer befinden. Energie und Timing Signal werden zu den zwei Durchführungsmodulen – eins mit gelbem, eins mit braunem Shielding – zum Messraum geleitet.

Die Hochspannungsversorgung wird von einem *Caen Mod. SY2527* Crate mit zwei *Caen Mod. A1512* Modulen und einem *Caen Mod. A1733P* Modul bedient. Die Module wurden so konfiguriert, dass sie die Detektoren mit 150 V versorgen, die über $1\,{}^{\mathrm{V}}/_{\mathrm{s}}$ hoch und und runtergefahren werden. Falls ein größerer Strom als $3,5\,\mu$A länger als 1 s durch die Detektoren fließt wird die Spannung heruntergefahren um die Detektoren zu schützen. Die entsprechenden Einstellungen sind in Tabelle 2 dargestellt.

V_0 set	I_0 set	Trip	$SV_{\max}$	V_1 set	I_1 set	RDown	TripInt	TripExt	RUp	PwDown
150 V	$3,5\,\mu$A	1 s	150 V	0 V	$0\,\mu$A	$1\,{}^{\mathrm{V}}/_{\mathrm{s}}$	0	16	$1\,{}^{\mathrm{V}}/_{\mathrm{s}}$	Ramp

Tabelle 2.: Einstellungen der HV-Versorgung.

Da sich das HV-Crate beim einschlaten nicht konsistent verhalten hat wurde es nach erfolgreichem einschalten nicht mehr ausgeschaltet. Die Probleme beinhalteten, dass sich das Crate beim Hochfahren aufhängte oder nach dem Anmelden keins der angeschlossenen Module anzeigte. Die besten Chancen, das Crate erfolgreich hochzufahren erreichte man indem man nach dem Einschalten fünf Minuten wartete, ohne eine Taste zu drücken und danach, sofern der Info-Bildschirm zu sehen war eine der Pfeil Tasten drückte um zum Login-Bildschirm zu gelangen.

Für eine genaue Zuordnung Detektor–Kabel–HV-Versorgung–Durchführung siehe Anhang A.1 und A.2. Es wurden hier Kabel aus alten Experimenten verwendet, die bereits in den Kabelkanälen lagen. Aufgrund

der unterschiedlichen Arten von Kabel (Abschirmung, Leiterdurchmesser, Länge) können die Signale aus den Vorverstärkern unterschiedlich beeinflusst werden, zum Beispiel durch unterschiedliche Dämpfungen.

3.5 Ausleseelektronik

Das Aufnehmen und Digitalisieren der Detektorsignale wurde durch NIM und VME Module bewerkstelligt. Dabei handelt es sich um standartisierte Crates, welche Module im Fall des NIM Standards mit Spannung versorgen, beim VME Standard zusätzlich noch Kommunikation unter den Modulen über einen Bus bereitstellen. Zunächst wurden die Signale über die NIM Moule verarbeitet um aus den Timing Signalen einen Trigger und ein Gate für die Datenaufnahme zu generieren, was im Experiment aus den im QCLAM-Spektrometer aufgenommenen Elektronensignalen erhalten werden soll. Die Timing Signale werden dann für die Flugzeitbestimmung verwendet um echte Koinzidenzen von zufällig gleichzeitig auftretenden Ereignissen zu trennen.

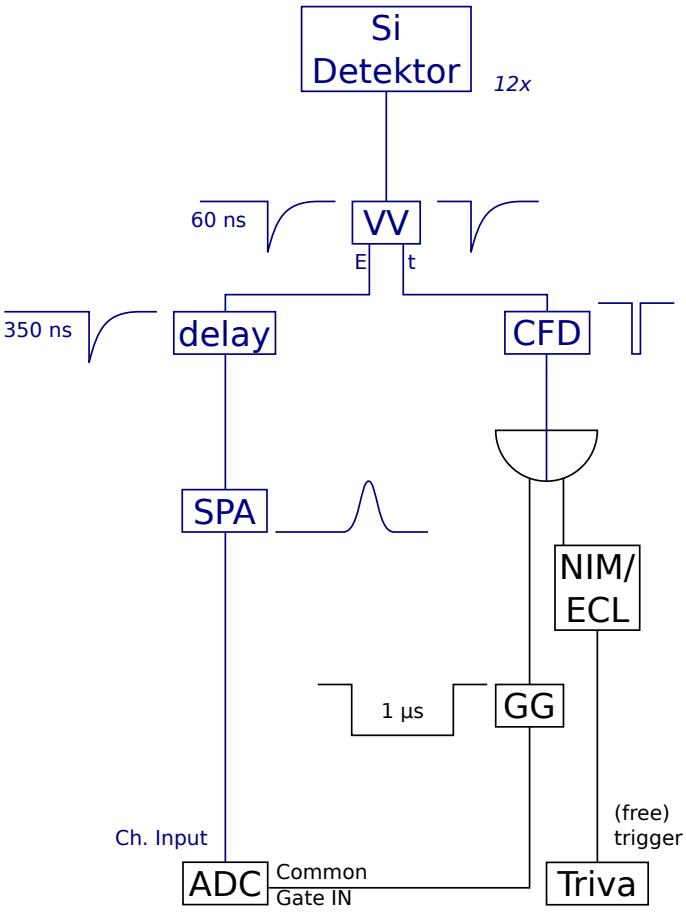


Abbildung 12.: Schaltplan der Ausleseelektronik für die Messung von Spektren.

Wie bereits in Abschnitt 2.4 beschrieben werden die Timing Signale zunächst mit einem CFD in logische Signale umgewandelt. Hierfür werden ein *Caen Mod. N843* 16 Channel CFD und ein *CF 8201* 8 Channel CFD der GSI verwendet. Anschließend werden die Signale in einem *CO 4010* Koinzidenzmodul der GSI geodert, eines der zwei Ausgangssignale wird mit Hilfe eines *Caen Mod. N638* NIM/ECL-ECL/NIM Konverters von NIM- in ECL-Logik umgewandelt und als Trigger an ein von der GSI gefertigtes Trigger Modul Triva 3 geleitet, das andere erzeugt an einem *GG 8000* 8 Channel Gate Generator der GSI das Gate für den *Caen Mod. V785 AC* 32 Channel Peak Sensing ADC.

Da es bis zum Erzeugen des Gates im Wesentlichen wegen dem CFD und dem Odern der Signale mitunter über 60 ns dauern kann – diese Zeit wurde als ungefährer Abstand zwischen Energie- und Timing-Signal bestimmt – werden die Energie Signale in einer Delaybox mit zwölf Kanälen zunächst um jeweils etwa 350 ns verzögert. Sie werden dann an einen *Caen Mod. N568B* 16 Channel SPA gegeben. Der SPA kann über einen Computer programmiert werden, als Einstellung wurden hier die Werte in Tabelle 3 verwendet. Die Signale aus dem SPA werden mit einem Flachbandkabel zum ADC weitergeleitet.

Fine Gain	Coarse Gain	PoleZero Adj.	Shaping Time	Output Polarity	Output Config.
10	5	255	1	positive	inverted

Tabelle 3.: Einstellungen für alle 16 Kanäle des SPA. Aus einem negativen Input wird ein positiver Output.

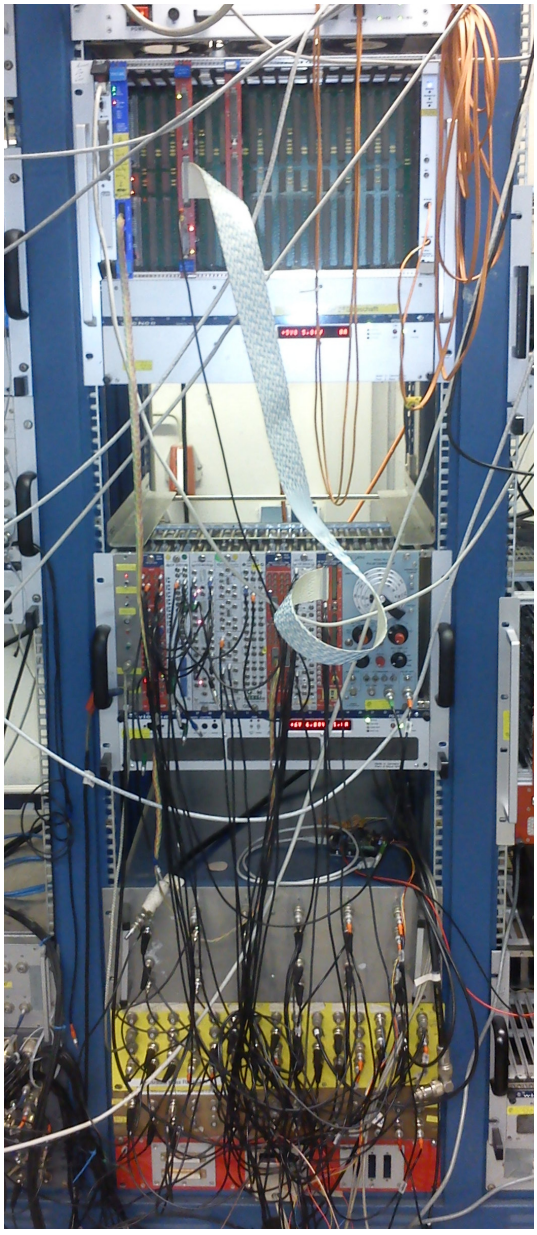


Abbildung 13.: Module im Messraum (von unten nach oben): Braune und gelbe Durchführung; silberne Delay Box; NIM Crate (von links nach rechts): nicht verwendet, 16Ch CFD, 8Ch CFD, Oder Modul, nicht verwendet, GG, NIM/ECL Konverter, nicht verwendet, SPA, Pulser; VME Crate (von links nach rechts): Rio, Triva, ADC, TDC.

4 Messung

4.1 Auslese Software

Um eine Messung mit dem Detektorsystem durchführen zu können benötigt man noch eine Methode mit der man die Messung steuern kann. Hierfür wird das an der Gesellschaft für Schwerionenforschung (GSI) entwickelte Multi Branch System (MBS) verwendet. Es dient dazu ADC und TDC zu initialisieren und konfigurieren, Aufnahmen zu starten und stoppen, sowie die Daten in eine Datei zu speichern. Diese kann anschließend ausgelesen werden um die Daten in eine darstellbare Form zu bringen, in der sie dann analysiert werden können.

4.1.1 Multi Branch System

In dem Crate, in dem sich ADC und TDC befinden sind desweiteren das Trigger Modul *Triva* und das Controller Modul *Rio3*. Auf dem Rio läuft eine virtuelle Maschine in der wiederum MBS läuft, welches die Datenaufnahme steuert. Sobald die Triva ein Trigger Signal erhält wird dieses an die Rio geleitet und dort von MBS interpretiert um zum Beispiel den Datenspeicher des ADC auszulesen.

MBS wird über Befehle in einer Kommandozeile ausgeführt und gesteuert. Das verarbeiten von Trigger Signalen wird in C programmiert und in der Datei `f_user.c` abgelegt. Aus dieser Datei wird mit dem richtigen Makefile und dem Befehl `make` das Programm `m_read_meb` angelegt, welches aus MBS ausgeführt werden kann.

Wurde MBS mit dem Befehl `mbs` gestartet und sind darin alle dienste gestartet – üblicherweise über ein Skript, welches mit dem Befehl `@startup` ausgeführt werden kann – kann die Datenaufnahme mit dem Befehl `start acquisition` oder kurz `sta acq` gestartet werden. Dies führt die Funktion `f_user_init` aus, in der der ADC konfiguriert und sein bereits vorhandener Speicher gelöscht wird. Die Funktion `f_user_readout` wird mit Trigger Typ 14 aufgerufen, welcher den Start der Datenaufnahme kodiert.

Alle nun folgenden Trigger Signale, welche die Triva erhält verursachen nach einer Wartezeit den Aufruf von `f_user_readout`, wobei der übergebene Trigger Typ durch die vier Trigger Eingänge der Triva binär kodiert ist. Die Wartezeit kann über den Parameter `TRIG_CVT` (trigger conversion time) in 100 ns Schritten konfiguriert werden. Der Parameter wird in der Datei `setup.usf` eingetragen, welche beim Start des entsprechenden Dienstes (siehe Anhang Abschnitt A.5) eingelesen wird. Die minimale Zeit erhält man aus der Dauer des Gates und der Zeit, die die Module zum digitalisieren des Signals benötigen. Für den Caen ADC sind dies $6\,\mu s$ Gate plus $(6-10)\,\mu s$ Konvertierzeit.

Mit `stop acquisition` beziehungsweise `sto acq` wird die Messung gestoppt. `f_user_readout` wird noch einmal mit Trigger Typ 15 aufgerufen, anschließend werden keine weiteren Trigger angenommen. Nach dem Beenden aller Dienste – mit dem Befehl `@shutdown` – wird die Datei, welche die Daten enthält abgeschlossen und MBS kann wieder beendet werden.

4.1.1.1 f_user.c

Für die Messungen wurde eine bereits vorhandene Datei [10], welche mehrere ADC-785 und TDC-775 von Caen auslesen kann an die Bedürfnisse an die Messungen mit dem Si-Ball angepasst. Sie ist in

der Lage bei einem erhaltenen Trigger sowohl den Speicher des ADC als auch des TDC auszulesen und hintereinander in eine Listmode Datei zu schreiben.

Hierfür sind Funktionen vorgesehen, die die wichtigsten Einstellungen an den Modulen vornehmen können, indem sie in die entsprechenden Register schreiben.

Nachdem zunächst das Gate des ADC invertiert war, sodass er immer dann zum konvertieren von Signalen aktiviert war, wenn kein Signal anlag und somit der Speicher sehr schnell voll gelaufen ist, hat es sich als sinnvolle Konfiguration ergeben bei jedem Trigger den gesamten Speicher des ADC auszulesen, sofern dieser nicht tatsächlich mit dem Aufnehmen oder Konvertieren von Signalen beschäftigt ist. Zu beachten ist hier, dass der ADC bei vollem Speicher das Busy-Bit auf 1 setzt, sodass diese Situation als Ausnahme berücksichtigt werden muss.

In aktueller Konfiguration wird bei jedem Trigger zunächst überprüft ob Daten vorliegen. Falls dies der Fall ist und der ADC entweder nicht beschäftigt ist oder der Speicher voll ist wird der komplette Speicher in 32-Bit Datenwörtern ausgelesen, so lange Daten bereit liegen. Zu beachten ist, dass bei vorliegendem ADC der Zeiger über den auf die Daten zugegriffen wird nach einem Lesezugriff automatisch auf das nächste Datenwort geschoben wird. Die gelesen Daten werden hintereinander in eine Listmode Datei geschrieben.

4.1.2 GSI EventAPI

In der Listmode Datei liegen die aus dem ADC gelesenen Daten binär in aneinander geketteten 32-Bit Wörtern vor. Im Falle des ADC enthalten diese Datenwörter neben den konvertierten Amplituden des gemessenen Signals auch weitere Informationen, wie zum Beispiel eine Adresse des ADC, den Typ des Datenworts – Header, Daten oder End Of Block (EOB) – oder ob der Wert über einem Schwellwert liegt. Diese Datenwörter befinden sich innerhalb weiteren, von MBS erstellten Strukturen mit Informationen über das jeweilige Event. Um die Datenwörter aus der Listmode Datei zu lesen und mit möglichst wenigen Einschränkungen weiter verarbeiten zu können wurde ein Programm basierend auf der GSI EventAPI geschrieben, welche eine Schnittstelle zwischen C und dem Listmode Format bietet. [4]

Das Programm `listmode2plot` erwartet als Parameter den Pfad und Dateinamen der Listmode Datei und liest aus dieser die von MBS in Events zusammengefassten Datenwörter. Falls das Datenwort tatsächlich Daten enthält, also weder Header, EOB, ungültig oder eine Information aus der Datenaufnahme ist wird der konvertierte Wert extrahiert und einer internen Histogrammstruktur, welche aus einer doppelt verketteten Liste besteht hinzugefügt. Da bei größeren Dateien die Umwandlung mitunter bis zu einer Stunde dauern kann werden die nach konvertiertem Wert sortierten Histogramme nach jeweils 100 000 gelesenen Events in von Gnuplot lesbare Textdateien geschrieben. Dies ermöglicht es sich einen ersten Eindruck zu verschaffen, desweiteren sind somit darstellbare Spektren vorhanden, selbst wenn das Programm abstürzen sollte, weil die Listmode Datei zum Beispiel nicht korrekt abgeschlossen wurde. Letzteres kann zum Beispiel passieren, wenn man in MBS vergisst das Shutdown-Skrip auszuführen, welches den Befehl `close file` enthält.

Um bei großen Messungen den Konvertiervorgang zu beschleunigen wurde die Ausgabe von nicht-Datenwörtern – diese enthalten Header, EOB, ungültige Daten oder Informationen über den Status des ADC bei Erhalten eines Triggers – durch auskommentieren der jeweiligen Zeilen deaktiviert. Die Ausgabe der EventAPI kann zum Beispiel durch

```
listmode2plot data.lmd > log.txt
```

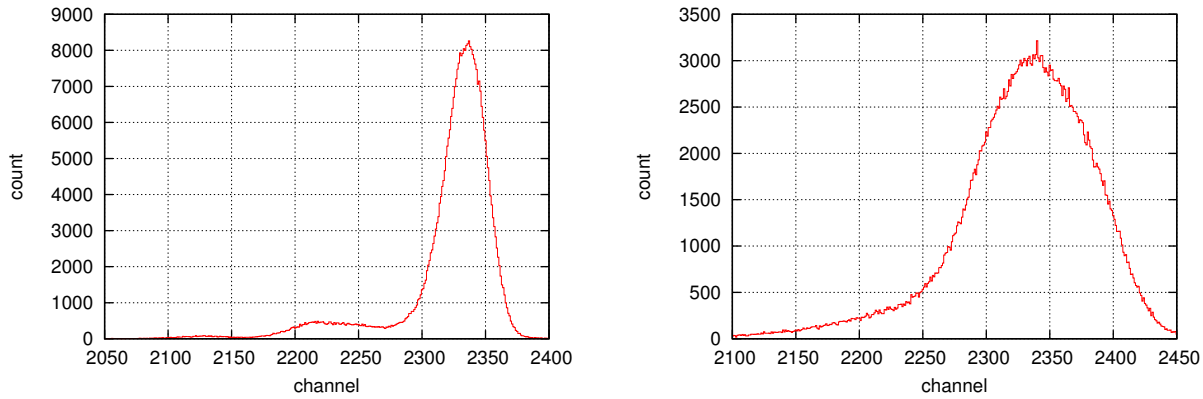
[4] Um die GSI Event API kompilieren zu können wurde in Zeile 1285 der Datei f_evt.c eine Abfrage auf true gesetzt. Das Problem weswegen diese Zeile nicht kompiliert werden konnte tritt abhängig vom Betriebssystem auf. Die Abfrage ist nur relevant falls eine zu öffnende Datei nicht geöffnet werden kann.

in eine Datei umgeleitet werden.

4.2 Gemessene Spektren

Gemessen wurden mehrere Energiespektren um die Einsatzfähigkeit des Si-Balls zu testen und mögliche Einflüsse unter den Detektoren beziehungsweise Defekte oder Probleme mit ihnen herauszufiltern. Hierzu wurde mit einzelnen Detektoren gemessen, mit nebeneinanderliegenden, mit möglichst vielen parallel, desweiteren wurden Spektren von Detektoren ohne anliegende Sperrspannung aufgenommen. Die wichtigen Ausschnitte aller Spektren sind in Anhang A.10 dargestellt.

4.2.1 Spektren einzelner Detektoren



(a) Detektor 3.6 (b) Detektor 3.3

Abbildung 14.: Einzeln über 1 h gemessene Detektoren.

Von den Energien und deren Wahrscheinlichkeiten im α Spektrum des ^{241}Am erwartet man Ergebnisse wie das in Abbildung 14(a) dargestellte Spektrum. Ein deutlicher Peak mit 5,49 MeV – hier in etwa bei Kanal 2340 –, einen deutlich schwächeren Peak mit 5,44 MeV – hier in etwa bei Kanal 2220 – und einen gerade noch erkennbaren Peak mit 5,39 MeV – hier in etwa bei Kanal 2130. Einige Detektoren zeigten jedoch nur einen Peak im Spektrum wie in Abbildung 14(b).

Als mögliche Ursache konnte die Verwendung des Timing Signals als Trigger herausgefiltert werden. Es scheint bei Ereignissen mit niedrigeren Energien eine andere Form oder Amplitude zu haben, welche vom CFD nicht mehr in ein logisches Signal umgewandelt werden kann. Bei einigen Detektoren ließ sich der Schwellwert am CFD nicht weiter optimieren, da er sonst bereits im Rauschen triggern würde. Dies liegt daran, dass einige Vorverstärker ein positives Timing Signal ausgeben, der CFD jedoch nur für negative Signale ausgelegt ist und man somit auf ein leichtes Überschwingen des Signals angewiesen ist, welches bei niedrigeren Energien zu gering ist.

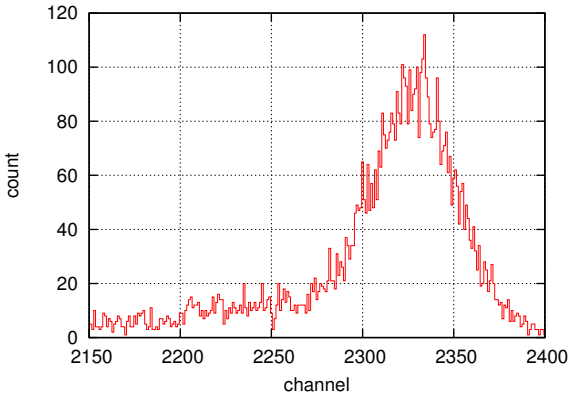


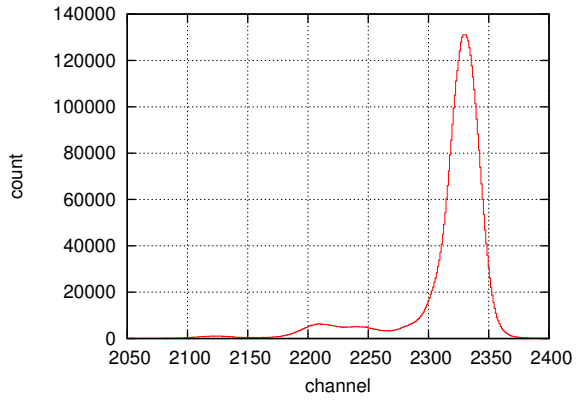
Abbildung 15.: Detektor 3.3 mit gepulstem Trigger.

Vorverstärker mit positivem Timing Signal wurden mit folgenden Detektoren verwendet: 1.4, 1.8-1.11, 2.3-2.5, 2.7-2.10, 3.3, 3.5, 3.8-3.10.

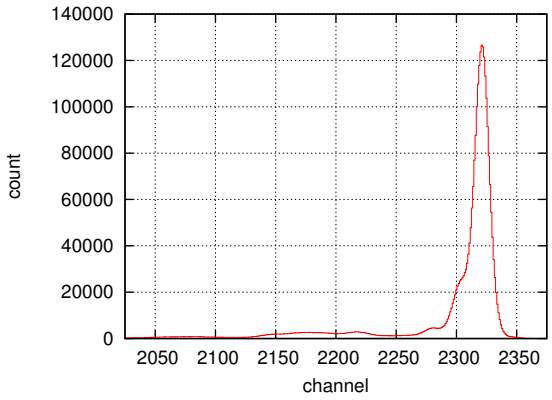
Der Versuch ein Spektrum mit einem im Pulser generierten Trigger auzunehmen zeigte aufgrund der geringen Wahrscheinlichkeit, dass ein Ereignis in das zufällige Gate fällt nur einen sehr schwachen α Peak. Es wies außerdem – wie auch die Spektren, die bei invertiertem Gate gemessen wurden – einen Peak bei niedrigen Energien auf, auf welchen in Abschnitt 4.2.3 genauer eingegangen wird. In Abbildung 15 kann man jedoch einen weiteren Peak neben dem Haupt Peak erahnen, was zeigt, dass die Überlegungen bezüglich des Triggers richtig sind.

4.2.2 Mehrere gleichzeitig betriebene Detektoren

Da bei Messungen mit dem Si-Ball nicht nur einzelne Detektoren betrieben werden, sondern mehrere nebeneinander liegende, wurde deren Einfluss aufeinander untersucht.



(a) Detektor 3.6



(b) Detektor 3.5

Abbildung 16.: Nebeneinander liegende über 12 h gemessene Detektoren.

Dabei fiel bei allen Detektoren ein doppelter Peak im Bereich des 5, 44 MeV Peaks auf, wie er in Abbildung 16(a) zu sehen ist. Für diese Messreihe wurde Detektor 3.5 zunächst einzeln gemessen, anschließend mit den direkt angrenzenden Detektoren, welche dann auch einzeln und zusammen aber ohne Detektor 3.5 gemessen wurden. Detektor 2.5, welcher auf einem höheren Arm als die direkt bestrahlten Detektoren lag wies noch weitere Verformungen und zusätzliche Peaks auf (Abbildung 16(b)).

Es ist jedoch anzumerken, dass die Peakdopplung mit einer Ausnahme (Detektor 1.10) immer nur bei der mittleren Energie auftrat, bei Detektor 3.4 allerdings auch ohne dass benachbarte Detektoren mit Spannung versorgt wurden. Dabei war die Dopplung jedoch nicht so deutlich zu sehen wie bei Messungen mit mehreren Detektoren.

Um der Ursache weiter auf den Grund zu gehen wurden Messungen durchgeführt, bei denen der Abstand der gleichzeitig betriebenen Detektoren variiert wurde. Der doppelte mittlere Peak tritt dabei auch auf, wenn zwischen den betriebenen Detektoren zwei weitere Detektoren liegen, die nicht mit Spannung versorgt sind. Ob die Detektoren auf dem selben Arm angebracht sind spielt auch keine Rolle, der als einziger Detektor auf Arm 2 betriebene Detektor 2.5 zeigte bei den Messun-

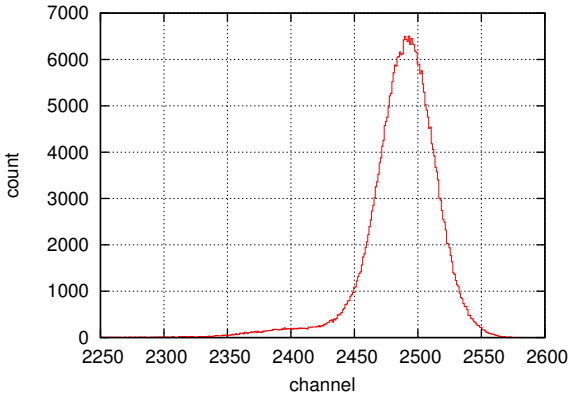
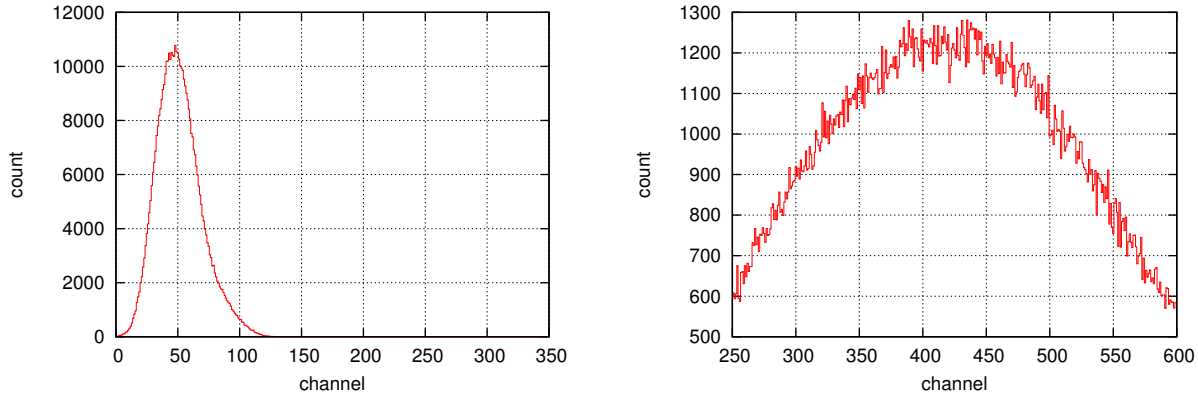


Abbildung 17.: Zusammen mit anderen Detektoren über 1, 25 h gemessener Detektor 3.3.

gen die stärksten Abweichungen eines normalen Spektrums.

Betrachtet man das Spektrum von Detektor 3.3 in Abbildung 17 fällt auf, dass hier ein zweiter Peak im Bereich der mittleren Energie zu erkennen ist wo vorher nur der Peak der höchsten Energie zu sehen war. Eine mögliche Erklärung der Peakdopplung ist also, dass dies ein zu niedrigeren Energien verschobener Anteil des hohen Energie Peaks ist. Die Verringerung der Energie könnte zum Beispiel als Resultat von den elektrischen Feldern in Halbleiter Dioden, die in Sperrrichtung betrieben werden auftreten. In dem Fall sollte dieser Effekt kein Problem für Messungen darstellen, so langenicht mehrere dicht beieinander liegende Energien gemessen werden sollen.

4.2.3 Niederenergie- und Threshold Messung



(a) Detektor 3.3 mit Spannung (b) Detektor 3.3 ohne Spannung

Abbildung 18.: Niedrige Kanäle aus Messungen mit Detektor 3.3.

Bei der Aufnahme von Eingangskanälen des SPA ohne dass an diese ein Kabel angeschlossen ist zeigt sich, dass jeder Kanal ein Rauschen hat, welches sich auf zwei bis drei benachbarte Kanäle im niedrigen Bereich erstreckt. Diese Signale werden vom ADC immer dann aufgenommen, wenn kein tatsächliches Signal am SPA anliegt, oder wenn zum Beispiel kein Kabel angeschlossen ist.

Selbst bei anschließen eines abgeschirmten Kabels mit zusätzlich abgeschirmten Stecker sind diese Signale vorhanden, ein Einfluss durch äußere elektrische Felder, die eine Spannung im Kabel induzieren ist also auszuschließen, diese Ereignisse sind Artefakte des SPA.

Abbildung 18 zeigt die zwei unterschiedlichen Signale, die bei niedrigen Kanälen zu sehen sind: 18(a) zeigt den Bereich bei einer Messung mit anliegender Spannung, Abbildung 18(b) zeigt ihn ohne anliegende Spannung. Letzterer Peak resultiert aus der Tatsache, dass die Silizium Halbleiter Dioden auch ohne anliegende Sperrspannung als Kondensator wirken können. Deponiert ein α Energie indem es Ladungsträger Paare erzeugt fließt kurz ein Strom durch den Widerstand des Siliziums, was als Spannung gemessen werden kann. Da die Ladungsträger jedoch nicht von einem hohen Potenzial abgesaugt werden ist das resultierende Spannungssignal geringer als bei anliegender Sperrspannung.

Der andere Peak wurde auch bei gepulsten Triggern gemessen, wenn die Quelle vom Detektor weg gedreht war. Er könnte aus γ Quanten aus dem ^{241}Am resultieren, welche nicht wie die α Teilchen vom Messingzylinder um die Quelle abgeschirmt werden.

4.2.4 Spektrum des Streifendetektors

Mit dem Streifendetektor lies sich kein vernünfitiges Spektrum aufnehmen, da sein Signal zu stark verrauscht ist. Betrachtet man das Spannungssignal eines Streifens auf einem Oszilloskop kann man einen Peak, der von einem α-Teilchen verursacht worden sein könnte nur erahnen. Die Amplitude hebt sich kaum aus dem Rauschen von etwa $\pm 100\,\mathrm{mV}$ hervor.

Auch das Timing Signal zeigt eine Form, die von dem erwarteten Signal stark abweicht plot. Versucht man dieses mit einem CFD in ein Trigger Signal umzuwandeln erhält man entweder permanent Trigger oder nur welche, die anscheinend nicht mit dem vermuteten Energiesignal korrelieren.

Es ist dabei anzumerken, dass durch den Streifendetektor nicht übermäßig viel Strom fließt. Bei der eingesetzten Spannungsquelle wird der Strom bis auf $0,1\,\mu\mathrm{A}$ genau angegeben unt überschreitet $0,2\,\mu\mathrm{A}$ nicht.

In einer vorhergehenden Bachelor Thesis [3] wurde zumindest der Hauptpeak des ^{241}Am bereits erfolgreich mit diesem Streifendetektor gemmesen. Da bei dieser Messung der Detektor auf eine andere Art und Weise mit Spannung versorgt wurde ist zu vermuten, dass dies damit zusammenhängt.

Desweiteren wurde bei der vorhergehenden Messung mit einer Blende gemessen um Störeffekte an Bondplättchen und Guardring zu vermeiden, ohne die Blende ergab sich jedoch lediglich ein zweiter Peak im Spektrum. Da nicht einmal diese Form des Spektrums aufgenommen werden kann lässt sich die nicht vorhandene Blende als Ursache für die Probleme mit dem Streifendetektor ausschließen.

Beim Öffnen der Streukammer nach den Messungen ist bei betrachten des Steckers am Detektor aufgefallen, dass die Spannung möglicherweise falsch gepolt ist, was die auftretenden Effekte erklären könnte.

5 Fazit

5.1 Detektoren

Es konnte gezeigt werden, dass die Detektoren prinzpiell Signale aufnehmen können. Abweichungen von optimalen Signalformen sin vermutlich nicht auf Detektor Defekte zurückzuführen.

Vorhergehende Messungen in einer anderen Bachelor Thesis [7] haben gezeigt, dass sich bei einigen Detektoren keine Sättigung im Sperrstrom einstellt. Dieser ging bereits nach kurzer Zeit auf bis zu $3\,\mu\mathrm{A}$, was sich nicht erklären lies. Wiederholtes überwachen des Stroms zeigte, dass dieser Effekt bei zumindest einem Detektor (Si B 30; Platz fünf auf Detektorarm drei) noch deutlicher als bei anderen auftritt, nachdem über mehrere Tage 150 V Sperrspannung angelegt waren hat ist der Strom jedoch nicht über 650 nA gestiegen. Was diesen Effekt verursacht und wieso der Strom jetzt geringer ist lässt sich immer noch nicht erklären, entsprechende Detektoren liefern jedoch weiterhin weitestgehend vernünftige Spektren.

Die Streifendetektoren konnten nicht vermessen werden, ein Umpolen der Sperrspannung könnte hier für eine Verbesserung sorgen.

5.2 Vorverstärker

Die verwendeten Vorverstärker wurden im Rahmen einer Bachelor Thesis [7] auf die selbe Verstärkung kalibriert. Die gemessenen Spektren zeigen jedoch, dass sich diese Einstellung über zwei Jahre ver-

schieben kann, so dass für eine vernünftige Energiemessung entweder für jedes Spektrum eine eigene Energiekalibrierung vorgenommen werden muss, oder die Vorverstärker erneut auf die selbe Verstärkung eingestellt werden müssen.

Abgesehen von der unterschiedlichen Polarität der Timing Signale haben sie jedoch funktioniert. Bei betrachten der Signale mit einem Oszilloskop konnten gelegentliche Signale mit wesentlich größerer Anstiegszeit beobachtet werden, die ein Timing Signal mit geringerer Amplitude als die anderen Signale hatten. Ob diese einen Einfluss auf die Messbarkeit anderer Teilchen als α Teilchen haben gilt es noch zu prüfen.

5.3 Ausleseelektronik

Für einen voll besetzten Si-Ball werden einige weitere Module zur Verarbeitung der Signale benötigt werden. Das beinhaltet CFDs für die richtige Polarität der Timing Signale, Delays und Koinzidenzmodule. Die Polarität der Timing Signale muss entweder in den Vorverstärkern angepasst werden oder genügend Kanäle bei CFDs zur Verfügung stehen, welche die entsprechende Polarität verarbeiten können.

Insbesondere ist bei einsetzen von weiteren Modulen darauf zu achten, dass die Kabel mit den richtigen Widerständen abgeschlossen sind. Im Laufe der Messungen zu dieser Bachelor Thesis war dies bei einem defekten Modul nicht mehr der Fall wodurch Reflexionen aufgetreten sind, die eine sinnvolle Verarbeitung der Signale unmöglich machten. Desweiteren sollte man immer das richtige Timing und die richtige Polarität und Signalform im Auge behalten, da es sonst schnell zu schwer erklärbaren Effekten im Spektrum kommen kann.

5.4 Datenaufnahme

Die im Rahmen dieser Bachelor Thesis entwickelte Konfiguration von MBS eignet sich zur Aufnahme von nicht-koinzidenten, Single-Energiespektren. Um auch koinzidente Ereignisse sinnvoll aufnehmen zu können sind ein paar Anpassungen vorzunehmen.

Um bei der Analyse der Listmode Datei einfach zwischen Spektrometer- und Si-Ball-Ereignissen zu unterscheiden empfiehlt es sich diese in ein Event mit eigenem Header zusammenzufassen. Der Header sollte zumindest Informationen über Anzahl und Reihenfolge der verschiedenen Ereignisse haben, zum Beispiel *ein Spektrometer-Ereignis, zwei Si-Ball-Ereignisse, zwei Zeit-Informationen* oder *ein Spektrometer-Ereignis, kein Si-Ball-Ereignis, keine Zeit-Information*.

Dies sind jedoch nur kleine Änderungen an den bereits funktionierenden Dateien.

5.5 Einsatzfähigkeit des Si-Balls

Die Messungen haben gezeigt, dass prinzipiell mit dem Aufbau des Si-Ball Prototypen α Teilchen gemessen werden können. Für Experimente mit α Teilchen im Bereich von 5 MeV oder mehr ist der Si-Ball als Detektor verwendbar.

Da im niederenergetischen Bereich die Elektronen aus dem Strahl einen Peak verursachen werden, der ganz dem hier gemessenen Peak ähneln wird reicht die Energieauflösung vermutlich nicht aus um ihn von anderen Teilchen dieses Energiebereichs zu trennen.

5.6 Schritte zu einer Koinzidenzmessung

Es ist das Ziel den Si-Ball in einer Koinzidenzmessung mit dem QCLAM Spektrometer zu betreiben. In der dazu nötigen Schaltung soll berücksichtigt werden, dass das Spektrometer Single-Spektren aufnehmen können soll, ohne auf den Si-Ball angewiesen zu sein. Im Schaltbild in Abbildung 3 wird daher das Trigger Signal zur Auslese aus dem Spektrometer gewonnen. Die Timing Signale aus dem Si-Ball werden lediglich zur Zeitmessung und für das Gate, welches ADC und TDC aktiviert verwendet. Letzteres ist jedoch nur dann der Fall, wenn auch ein Ereignis im Spektrometer gemessen wurde, um zufällig auftretende Ereignisse in der Aufnahme auszuschließen. Die Timing Signale aus dem Si-Ball müssen daher so weit verzögert werden, dass sie in jedem Fall hinter dem Trigger des Spektrometers liegen, jedoch noch zeitnah zu diesem, dass eine Gleichzeitigkeit ermittelt und dadurch das Gate für ADC und TDC geöffnet werden kann (der TDC wird im Common Start Modus betrieben). Entsprechend müssen auch die Energie Signale verzögert werden, dass sie im Gate liegen.

Wird in den Driftkammern kein Ereignis registriert könnte der Trigger aus dem Spektrometer auch durch ein zufälliges Ereignis ausgelöst worden sein und das gesamte Event muss verworfen werden. Dazu besteht die Möglichkeit in diesem Fall einen Trigger an die Triva zu geben, bei welchem keine Daten aus ADC, TDC und Spektrometer ausgelesen werden, sondern diese gelöscht werden. Eine andere Möglichkeit wäre es dies in die Busy-Logik einzubringen, was sich aber vermutlich als komplizierter erweisen wird.

Um eine weitere Datenaufnahme zu unterbinden während eines der Module beschäftigt ist wird das Gate zur Datenaufnahme nur dann geöffnet, wenn alle Module bereit sind. Zu beachten ist hierbei der Fall, dass der verwendete ADC ein Busy Signal ausgibt wenn sein Speicher komplett voll ist.

5.7 Weitere Optimierungen

Um den RIO und insbesondere desen Speicherkapazität zu entlasten ist es sinnvoll die Daten über das Netzwerk weg zu schreiben. Das Mittel der Wahl ist hier ein RFIO-Server, für den es bereits einen in Linux ausführbaren Dienst gibt, der mit MBS kompatibel ist. Dieser müsste auf einem entsprechenden Computer mit verfügbarem Festplattenspeicher gestartet werden und der `open file` Befehl im Startup Skript entsprechend angepasst werden.

Eine weitere Verbesserung zur Datenaufnahme wär die Möglichkeit der Live Analyse, welche bei laufendem RFIO-Server auch relativ einfach erreicht werden kann, indem man in dem entwickelten Konvertierprogramm der Event API statt dem Dateinamen die IP des RFIO-Servers übergibt.

Um die Energieauflösung zu verbessern könnten die Detektoren gekühlt werden, was das Rauschen verringern würde. Dazu müsste optimalerweise die Wärme komplett aus der Streukammer transportiert werden.

Um den Zug an den Vorverstärkern etwas zu entlasten könnte man über eine andere Methode der Zugentlastung nachdenken. Am meisten würde dies durch das ersetzen einiger Kabel erreicht werden, die etwas länger sind als die bereits vorhandenen. Insbesondere die Signal Kabel, die zu den Durchführungen zum Messraum führen sind teilweise gerade so lang genug, dass sie überhaupt verwendet werden konnten.

Abbildungsverzeichnis

Literaturverzeichnis

[1] P. A. Tipler and G. Mosca, *Physik für Wissenschaftler und Ingenieure*, Spektrum Akademischer Verlag, 2006.

[2] W. R. Leo, *Techniques for Nuclear and Particle Physics Experiments*, chapter 10, 12, Springer-Verlag, 1987.

[3] R. Kissel, Charakterisierung von Siliziumstreifendetektoren, Bachelor Thesis, 2012.

[4] Persönliche Gespräche mit Jörg.

[5] H. Spieler, *Semiconductor Detector Systems*, Oxford University Press, 2005.

[6] D. G. Lutz, *Semiconductor Radiation Detectors*, Springer, 2007.

[7] A.-L. Hartig, Charakterisierung von Siliziumdetektoren für die Untersuchung von Aufbruchsreaktionen am S-DALINAC, Bachelor Thesis, 2010.

[8] http://www.nndc.bnl.gov/nudat2/decaysearchdirect.jsp?nuc=241AM&unc=nds, 2012.

[9] M. Patrizio and S. Schlemme, Entwicklung einer Kalibriereinheit für das Goniometer, Miniforschung, 2011.

[10] Persönliche Gespräche mit Michael Reese.

A Anhang

A.1 Zuordnung Vorverstärker

A.2 Zuordnung Detektoren

A.3 Erlaubte Signalform an den Modulen

Die Eingangssignale am SPA dürfen eine Amplitude von 8 V nicht überschreiten. Das Ausgangssignal hat ebenfalls bis zu 8 V, die Verstärkung muss also an die Bedürfnisse des ADC angepasst werden.

Das Signal an den Eingängen des Caen ADC darf eine Amplitude von 4 V haben. Hat das Signal ein Überschwingen in eine negative Polarität darf die Spannungsdifferenz 4 V nicht überschreiten. Dies kann zum Beispiel der Fall sein, wenn der Caen SPA ein Signal umwandelt, dessen Abfallzeit zu gering ist. Das Gate muss negative Polarität haben. Falls es nicht an ein anderes Modul weitergeleitet wird – zum Beispiel einen TDC – muss der Common Ausgang mit 50 Ω abgeschlossen werden.

Die Triva erwartet an den Trigger Eingängen logische Signale im ECL-Standard (*emitter coupled logic*). Da die verwendeten NIM Module logische Signale im NIM-Standard (*nuclear instrumentation module*) ausgeben müssen dies zunächst konvertiert werden. In beiden Standards haben die Signale eine Dauer von wenigen 10 ns, die Amplituden sind in Tabelle 6 dargestellt. Die logischen Signale werden durch CFDs oder logische Module erzeugt.

A.4 Anmelde- und Kopiervorgang am RIO

MBS läuft in einer virtuellen Maschine auf *lynxboot* (IP: 192.168.2.2). Um sich über ein Terminal auf lynxboot anzumelden verwendet man den Befehl

```
ssh username@192.168.2.2
```

Von dort kann man sich per Telnet mit dem RIO verbinden.

```
telnet r3-1
```

Es wurde hier mit dem Benutzer `user3_1` auf dem RIO `r3-1` gearbeitet.

Die Daten für MBS befinden können auf dem RIO über die Verknüpfung

```
/home/user3_1/si-ball
```

erreicht werden. Dieser Ordner befindet sich auf lynxboot in

```
/LynxOS/mbsusr/user3_1/mbsrun/rio3/rio3_vme/si_ball/
```

Von einem beliebigen Rechner, der sich im selben Netzwerk wie lynxboot befindet können Daten über

Platz	Detektor	VV	I / μA	VV Anschlüsse		
				Spannung	HV Kabel	HV Ch
			Detektorarm 1			
1						
2	46	29		13	D/II8	0.00.08
3	42	14		14	D/II9	0.00.09
4	38	311		15	D/II10	0.00.10
5	35	6		16	D/II11	0.00.11
6	32	1		1		
7	2	355		2		
8	1	302		3		
9	3	359		4		
10	5	350		5		
11	7	353		6		
12						
			Detektorarm 2			
1						
2	8	27		19	B/I4	0.02.10
3	9	352		20	B/I5	0.02.11
4	14	362		21	B/I6	0.02.00
5	15	357		22	B/I7	0.02.01
6	24	9		23	B/I8	0.02.02
7	25	306		24	B/I9	0.02.03
8	26	351		9	B/I10	0.02.04
9	27	297	0,1	10	B/I11	0.02.05
10	28	301		11	D/II6	0.00.06
11	39	15	0,2	12	D/II7	0.00.07
12						
			Detektorarm 3			
1						
2	34	299		25	C/II0	0.00.00
3	33	303	0,4	26	C/II1	0.00.01
4	31	23	1,1	27	C/II2	0.00.02
5	30	13	2,9	28	C/II3	0.00.03
6	23	7	2,9	29	C/II4	0.00.04
7	21	16	0,8	30	C/II5	0.00.05
8	20	198	0,7	31	B/I0	0.02.06
9	13	295	0,8	32	B/I1	0.02.07
10	10	360	0,7	17	B/I2	0.02.08
11	4	349	1,3	18	B/I3	0.02.09
12						

Tabelle 4.: Zuordnung der Detektoren zu Vorverstärkern und deren Anschlüsse. Die HV Kabel der Detektoren 6-11 am Detektorarm 3 wurden nach abgeschlossenen Messungen an Detektoren 6-11 an Detektorarm 1 gesteckt.

Platz	Detektor	VV Anschlüsse			
		Kabelbezeichnung		Durchführung	
		Energy	Timing	Energy	Timing
		Detektorarm 1			
1					
2	46	1.2	1.3	20g	21g
3	42	6.1	6.2	34g	35g
4	38	6.3	18.1	36g	1b
5	35	18.2	18.3	2b	3b
6	32	16.1	16.2	4b	5b
7	2	16.3	17.1	6b	7b
8	1	17.2	17.3	8b	9b
9	3	Si.1	Si.2	11b	12b
10	5	Si.3	Si.4	13b	14b
11	7	Si.5	Si.6	15b	26b
12					
		Detektorarm 2			
1					
2	8	11.3	14.1	51g	59g
3	9	14.2	14.3	60g	6g
4	14	3.1	3.2	25g	26g
5	15	3.3	7.1	27g	37g
6	24	7.2	7.3	38g	39g
7	25	4.1	4.2	28g	29g
8	26	4.3	8.1	30g	40g
9	27	8.2	8.3	41g	42g
10	28	5.1	5.2	31g	32g
11	39	5.3	1.1	33g	19g
12					
		Detektorarm 3			
1					
2	34	2.1	2.2	22g	23g
3	33	2.3	15.1	24g	16b
4	31	10.1	10.2	46g	47g
5	30	15.2	15.3	17b	18b
6	23	10.3	13.1	48g	55g
7	21	13.2	13.3	56g	57g
8	20	12.1	12.2	52g	53g
9	13	12.3	9.1	54g	43g
10	10	9.2	9.3	44g	45g
11	4	11.1	11.2	49g	50g
12					

Tabelle 5.: Zuordnung der Detektoren zu ihren Signal-Anschlüssen. In den Spalten unter *Durchführung* steht *g* für gelbes und *b* für braunes Shielding.

	0	1	Impedanz
NIM	0 V	$-0,8$ V	50 Ω
ECL	$-0,9$ V	$-1,75$ V	$(100-110)\,\Omega$

Tabelle 6.: Logische Signale im NIM- und ECL-Standard. Die Spannung des *fast negative* NIM-Standards ist eigentlich über den Strom bei 50 Ω Impedanz definiert. [2, S. 252f]

```
scp -r /lokaler/mbs/ordner/*
    user3_1@192.168.2.2:/LynxOS/mbsusr/user3_1/mbsrun/rio3/rio3_vme/si_ball/
scp user3_1@192.168.2.2:/LynxOS/mbsusr/user3_1/mbsrun/rio3/rio3_vme/si_ball/data.lmd
    ./
```

kopiert werden.

A.5 MBS Skripte

In MBS können Skripte verwendet werden, welche mit der Endung `.scom` gespeichert werden und über `@skriptname` aufgerufen werden. Es werden dann alle Befehle nacheinander ausgeführt, welche im Skript stehen. Startup und Shutdown Skript wurden im Wesentlichen aus bereits funktionierenden Experimenten übernommen.

A.5.1 startup.scom

Dast Startup Skript enthält folgende Zeilen:

```
start task m_util
load setup setup.usf
set trig_mod
start task m_read_meb "./m_read_meb"
start task m_collector
start task m_transport
start task m_daq_rate
open file "./data.lmd" -disk
```

Zu aller erst wird der Task `m_util` gestartet. Er würd für diverse Befehle benötigt, insbesondere den Start der Datenaufnahme. Anschließend wird die Setup Datei für das Trigger Modul geladen und mit dem nächsten Befehl mit diesen Einstellungen das Trigger Modul initialisiert. Das aus der `f_user.c` Datei erzeugte Programm wird ausgeführt und darin die Funktion `\_user_get_virt_ptr` aufgerufen. Der Task `m_collector` ermöglicht die online Visualisierung innerhalb von MBS, welche hier jedoch nicht verwendet wurde. Der Task zum speichern von Daten wird gestartet. Dieser ermöglicht es Dateien auf ein Tape oder eine Festplatte zu schreiben, oder sie an einen Server im Netzwerk zu schicken. Der Task `m_daq_rate` taucht im Reference Manual nur in einem Platzhalter Befehl auf (wurde hier nicht entfernt, da das Skript funktionierte). Es wird eine lokale Listmode Datei `data.lmd` erzeugt, in die alle Daten geschrieben werden.

Sollen die Daten über das Netzwerk an einen RFIO-Server geschickt werden, muss zum einen auf dem entsprechenden Server rfio-programm laufen, zum anderen muss die letzte Zeile im Skript mit

```
#### open rfio ####
```

ersetzt werden.

Auf den Start der Datenaufnahme mit dem Befehl `sta acq` wurde im Skript bewusst verzichtet, da es Szenarien geben kann, in denen diese erst verzögert gestartet werden soll.

Für ein größeres Experiment ist es eventuell von Interesse, das Öffnen der Listmode Datei aus dem Skript zu nehmen um bei mehreren aufeinanderfolgenden Messungen die Dateinamen anpassen zu können. Zu beachten ist dabei jedoch, dass ohne eine Datei zu öffnen das Experiment zwar scheinbar normal durchgeführt wird, die Daten jedoch nicht gespeichert werden können.

A.5.2 shutdown.scom

Im Shutdown Skript stehen diese Befehle:

```
stop acq
close file
stop task m_transport
stop task m_collector
stop task m_read_meb
stop task m_util
```

Es stoppt die Datenaufnahme, falls dies nicht schon geschehen ist und schließt die Listmode Datei. Wird dieser Befehl nicht ausgeführt kann dies zur Korruption der Datei führen, wodurch sie im Zweifelsfall nicht mehr korrekt gelesen werden kann, beziehungsweise es sind noch nicht alle Daten in die Listmode Datei geschrieben. Die folgenden Befehle stoppen alle Tasks, die im Startup Skript gestartet wurden.

A.6 f_user.c

Die `f_user.c` Datei besteht im Wesentlichen aus drei Funktionen:

- `f_user_get_virt_ptr` – initialisiert Pointer zu Adressen der Module
- `f_user_init` – initialisiert Module
- `f_user_readout` – verarbeitet Trigger

Der Übersicht halber wurden die Funktionen für ADC und TDC in extra C Dateien ausgelagert, die je nach Bedarf als Modul integriert werden können.

Bei der Initialisierung der Pointer wird wie bei den anderen Funktionen auch die Adresse des VME Crates im Pointer `pl_loc_hwacc` übergeben. Dieser wird als Offset verwendet um die Adresse des ADC in den Pointer `adc1` zu schreiben. Hierfür wird der entsprechenden Funktion aus dem ADC Modul der Wert der Rädchen auf der Rückseite des ADC übergeben. Wird dieser verändert oder ein anderer ADC verwendet muss das hier angepasst werden. Über diese Adresse können die Einstellungen des ADC verändert und auf seinen Speicher zugegriffen werden.

Bei der Initialisierung der Module wird genau wie bei der Auslesefunktion auch eine Fallunterscheidung für `bh_crate_nr` gemacht, welche nur bei der Verwendung mehrerer RIOs in unterschiedlichen Crates relevant ist. Über die entsprechenden Funktionen aus dem ADC Modul wird beim ADC ein Software Reset durchgeführt, welcher fast alle Einstellungen auf einen Standard zurück setzt und den Multi Event Buffer (MEB) löscht. Die Einstellungen werden auf ein selbst definiertes Standard Setup gesetzt. Der

Schwellwert aller Kanäle wird auf 108 gesetzt. Er berechnet sich aus dem Produkt von dem der Funktion übergebenen Wert von 54 und 2, falls Step Threshold aktiviert ist (dies ist der Standard Wert), beziehungsweise 16, falls Step Threshold deaktiviert ist. Anschließend werden Zero Suppression aktiviert (Werte unter dem Schwellwert werden nicht gespeichert) und Overflow Suppression deaktiviert (es wird nicht gewarnt, falls der Wert eines Ereignisses möglicherweise zu groß ist, dass der ADC ihn nicht korrekt konvertieren konnte). Eine ähnliche Konfiguration für einen möglichen TDC ist auskommentiert.

In der Auslesefunktion wird neben der Fallunterscheidung des Crates noch eine für den Trigger Typ gemacht, welcher mit der Variable `bh_trig_typ` übergeben wird. Bei Start (14), Stop (15) und erhaltenem Trigger auf Trigger Eingang eins (1) wird die Auslesefunktion aus dem ADC Modul aufgerufen. Sie schreibt alle zu speichernden Daten an die Adresse des Pointers `pl_dat` und gibt die Anzahl an geschriebenen 32-Bit Wörtern zurück, welche für MBS an die Adresse des Pointers `l_se_read_len` geschrieben werden muss. Für Debug Zwecke wird nun noch eine Variable hoch gezählt, die alle 1000 erhaltenen Trigger ausgegeben wird, dass man sehen kann ob die Datenaufnahme noch funktioniert.

```
/* N.Kurz, GSI, 12-Feb-1999 */
/* H. Lauinger, Sep-2012 */

#include <smem.h>

#include "stdio.h"
#include "s_veshe.h"

#include "modules/CaenV785.h"
#include "modules/CaenV775.h"

/***********************************************************************/

/*
 * all pointer which are used for read/write operations in the functions
 * f_user_init and f_user_readout must be defined here as static variables
 */

static CaenV785_module adc1;
//static CaenV775_module tdc1;
int trigger_count = 0;

/***********************************************************************/

int f_user_get_virt_ptr (long  *pl_loc_hwacc, long  pl_rem_cam[])
{
  /*
   * create virtual pointer to be used in f_user_init and f_user_readout
   */
  adc1 = CaenV785_module_init(0x0000, pl_loc_hwacc);
//  tdc1 = CaenV775_module_init(0x0001, pl_loc_hwacc);
}

/***********************************************************************/

int f_user_init (unsigned char   bh_crate_nr,
```

```
                  long           *pl_loc_hwacc,
                  long           *pl_rem_cam,
                  long           *pl_stat)

{
  int i = 0;
  switch (bh_crate_nr)
  {
    case 0:
      printf("adc firmware = %d\n", adc1.reg->firmware_revision);
//      printf("tdc firmware = %d\n", tdc1.reg->firmware_revision);

      printf("init adc1\n");
      CaenV785_SoftReset(&adc1);
      CaenV785_StandardSetup(&adc1);
//      CaenV785_DisplayStoredThresholds(&adc1);
      //save space
      for (i = 0; i < 32; i++) {
          // set Threshold to 54*2 = 100 (Step Threshold defaults 1)
         CaenV785_SetThreshold(&adc1, i, 54);
      }
      CaenV785_ZeroSuppressionOn(&adc1);
      CaenV785_OverflowSuppressionOff(&adc1);
/*      printf("init tdc1\n");
      CaenV775_SoftReset(&tdc1);
      CaenV775_StandardSetup(&tdc1);
      CaenV775_StepThresholdOff(&tdc1);
      CaenV775_ZeroSuppressionOn(&tdc1);
      CaenV775_OverflowSuppressionOn(&tdc1);
      CaenV775_EnableAllChannels(&tdc1);
      CaenV775_ResetThresholds(&tdc1);
      CaenV775_DisplayStoredThresholds(&tdc1);*/
      break;
    default:
      break;
  }
  return 1;
}

/*****************************************************************************/

int f_user_readout (unsigned char   bh_trig_typ,
                    unsigned char   bh_crate_nr,
                    register long  *pl_loc_hwacc,
                    register long  *pl_rem_cam,
                    long           *pl_dat,
                    s_veshe        *ps_veshe,
                    long           *l_se_read_len,
                    long           *l_read_stat)
{
```

```
  *l_se_read_len = 0;
  switch (bh_crate_nr)
  {
    case 0:

      switch (bh_trig_typ)
      {
        case 14:
        case 15:
        case 1:
          *l_se_read_len += CaenV785_Readout(&adc1, pl_dat);
//          *l_se_read_len += CaenV775_Readout(&tdc1, pl_dat);
          trigger_count++;
          if (trigger_count % 1000 == 0 || bh_trig_typ > 7) {
            printf("received %i trigger\n", trigger_count);
          }
          break;
        case 2:
        case 12:
        case 13:
          break;
        default:
          break;
      }
      break;
    default:
      break;
  }

  return 1;
}

/*****************************************************************************/
```

A.7 CaenV785.c

Auf den ADC wird über eine Struktur `CaenV785_module` zugegriffen. Sie enthält einen Pointer zur Adresse des MEB und einen zu den Registern, über die Einstellungen geändert und Status abgefragt werden können. Die Definition dieser Struktur inklusive der Bezeichnung der einzelnen Register ist in der Header Datei `Caen_v785.h` zu finden. Die Adressen der Register stehen im Handbuch zum ADC, die Größe jedes Registers beträgt 16 Bit. Die Struktur ermöglicht den Zugriff auf jeden Register, ohne die individuellen Adressen anzugeben, indem die Pointer auf 16 Bit Wörter in der richtigen Reihenfolge definiert werden. Bei Initialisierung des Moduls wird die Adresse des ersten Registers übergeben und als Adresse einer `CaenV785_reg` Struktur gespeichert. Diese Struktur enthält nun mehrere Pointer auf 16 Bit Wörter, auf die über ihre Bezeichner zugegriffen werden kann. Der erste Pointer (`firmware_revision`) zeigt auf die ersten 16 Bit, der zweite (`geo_addr`) auf die zweiten 16 Bit, der dritte (`mcst_addr`) auf die nächsten 15 Bit und so weiter. Vergleicht man dies mit der Adress Tabelle im Handbuch zum ADC sieht man, das dies mit den Adressen der einzelnen Register übereinstimmt, wenn man nur die Adresse des ersten Registers richtig übergeben hat. Da zwischen einigen Registern mehr als 16 Bit Platz sind, welche jedoch nicht

verwendet werden, enthält die CaenV785_reg Struktur hier einige Dummy-Pointer, die auf diese nicht verwendeten Register zeigen.

Als Beispiel für einen Zugriff soll überprüft werden ob auf dem ADC Daten bereit liegen. Diese Information steht im nullten Bit des ersten Status Registers. Zeigt ein Pointer adc1 einer CaenV785_module Struktur nun also auf die Adresse des ADC kann auf diesen Wert über

```
adc1->reg->status_reg_1 & 0x1
```

zugegriffen werden. Zum lesen des Busy Status (zweites Bit desselben Registers) verwendet man also

```
(adc1->reg->status_reg_1 >> 2) & 0x1
```

Zum schreiben von Einstellungen gibt es Bit Set Register. Um eine Einstellung zu aktivieren muss eine 1 an die entsprechende Stelle des Registers geschrieben werden. Um die Einstellung zu deaktivieren muss eine 1 an dieselbe Stelle des entsprechenden Bit Clear Registers geschrieben werden. Um Step Threshold zu aktivieren bzw. deaktivieren verwendet man also

```
adc1->reg->bit_set_2 = 0x0100;
adc1->reg->bit_clear_2 = 0x0100;
```

oder

```
adc1->reg->bit_set_2 = 1 << 8;
adc1->reg->bit_clear_2 = 1 << 8;
```

Für die wichtigen Einstellungen enthält das Modul Funktionen, welche im Quelltext erklärt sind.

Drei andere wichtige Funktionen sind CaenV785_module_init, welche beim initialisieren aufgerufen wird, CaenV785_Readout, welche bei jedem Trigger aufgerufen wird und die Setup Funktion CaenV785_StandardSetup.

Beim initialisieren wird eine CaenV785_module Struktur erzeugt. Die Adresse des ADC setzt sich aus der Adresse des Crates (pl_loc_hwacc), der Einstellung am ADC, die aus f_user.c übergeben wird (address) und einem Offset (0x0000) zusammen. Ist am ADC also beispielsweise 00 01 eingestellt muss zu der Adresse des Crates 0x00010000 addiert werden.

Beim Aufruf der Auslesefunktion wird zunächst überprüft ob der Trigger bei leerem MEB kam. Ist dies 500 mal der Fall wird zu Debug Zwecken eine Meldung asugegeben. Ist dies nicht der Fall wird das Busy Bit und der Zustand des MEB abgefragt. Ist das Busy Bit 1 und der MEB nicht voll (in dem Fall ist das Busy Bit immer 1) ist der ADC noch mit der Konvertierung der Daten beschäftigt. Der Speicher wird nicht ausgelesen und auch hier wird beim 500sten Mal zu Debug Zwecken eine Meldung ausgegeben.

Ist der ADC auch nicht mit dem Umwandeln von Daten beschäftigt wird der Speicher ausgelesen so lange Daten bereit sind. Um nicht in eine Endlosschleife zu geraten wird hier als Obergrenze die maximale Speichergröße von 34 Datenwörtern als Abbruchkriterium verwendet. Beim lesen des MEB verschiebt der ADC den Pointer automatisch auf das nächste Datenwort. Der Wert des ersten Worts im MEB wird daher in eine temporäre Variable gespeichert, um ihn für verschiedene Abfragen mehrfach lesen zu können. Enthält das Datenwort einen Header wird die Anzahl an folgenden Wörtern mit Daten gelesen, der Header in den Pointer für MBS geschrieben, die Wörter mit Daten in einer Schleife ausgelesen und ebenfalls in diesen Pointer geschrieben und anschließend ein weiteres Wort, welches den End Of Block (EOB) enthält gelesen und gespeichert. Da auf jedes Wort nur einmal zugegriffen werden muss können diese direkt in den Pointer von MBS geschrieben werden, ohne sie zunächst in eine temporäre Variable zu speichern. Da der Pointer auf den MEB automatisch vom ADC auf das nächste Wort geschoben wird

muss dieser nicht manuell erhöht werden. Zu Debug Zwecken wird auch hier bei jedem 10000sten Wort mit Daten eine Meldung ausgegeben.

Enthielt das erste Datenwort keinen Header wird es dennoch gespeichert um Datenverlust vorzubeugen. Ist man sich sicher, dass keine gültigen Daten vorliegen, falls das erste Datenwort kein Header ist kann diese Zeile auskommentiert werden um Speicherplatz zu sparen und das Umwandeln der Listmode Datei in Histogramme zu beschleunigen.

Die Auslese Funktion beinhaltet weitere auskommentierte Zeilen, die kodierte Informationen über den Status des ADC in die Listmode Datei schreiben, falls das Auslesen getriggert wurde, während der ADC noch nicht ausgelesen werden konnte.

Liegt bei jedem Trigger genau ein Event im MEB ist die Schleife zum auslesen des gesamten MEB obsolet. Sie wurde eingefügt, da der Speicher aufgrund der falschen Polarität des Gates sehr schnell voll gelaufen ist, die Ursache zu diesem Zeitpunkt jedoch noch nicht bekannt war.

In der Standard Setup Funktion wird um Fehler auszuschließen der MEB des ADC gelöscht und der Speichertest deaktiviert. Aktivieren der Sliding Scale verringert die Anzahl an Kanälen zwar auf etwa 3800, es vermidert jedoch kanalspezifische Ungenauigkeiten bei der Konvertierung. Jedes Signal wird um einen zufälligen, bekannten Offset verschoben, dadurch bei gleichen Amplituden immer in unterschiedlichen Kanalbereichen konvertiert und anschließend das Offset wieder abgezogen. Desweiteren werden im Setup Step Threshold und Overflow Suppression aktiviert, Zero Suppression deaktiviert, alle Eingänge aktiviert und alle Schwellwerte auf null zurückgesetzt.

```
#include "CaenV785.h"

#include <stdio.h>
#include <string.h>

//#include "adc_785_debug_tools.h"

int CaenV785_event_count = 0, CaenV785_busy_count = 0,
   CaenV785_empty_count = 0, CaenV785_invalid_count = 0;


// calculate module base address and return a module-structure
CaenV785_module CaenV785_module_init(uint16_t address, long *pl_loc_hwacc)
{
  long module_base_addr = (long)pl_loc_hwacc + address*CAEN_V785_OFFSET;

  struct CaenV785_module module =
  {
    (uint32_t *) ( module_base_addr ), // output_buffer
    (struct CaenV785_reg*) ( module_base_addr + CAEN_V785_REG_OFFSET )  //
        reg
  };

  return module;
}


// Read the Output Buffer of CAEN ADC V785
int CaenV785_Readout(CaenV785_module *module, long *mbs_data)
```

```
{
  long i, temp, loop_count = 0;
  long num_data_words, lw_type;
  long *mbs_data_start = mbs_data;


  //if(!(module->reg->status_reg_1 & 0x04)) // module not busy
  if (CaenV785_CheckBufferStatus(module) != 0) // meb not empty
  {

    if (CaenV785_CheckBusyStatus(module) == 1 &&
       CaenV785_CheckBufferStatus(module) != 2) // module is busy
    {
//      *mbs_data++ = 0x05000000 + (module->reg->status_reg_1 & 0x01FF);
   // module busy
      CaenV785_busy_count++;
      if (CaenV785_busy_count % 500 == 0) {
  printf("counted %i trigger while busy (and meb not full)\n",
     CaenV785_busy_count);
      }
    }
    else
    {
      while (CaenV785_CheckDreadyStatus(module) == 1 && loop_count < 34)
      {
  loop_count++;

  // read the first longword
  temp = *module->output_buffer; // for some reason, the output_buffer
     pointer must not be incremented !!

  // extract longword "type" (2=header,6=invalid data)
  lw_type = (temp >> 24) & 0x7;

  if (lw_type == 2)
  {
    // extract number of data longwords
    num_data_words = (temp >> 8) & 0x3F;
    *mbs_data++ = temp;
//    printf("wrote header of %i data longwords\n", num_data_words);
    // read and pass on the data
    for(i=0 ; i < num_data_words; i++)
    {
      *mbs_data++ = *module->output_buffer;
      /*    printf("\tch: %d un: %d ov: %d -> val: %d\n",
       *      (*(mbs_data-1) >> 16) & 0x1F,
       *      (*(mbs_data-1) >> 13) & 0x1 ,
       *      (*(mbs_data-1) >> 12) & 0x1 ,
       *      (mbs_data-1) & 0xFFF
       *      ); */
```

```
      CaenV785_event_count++;
      if (CaenV785_event_count % 10000 == 0) {
        printf("counted %i valid subevents above threshold\n",
           CaenV785_event_count);
      }
    }
    // read the EOB longword (GEO and "event counter" info)
    *mbs_data++ = *module->output_buffer;
    //  printf("wrote EOB\n");
  }
  else  // most likely no valid data (lwtype was 6)
        {
//    printf("header expected: %X\n", temp);
    CaenV785_invalid_count++;
    if (CaenV785_invalid_count % 1000 == 0) {
      printf("counted %i events without header\n", CaenV785_invalid_count)
         ;
    }
  //save space
    *mbs_data++ = temp;
        }
      }  // while DReady
    }
    // clear memory to prevent memory overflow
    //    CaenV785_ClearData(module);
  } // meb not empty
  else
  {
//    *mbs_data++ = 0x07000000 + (module->reg->status_reg_2 & 0x00FF);  //
    meb empty
    CaenV785_empty_count++;
    if (CaenV785_empty_count % 500 == 0) {
      printf("counted %i trigger while empty meb\n", CaenV785_empty_count)
         ;
    }
  }

  // return number of bytes read
  return (char*)mbs_data - (char*)mbs_data_start;
}


// module setup
void CaenV785_SoftReset (CaenV785_module *module)
{
  module->reg->bit_set_1 = 0x80;
  module->reg->bit_clear_1 = 0x80;
  printf("software reset done\n");
}
```

```
void CaenV785_StandardSetup(CaenV785_module *module)
{
  CaenV785_ClearData(module);

  // deactivate mem test
  module->reg->bit_clear_2 = 0x1;

  module->reg->event_counter_reset = 1;

  //  module->reg->bit_clear_2 = 0x307b; // 0011 0000 0111 1011
  //  module->reg->bit_set_2 = 0x4980;   // 0100 1001 1000 0000

  CaenV785_SlidingScaleOn(module);
  CaenV785_StepThresholdOn(module);
  CaenV785_ZeroSuppressionOff(module);
  CaenV785_OverflowSuppressionOn(module);
  CaenV785_EnableAllChannels(module);
  CaenV785_ResetThresholds(module);


  printf("standard setup done\n");
}

void CaenV785_SlidingScaleOn(CaenV785_module *module) {
  // enable sliding scale + write 1 in unused bit
  module->reg->bit_set_2 = 0x00a0;
}

void CaenV785_SlidingScaleOff(CaenV785_module *module) {
  // disable sliding scale + write 0 in unused bit
  module->reg->bit_clear_2 = 0x00a0;
}

void CaenV785_StepThresholdOn(CaenV785_module *module)
{
  // low threshold values
  // threschold value = set threshold * 2
  module->reg->bit_set_2 = 0x100;
}

void CaenV785_StepThresholdOff(CaenV785_module *module)
{
  // low threshold values
  // threschold value = set threshold * 16
  module->reg->bit_clear_2 = 0x100;
}



void CaenV785_ZeroSuppressionOn(CaenV785_module *module)
```

```
{
  module->reg->bit_clear_2 = 0x10;  /* clear Bit 4 */
}


void CaenV785_ZeroSuppressionOff(CaenV785_module *module)
{
  module->reg->bit_set_2 = 0x10;    /* set Bit 4 */
}


void CaenV785_EnableAllChannels(CaenV785_module *module)
{
  int i;

  for (i=0; i<32; i++)
  {
    module->reg->thresholds[i] = module->reg->thresholds[i] & 0xfeff;   /*
        clear kill bit(8) */
  }
}


void CaenV785_DisableChannels(CaenV785_module *module, unsigned long mask)
{
  // ----- disable chosen channels
  //-----------------------------------------

  int i;
  unsigned long ch_bit=1;

  for (i=0; i<32; i++)
  {
    if (ch_bit & mask)
    {
      module->reg->thresholds[i] = module->reg->thresholds[i] | 0x0100;
         /* set kill bit(8) */
    }
    ch_bit=2*ch_bit;
  }
}


void CaenV785_SetKillThreshold(CaenV785_module *module, int channel) {
  // don't store values under the channel's threshold in MEB
  module->reg->thresholds[channel] = (module->reg->thresholds[channel] & 0
     xfeff) + 0x0100;
}
```

```
void CaenV785_SetSaveThreshold(CaenV785_module *module, int channel) {
  // store values under the channel's threshold in MEB
  module->reg->thresholds[channel] = module->reg->thresholds[channel] & 0
     xfeff;
}


void CaenV785_SetThreshold(CaenV785_module *module, int n, int val)
{
  // Set the threshold value [val] for the
  // n_th channel
  //   !!! be careful not to overwrite the "kill" bit (8)!
  //-------------------------------------
  short l_val;

  if (n>= 0 && n < 32 && val <= 255)
  {
    l_val = ((module->reg->thresholds[n] >> 8) << 8) + val;
    module->reg->thresholds[n] = l_val;
  }
}

void CaenV785_ResetThresholds(CaenV785_module *module)
{
  //  Reset/Initialize the V785 thresholds
  //---------------------------------------------

  int i;

  for(i=0 ; i < 32 ; i++)
  {
    CaenV785_SetThreshold(module, i, 0) ;
  }
}


void CaenV785_DisplayStoredThresholds(CaenV785_module *module)
{
  // Display thresholds stored in V785
  //-------------------------------------

  int i;
  char dummy[10];


  printf("\n Channel/Threshold/Kill-bit -> \n") ;
  for(i=0 ; i < 32; i++)
    printf(" %d/%d/%d ",i,module->reg->thresholds[i] & 0xff,
     (module->reg->thresholds[i] & 0x100)>>8) ;
```

```
    printf("\n");
    fflush(stdout);


    //F_ERROR(ERR__MSG_INFO,0,"Channel/Threshold/Kill-bit ->",l_pr_mask);
    //sprintf(c_line," ");
    for(i=0;i<32;i++)
    {
      //sprintf(dummy," %02d/%02d/%d ",i,module->reg->thresholds[i] & 0
         xff,
      //                  (module->reg->thresholds[i] & 0x100)>>8) ;
      //strcat(c_line,dummy);
      //if (strlen(c_line) >= 55)
      {
        //F_ERROR(ERR__MSG_INFO,0,c_line,l_pr_mask);
        //  sprintf(c_line," ");
      }
    }

    //strcat(c_line,".");
    //F_ERROR(ERR__MSG_INFO,0,c_line,l_pr_mask);

    return ;
}


void CaenV785_SetFileThresholds(CaenV785_module *module)
{
  // Read file "thresholds.v785"
  // and Set the treshold value [val] for the
  // n_th channel
  //-------------------------------------

  FILE *fp;
  int  i,ival;
  char txt[50];

  // ----- Open the thresholds file -----

  //sprintf(txt,"./caen/thresh_v785_%d.dat",card);
  if ((fp=fopen(txt,"r")) != NULL)
  {
    //    printf("\nFound file %s -> Read/Set ........", txt) ;

    //sprintf(c_line,"Found file %s -> Read/Set ...", txt);
    //F_ERROR(ERR__MSG_INFO,0,c_line,l_pr_mask);

    // ----- Get/Set the values -----
    //while(fgets(txt, 50, fp) != NULL)
    {
```

```
      //if (strchr("!@#$%*",txt[0]) != NULL || strlen(txt) < 2) continue ;
      //sscanf(txt, "%d %d ", &i, &ival) ;
      //CaenV785_SetThreshold(card, i, ival) ;
    }

    //fclose(fp) ;
    //    printf("....Ok! done\n") ;
  }
  else
    //    printf("\n*** file %s NOT-Found --> No-Action ***\n", txt) ;
  {
    //sprintf(c_line,"file %s NOT found --> No Action!", txt);
    //F_ERROR(ERR__MSG_INFO,0,c_line,l_pr_mask);
  }
}


int CaenV785_CheckDreadyStatus(CaenV785_module *module)
{
  //  this function checks Status Register 1
  //  to see if the DREADY bit (#0) is set or not
  int dready;

  dready = module->reg->status_reg_1 & 0x1;
  return dready ;
}


void CaenV785_ClearData(CaenV785_module *module)
{
  //  this function clears data, read & write pointers,
  //  event counter and QAC sections...
  //  by toggling bit 2 of register 2

  module->reg->bit_set_2 = 0x4;
  module->reg->bit_clear_2 = 0x4;
}


int CaenV785_CheckBufferStatus(CaenV785_module *module)
{
  //  this function checks Status Register 2
  //  to see if the output buffer contains data
  //  return val: 0: buffer empty, 2: buffer full, 1: not empty - not full

  int l_val,
  buffstat;

  l_val = module->reg->status_reg_2 ;
  if (l_val & 0x2)        /* buffer empty */
```

```
    buffstat=0 ;
  else
  {
    if (l_val & 0x4)       /* buffer full */
      buffstat=2 ;
    else
      buffstat=1 ;         /* something between empty and full */
  }

  return buffstat ;
}


int CaenV785_CheckBusyStatus(CaenV785_module *module)
{
  // checks BUSY bit in status register 1 (#2) if module is busy (1) or
     not (0)
  return ((module->reg->status_reg_1 >> 2) & 0x1);
}


void CaenV785_OverflowSuppressionOn(CaenV785_module *module)
{
  module->reg->bit_clear_2 = 0x8;  /* clear Bit 3 */
}


void CaenV785_OverflowSuppressionOff(CaenV785_module *module)
{
  module->reg->bit_set_2 = 0x8;    /* set Bit 3 */
}
```

A.8 listmode2plot.c

Beim schreiben dieses Programms wurde eine doppelt verkettete Liste zum internen speichern des Histograms gewählt. Dies sollte Platz sparen und auf ADCs mit höherer Auflösung erweiterbar sein. Da im Nachhinein die Kanäle, die hier eingespart wurden zum plotten und fitten wieder hinzugefügt werden mussten wäre eine optimalere Lösung statt dessen ein Array mit 4096 Einträgen zu verwenden. Dies würde vermutlich auch die Rechenzeit beim konvertieren verringern.

```
/*
 * listmode2plot.c
 *
 * converts MBS listmode file to plottable histograms
 *
 * 08/2012 Henno Lauinger
 *
 */

#include <stdio.h>
```

```
#include <stdlib.h>
#include <math.h>
#include <string.h>

#include "typedefs.h"
#include "s_filhe_swap.h"
#include "s_bufhe_swap.h"
#include "s_ve10_1.h"
#include "s_ves10_1.h"
#include "s_evhe_swap.h"
#include "f_evt.h"

#include "histogram_structs.h"
#include "adc_785.h"
#include "adc_785_debug_tools.h"


static struct energy_count* energy_hist[32];
static struct energy_count* energy_hist_all;
unsigned int evt_counter = 0;
unsigned int evt_k_counter = 0;
static char* file_out_name = "plotfiles/output.dat";


int readListmodeFile(char* filename) {
  s_evt_channel *input_channel;
  s_filhe *file_header;
  s_bufhe *buffer_header;

  int32_t *event_data;
  int events_read = 0;

  // initialize the input channel
  input_channel = f_evt_control();


    /*+   first argument of f_evt_get_open()    : Type of server:
       */
    /*-               GETEVT__FILE   : Input from file
       */
    /*-               GETEVT__STREAM : Input from MBS stream server
       */
    /*-               GETEVT__TRANS  : Input from MBS transport
       */
    /*-               GETEVT__EVENT  : Input from MBS event server
       */
    /*-               GETEVT__REVSERV: Input from remote event server
       */
```

```
  //   second argument of f_evt_get_open()    : name of server (for ex.
     "r4-4" in case of online analysis or "filename.lmd" in case of
     reading data from file)
  int32_t result = f_evt_get_open(GETEVT__FILE, filename, input_channel,
      (CHARS**) (&file_header), 1, 0);

  printf("result = %i\n", result);

  if (result != GETEVT__SUCCESS)
    return 1;

  if (file_header != 0)
  {
    printf("filhe_dlen : %i\n", file_header->filhe_dlen);
    printf("filhe_file : %s\n", file_header->filhe_file);
    printf("filhe_user : %s\n", file_header->filhe_user);
  }

  printf("\n");


  for(;;)
  {
    result = f_evt_get_event(input_channel, &event_data, (INTS4**)(&
       buffer_header));
    if (result != GETEVT__SUCCESS)
break;

    ++events_read;
    // uncomment the following lines to output the "raw data and header
       info from the event"
    /*      std::cout << std::endl << "processing event number " << std
       ::dec << events_read << std::endl;
    *
    *   std::cout << "f_evt_type(...) output: " << std::endl;
    *   f_evt_type(buffer_header, (s_evhe*)event_data, -1, 0,1,0);
    *   std::cout <<
       "----------------------------------------------------" << std::
       endl;*/

    int sub;
    for (sub = 1; result != GETEVT__NOMORE; ++sub)
    {
s_ves10_1 *subevent_header;
int32_t *data;
int32_t length;

result = f_evt_get_subevent((s_ve10_1*)event_data, sub, (int32_t**)&
   subevent_header, &data, &length);
```

```
  if (result == GETEVT__SUCCESS)
  {
    // do something with the subevent data
    output(data, length);
  }
      }

    }
    printf(" read %i events \n", events_read);

    f_evt_get_close(input_channel);
}



int output(int32_t *data, size_t length) {
  // write data to struct
  int i;
  for (i = 0; i < length; i++){
    int32_t type = getType(*data);
    if (type == type_data) {
/*      addNewEnergyNode(getData(*data), energy_hist[getChannel(*data)]);
      addNewEnergyNode(getData(*data), energy_hist_all);*/
      addEnergySorted(getData(*data), energy_hist[getChannel(*data)]);
      addEnergySorted(getData(*data), energy_hist_all);
    }
    else if (type != type_header && type != type_eob) {
      char* string = (char*) malloc(33);
      toBinaryString(*data, string);
      printf("no data (%i): %i/%i\t%s\n", type, i+1, (int) length, string)
         ;
      free(string);
    }
    data++;
  }


  if (evt_counter++ >= 100000) {
    evt_k_counter++;
    fprintf(stderr, "writing first %i events\n", evt_k_counter * 100000);
    writeData();
    fprintf(stderr, "done\n");
    evt_counter = 0;
  }

  return 0;
}
```

```
int writeData() {

  // open files
  FILE* f_out = fopen(file_out_name, "wb");
  if (f_out == NULL) {
    fprintf(stderr, "couldn't open output file\n");
    return 1;
  }
  fprintf(f_out, "# energy\tcount\r\n");

  FILE* f_ch_out[32];
  int32_t i;
  char* f_ch_name = (char*) malloc(19);
  for (i = 0; i < 32; i++) {
    sprintf(f_ch_name, "plotfiles/ch%i.dat", i);
    f_ch_out[i] = fopen(f_ch_name, "wb");
      if (f_out == NULL) {
      fprintf(stderr, "couldn't open output file\n");
      return 1;
    }
    fprintf(f_ch_out[i], "# energy\tcount\r\n");
  }

  // write memory to plottable files

  // combined
/*  printEnergyList(energy_hist_all);
  energy_hist_all = getFirstEnergyNode(sortEnergyNodes(energy_hist_all));
  fprintf(stderr, "2\n");*/
  energy_hist_all = getFirstEnergyNode(energy_hist_all);
  fprintf(f_out, "%i\t%i\r\n", energy_hist_all->energy, energy_hist_all->
     count);
  while (energy_hist_all->next_energy != NULL) {
    energy_hist_all = energy_hist_all->next_energy;
    fprintf(f_out, "%i\t%i\r\n", energy_hist_all->energy, energy_hist_all
       ->count);
  }
  fclose(f_out);

  // seperate channels
  for (i = 0; i < 32; i++) {
//    energy_hist[i] = getFirstEnergyNode(sortEnergyNodes(energy_hist[i]))
   ;
    energy_hist[i] = getFirstEnergyNode(energy_hist[i]);
    fprintf(f_ch_out[i], "%i\t%i\r\n", energy_hist[i]->energy, energy_hist
       [i]->count);
    while (energy_hist[i]->next_energy != NULL) {
      energy_hist[i] = energy_hist[i]->next_energy;
      fprintf(f_ch_out[i], "%i\t%i\r\n", energy_hist[i]->energy,
         energy_hist[i]->count);
```

```
    }
    fclose(f_ch_out[i]);
  }


  return 0;
}



int main(int argc, char *argv[]) {

  // usage
  if (argc != 2) {
    printf("usage: \n\t%s <listmode file>\n", argv[0]);
    printf(
      "\t\t converts file.lmd to file.bit\n");
    return 1;
  }

  // get input file name and generate output file name
  int name_length = (int) strlen(argv[1]);
  char file_name[name_length];
  char file_base[name_length];
  char file_ext[4];
  file_out_name = (char*) malloc(name_length + 10);
  strcpy(file_name, argv[1]);
  strncpy(file_ext, file_name+name_length-3, 3);
  strncpy(file_base, file_name, name_length-4);
  file_base[name_length-4] = '\0';
//  file_base[sizeof(file_base)-1] = '\0';

  strcpy(file_out_name, "plotfiles/");
  strcat(file_out_name, file_base);
  strcat(file_out_name, ".dat");

  printf("in:\t%s\nbase:\t%s\next:\t%s\nout:\t%s\n", file_name, file_base,
      file_ext, file_out_name);



  // initialize histogram_structs and open output files

  energy_hist_all = newEnergyNode(0);
  int i = 0;
  for (i = 0; i < 32; i++) {
    energy_hist[i] = newEnergyNode(0);
  }

  // read listmode file
```

```
  readListmodeFile(file_name);

  //write all data

  writeData();


//  fclose(f_in);
  return 0;
}
```

A.9 histogram_structs.c

```
#include "histogram_structs.h"
#include <stdlib.h>
#include <stdio.h>


/**
 * creates a new energy node
 */
struct energy_count* newEnergyNode(int32_t energy) {
  static struct energy_count* result;
  result = (struct energy_count*) malloc(sizeof(struct energy_count));

  if (result == NULL) {
    perror("couldn't allocate memory for energy node");
    exit(EXIT_FAILURE);
  }

  result->energy = energy;
  result->count = 1;
  result->prev_energy = NULL;
  result->next_energy = NULL;

  return result;
}

/**
 * adds an existing energy node to a list of energy nodes
 *
 * if the list allready contained the energy, the
 * counts will be added
 */
void addEnergyNode(struct energy_count* newNode, struct energy_count*
   listNode) {
  struct energy_count* tmpNode = getEnergyNode(newNode->energy, listNode);
  if (tmpNode == NULL) {
```

```
    newNode->prev_energy = listNode;
    newNode->next_energy = listNode->next_energy;
    if (newNode->next_energy != NULL) {
      newNode->next_energy->prev_energy = newNode;
    }
    listNode->next_energy = newNode;
  }
  else {
    tmpNode->count += newNode->count;
  }
}

/**
 * adds a new energy node to a list of energy nodes
 *
 * if the list allready contained the energy, one
 * count will be added
 */
void addNewEnergyNode(int32_t energy, struct energy_count* listNode) {
  struct energy_count* tmpNode = getEnergyNode(energy, listNode);
  if (tmpNode == NULL) {
    fprintf(stderr, "adding new energy %i to list\n", energy);
    printEnergyList(listNode);
    free(tmpNode);
    tmpNode = newEnergyNode(energy);
    tmpNode->prev_energy = listNode;
    tmpNode->next_energy = listNode->next_energy;
    if (tmpNode->next_energy != NULL) {
      tmpNode->next_energy->prev_energy = tmpNode;
    }
    listNode->next_energy = tmpNode;
  }
  else {
    tmpNode->count++;
  }
}

/**
 * sorts an energy into a sorted list
 *
 * if the list allready contained the energy, one
 * count will be added
 *
 * @returns the energy's node in the sorted list; NULL on error
 */
struct energy_count* addEnergySorted(int32_t energy, struct energy_count*
   sortedListNode){
  if (sortedListNode->energy == energy) {
    sortedListNode->count++;
    return sortedListNode;
```

```
  }
  if (sortedListNode->energy > energy) {

    if (sortedListNode->prev_energy == NULL) {

      struct energy_count* newNode = newEnergyNode(energy);
      newNode->prev_energy = NULL;
      newNode->next_energy = sortedListNode;
      sortedListNode->prev_energy = newNode;

      return newNode;

    } else if (sortedListNode->prev_energy->energy < energy) {

      struct energy_count* newNode = newEnergyNode(energy);
      newNode->prev_energy = sortedListNode->prev_energy;
      newNode->next_energy = sortedListNode;
      sortedListNode->prev_energy = newNode;
      newNode->prev_energy->next_energy = newNode;

      return newNode;

    } else {

      return addEnergySorted(energy, sortedListNode->prev_energy);
    }
  } else {
    if (sortedListNode->next_energy == NULL) {

      struct energy_count* newNode = newEnergyNode(energy);
      newNode->next_energy = NULL;
      newNode->prev_energy = sortedListNode;
      sortedListNode->next_energy = newNode;

      return newNode;

    } else if (sortedListNode->next_energy->energy > energy) {

      struct energy_count* newNode = newEnergyNode(energy);
      newNode->next_energy = sortedListNode->next_energy;
      newNode->prev_energy = sortedListNode;
      sortedListNode->next_energy = newNode;
      newNode->next_energy->prev_energy = newNode;

      return newNode;

    } else {

      return addEnergySorted(energy, sortedListNode->next_energy);
    }
```

```
  }

  return NULL;
}

/**
 * @returns the energy node with given energy from listNode;
 *    NULL if list didn't contain given energy
 */
struct energy_count* getEnergyNode(int32_t energy, struct energy_count*
   listNode) {
  if (listNode->energy == energy) {
    return listNode;
  }
  struct energy_count* tmpNode = listNode;

  // search previous nodes
  while (tmpNode->prev_energy != NULL) {
    tmpNode = tmpNode->prev_energy;
    if (tmpNode->energy == energy) {
      return tmpNode;
    }
    if (tmpNode == listNode) {
      fprintf(stderr, "searched for energy %i previous to node %x (%i)\n",
          energy, listNode, listNode->energy);
      fprintf(stderr, "loop in list while searching for energy; breaking\n
         ");
      return NULL;
    }
  }

  // search next nodes
  while (tmpNode->next_energy != NULL) {
    tmpNode = tmpNode->next_energy;
    if (tmpNode->energy == energy) {
      return tmpNode;
    }
    if (tmpNode == listNode) {
      fprintf(stderr, "searched for energy %i after node %x (%i)\n",
         energy, listNode, listNode->energy);
      fprintf(stderr, "loop in list while searching for energy; breaking\n
         ");
      return NULL;
    }
  }

  // energy not found
  return NULL;
}
```

```
/**
 * @returns counted events with given energy_count
 */
int32_t getCount(int32_t energy, struct energy_count* listNode) {
  struct energy_count* tmpNode = getEnergyNode(energy, listNode);
  if (tmpNode == NULL) {
    return 0;
  }
  else {
    return tmpNode->count;
  }
}

/**
 * sorts the energy nodes for energies
 *
 * @returns last node (highest energy)
 */
struct energy_count* sortEnergyNodes(struct energy_count* listNode) {
  fprintf(stderr, "sort %x\n", listNode);
  struct energy_count* tmpNode = getFirstEnergyNode(listNode);
  tmpNode = swapSortEnergyNodes(tmpNode);

  return tmpNode;
}

/**
 * @returns first node in list
 */
struct energy_count* getFirstEnergyNode(struct energy_count* listNode) {
//  fprintf(stderr, "getFirstEnergyNode %x\n", listNode);
  if (listNode == NULL) {
    fprintf(stderr, "getFirstEnergyNode: listNode NULL\n");
    exit(1);
  }
  if (listNode->prev_energy == NULL) {
    return listNode;
  }
  else {
    return getFirstEnergyNode(listNode->prev_energy);
  }
}

/**
 * swaps two items from a list
 */
void swapEnergyNodes(struct energy_count* firstNode, struct energy_count*
   secondNode) {
```

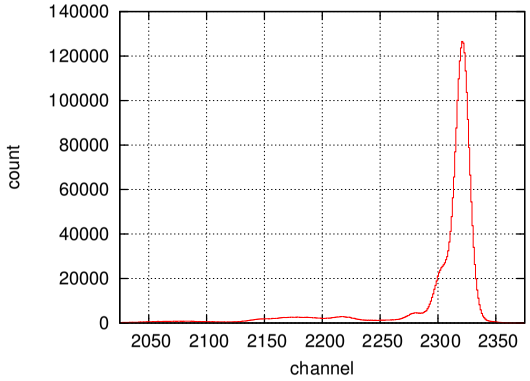
```
  if (firstNode == NULL || secondNode == NULL || firstNode->prev_energy ==
      firstNode
    || firstNode->next_energy == firstNode || secondNode->prev_energy ==
       secondNode
    || secondNode->next_energy == secondNode) {
      fprintf(stderr, "couldn't swap; list order corrupted\n");
      exit(1);
    }

  struct energy_count* oldPrevA = firstNode->prev_energy;
  struct energy_count* oldNextA = firstNode->next_energy;
  struct energy_count* oldPrevB = secondNode->prev_energy;
  struct energy_count* oldNextB = secondNode->next_energy;

  // nodes are adjacent
  if (oldNextA == secondNode || oldPrevA == secondNode) {
    if (swapAdjacentEnergyNodes(firstNode, secondNode) != 0) {
      fprintf(stderr, "couldn't swap adjacent nodes\n");
      exit(1);
    }
  }
  // nodes are not adjacent
  else {
    firstNode->prev_energy = oldPrevB;
    firstNode->next_energy = oldNextB;

    secondNode->prev_energy = oldPrevA;
    secondNode->next_energy = oldNextA;

    oldPrevA->next_energy = secondNode;
    oldNextA->prev_energy = secondNode;

    oldPrevB->next_energy = firstNode;
    oldNextB->prev_energy = firstNode;
  }


}

/**
 * swaps two adjacent items from a list
 */
int32_t swapAdjacentEnergyNodes(struct energy_count* firstNode, struct
   energy_count* secondNode) {
  if (firstNode == NULL || secondNode == NULL) {
    return 1;
  }

  // get old pointers
  struct energy_count* oldPrevA = firstNode->prev_energy;
```

```
  struct energy_count* oldNextA = firstNode->next_energy;
  struct energy_count* oldPrevB = secondNode->prev_energy;
  struct energy_count* oldNextB = secondNode->next_energy;

  if (firstNode->next_energy == secondNode && secondNode->prev_energy ==
     firstNode) {
    // swap pointers of firstNode and secondNode
    firstNode->prev_energy = secondNode;
    firstNode->next_energy = oldNextB;
    secondNode->prev_energy = oldPrevA;
    secondNode->next_energy = firstNode;

    // change pointers to firstNode and secondNode
    if (oldPrevA != NULL) {
      oldPrevA->next_energy = secondNode;
    }

    if (oldNextB != NULL) {
      oldNextB->prev_energy = firstNode;
    }

    return 0;
  }

  if (firstNode->prev_energy == secondNode && secondNode->next_energy ==
     firstNode) {
    // swap pointers of firstNode and secondNode
    firstNode->prev_energy = oldPrevA;
    firstNode->next_energy = secondNode;
    secondNode->prev_energy = firstNode;
    secondNode->next_energy = oldNextB;

    // change pointers to firstNode and secondNode
    if (oldPrevB != NULL) {
      oldPrevB->next_energy = firstNode;
    }

    if (oldNextA != NULL) {
      oldNextA->prev_energy = secondNode;
    }

    return 0;
  }

  return 1;
}

/**
 * sorts energy nodes for energies by swapping listNode with previous node
 * if energy is lower
```

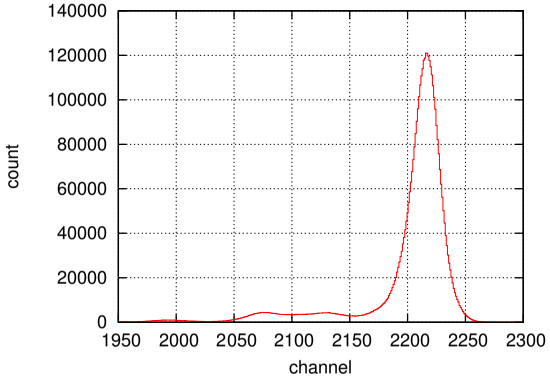
```
 * begin with first node
 */
struct energy_count* swapSortEnergyNodes(struct energy_count* listNode) {
/*  if (listNode->prev_energy == NULL) {
    fprintf(stderr, "first\n");
  } else if (listNode->next_energy == NULL) {
    fprintf(stderr, "last\n");
  } else {
    fprintf(stderr, "swapSortEnergyNodes ln: %x (%i)\tp: %x (%i)\tn: %x (%
       i)\n", listNode, listNode->energy, listNode->prev_energy, listNode
       ->prev_energy->energy, listNode->next_energy, listNode->next_energy
       ->energy);
  }*/
//  printEnergyNodes(listNode);
  if (listNode->prev_energy == NULL) {
    if (listNode->next_energy == NULL) {
      return listNode;
    }
    else {
      return swapSortEnergyNodes(listNode->next_energy);
    }
  }
  else {
    if (listNode->prev_energy->energy > listNode->energy) {
      swapEnergyNodes(listNode->prev_energy, listNode);
      return swapSortEnergyNodes(listNode);
    }
    else {
      if (listNode->next_energy == NULL) {
  return listNode;
      }
      else {
  return swapSortEnergyNodes(listNode->next_energy);
      }
    }
  }
}

/**
 * prints energy/counts
 */
void printEnergyNodes(struct energy_count* listNode) {
  struct energy_count* tmpNode = getFirstEnergyNode(listNode);

  printf("energy\tcounts\n");
  printf("%i\t%i\n", tmpNode->energy, tmpNode->count);

  while (tmpNode->next_energy != NULL) {
    tmpNode = tmpNode->next_energy;
    printf("%i\t%i\n", tmpNode->energy, tmpNode->count);
```

```
  }

  printf("\n");
}

/**
 * prints energy list
 */
void printEnergyList(struct energy_count* listNode) {
  struct energy_count* tmpNode = getFirstEnergyNode(listNode);
  if (tmpNode->next_energy == NULL) {
    fprintf(stderr, "only one energy: %i\n", tmpNode->energy);
  } else {
    fprintf(stderr, "\t<- %i ->\t%i\n", tmpNode->energy, tmpNode->
       next_energy->energy);
    while (tmpNode->next_energy->next_energy != NULL) {
      tmpNode = tmpNode->next_energy;
      fprintf(stderr, "%i\t<- %i ->\t%i\n", tmpNode->prev_energy->energy,
         tmpNode->energy, tmpNode->next_energy->energy);
    }
    fprintf(stderr, "%i\t<- %i ->\n", tmpNode->energy, tmpNode->
       next_energy->energy);
  }
}
```

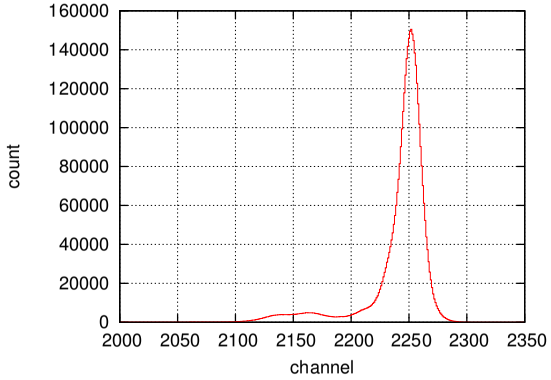
A.10 Spektren



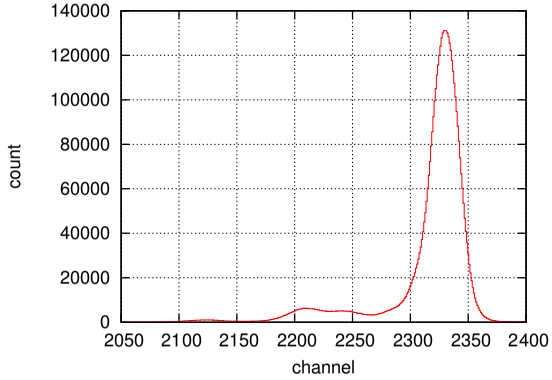
(a) Detektor 2.5



(b) Detektor 3.4

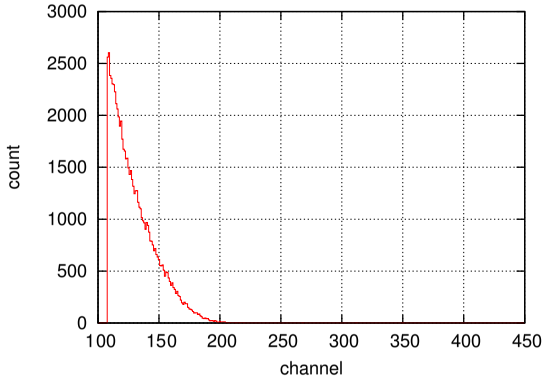


(c) Detektor 3.5

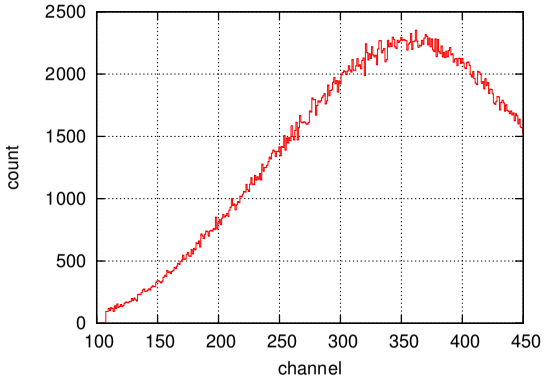


(d) Detektor 3.6

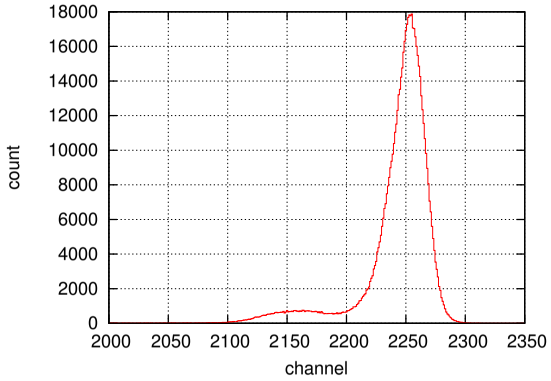
Abbildung 19.: Direkte Bestrahlung von Detektor 3.5 und benachbarten Detektoren. Dauer: 12 h. Der Strom an Detektor 3.5 betrug am Ende der Messung 560 nA.



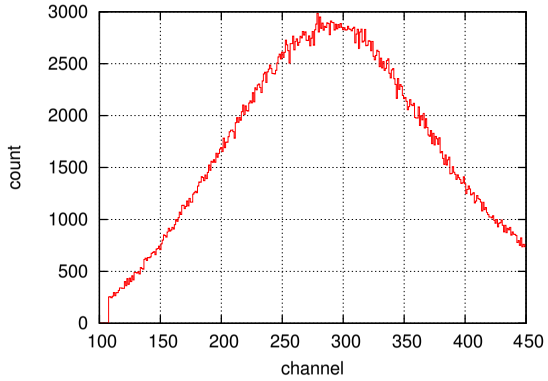
(a) Detektor 2.5



(b) Detektor 3.4

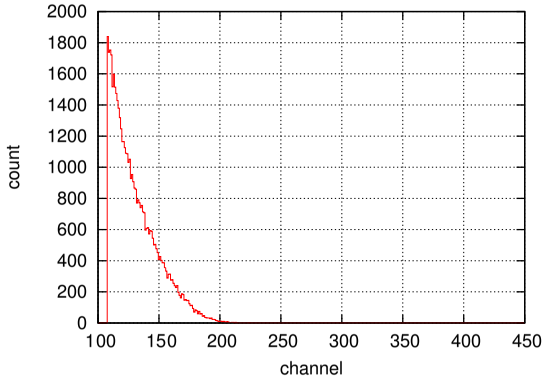


(c) Detektor 3.5

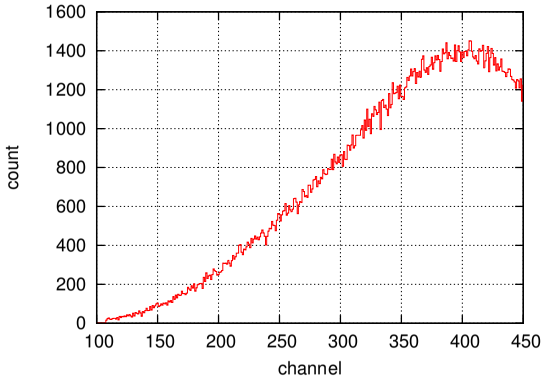


(d) Detektor 3.6

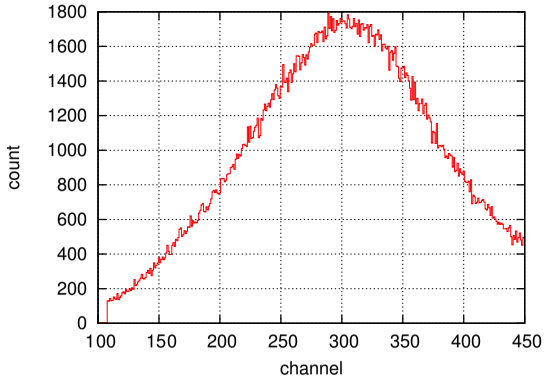
Abbildung 20.: Direkte Bestrahlung von Detektor 3.5 und benachbarten Detektoren. Dauer: $2\,\mathrm{h}$. Nur an Detektor 3.5 lag Spannung an und nur er sorgte für einen Trigger. Der Strom an Detektor 3.5 betrug am Ende der Messung $610\,\mathrm{nA}$.



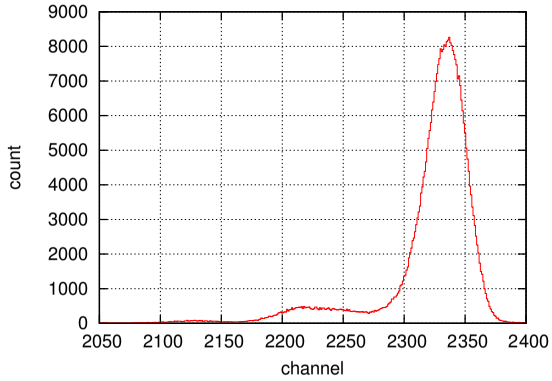
(a) Detektor 2.5



(b) Detektor 3.4

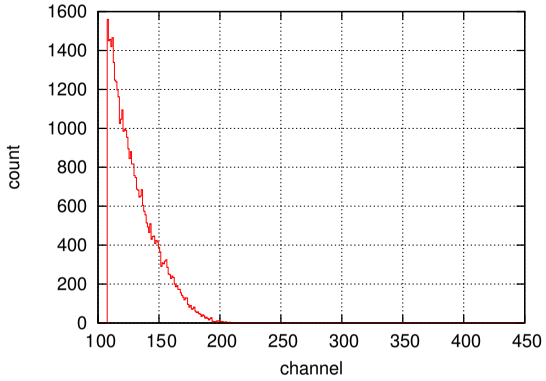


(c) Detektor 3.5

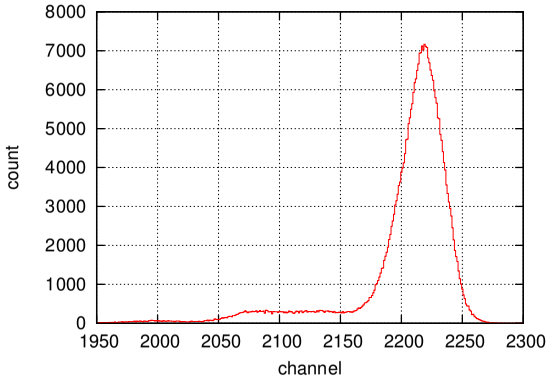


(d) Detektor 3.6

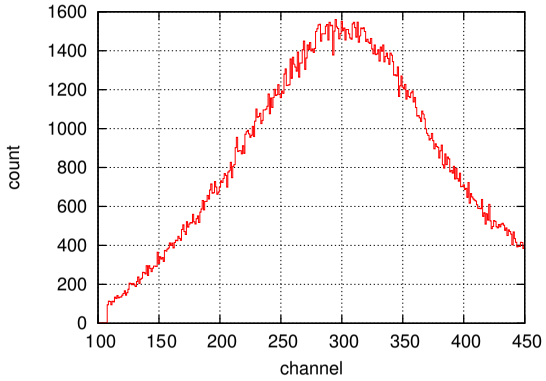
Abbildung 21.: Direkte Bestrahlung von Detektor 3.5 und benachbarten Detektoren. Dauer: 1 h. Nur an Detektor 3.6 lag Spannung an und nur er sorgte für einen Trigger.



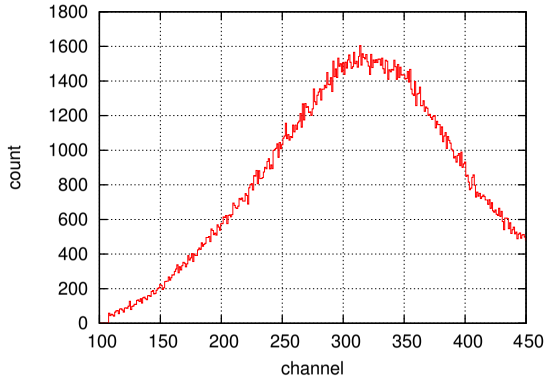
(a) Detektor 2.5



(b) Detektor 3.4

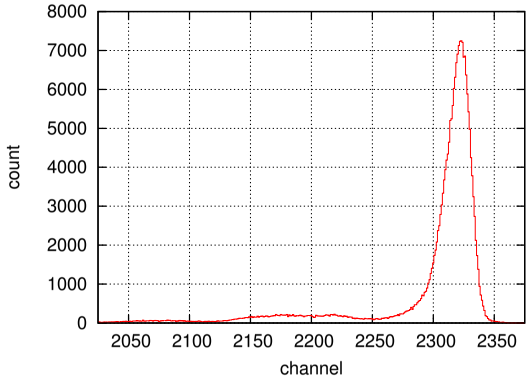


(c) Detektor 3.5

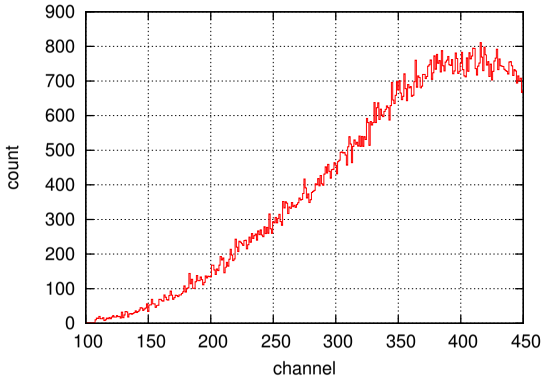


(d) Detektor 3.6

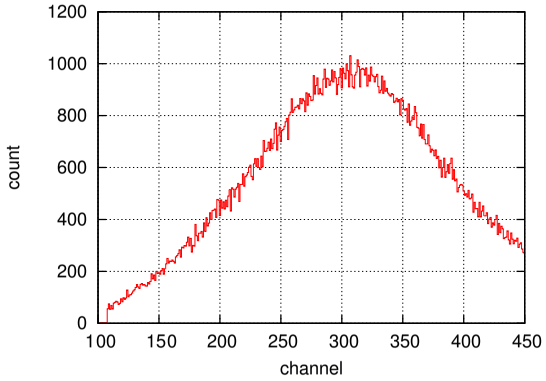
Abbildung 22.: Direkte Bestrahlung von Detektor 3.5 und benachbarten Detektoren. Dauer: 1 h. Nur an Detektor 3.4 lag Spannung an und nur er sorgte für einen Trigger.



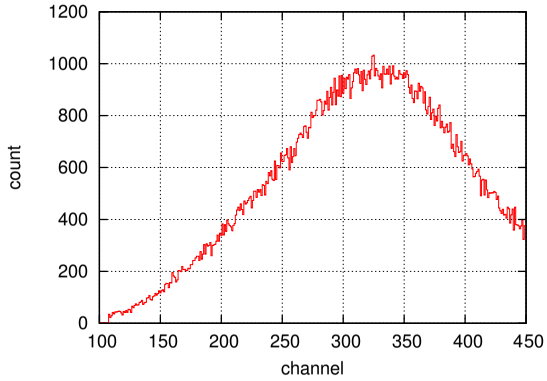
(a) Detektor 2.5



(b) Detektor 3.4

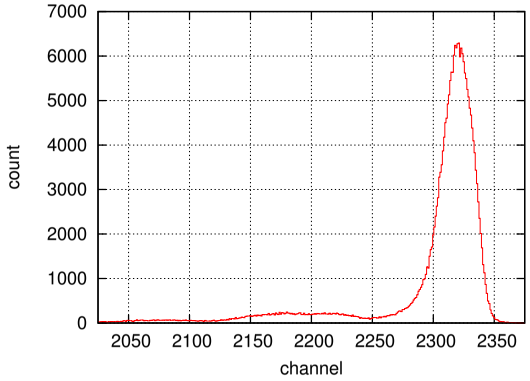


(c) Detektor 3.5

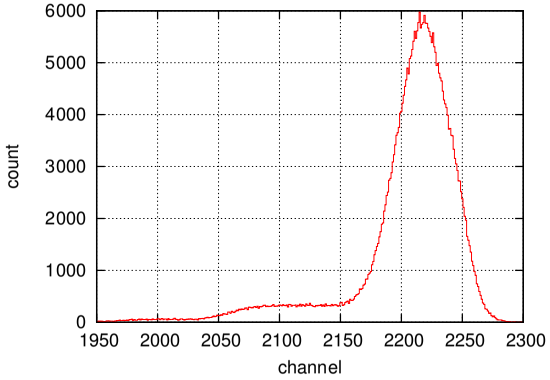


(d) Detektor 3.6

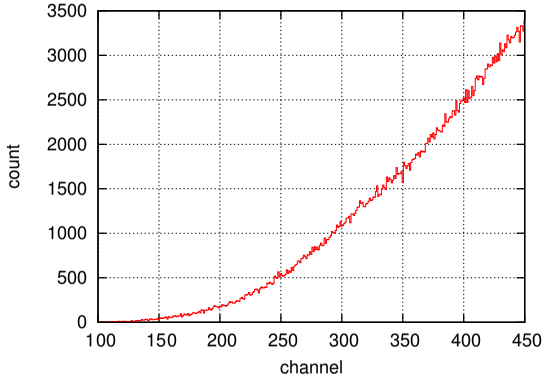
Abbildung 23.: Direkte Bestrahlung von Detektor 3.5 und benachbarten Detektoren. Dauer: 1 h. Nur an Detektor 2.5 lag Spannung an und nur er sorgte für einen Trigger.



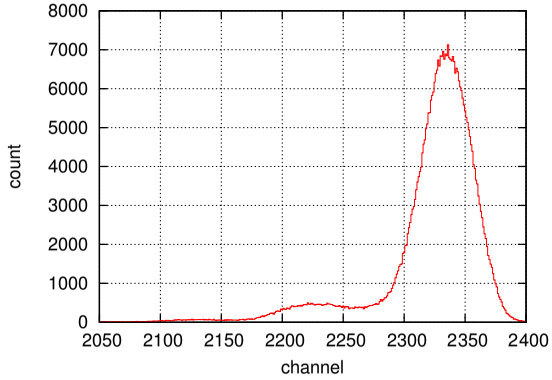
(a) Detektor 2.5



(b) Detektor 3.4

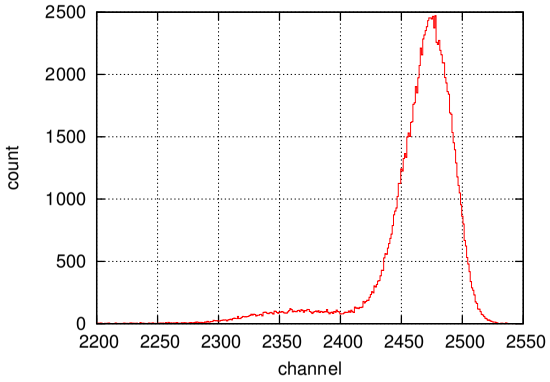


(c) Detektor 3.5

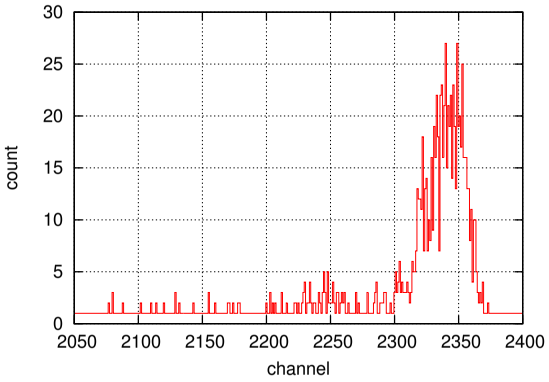


(d) Detektor 3.6

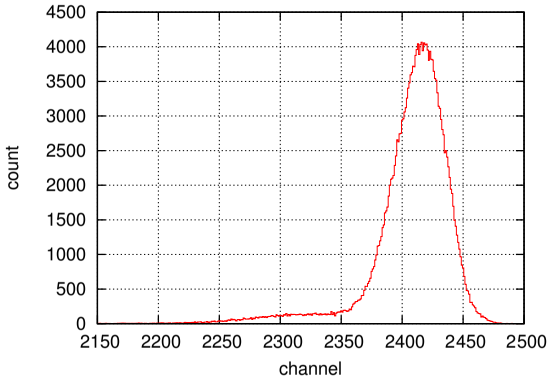
Abbildung 24.: Direkte Bestrahlung von Detektor 3.5 und benachbarten Detektoren. Dauer: 1 h. An Detektor 3.5 lag keine Spannung an und er trug nicht zum Trigger bei.



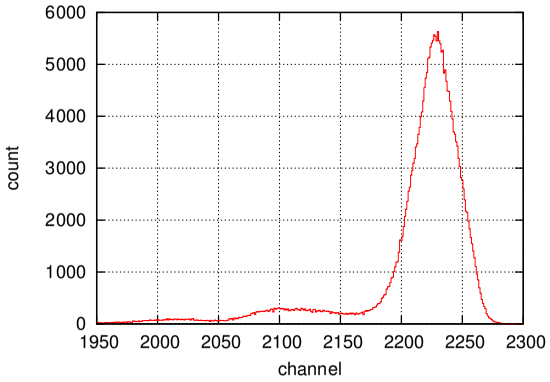
(a) Detektor 2.2



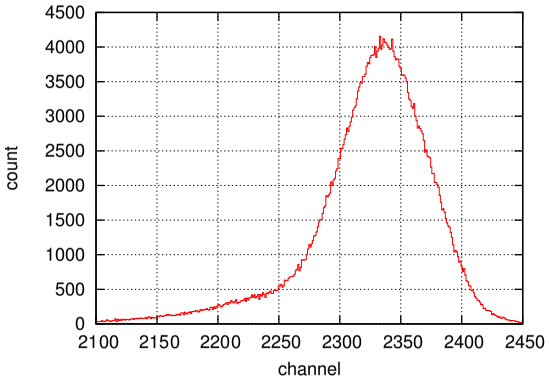
(b) Detektor 2.3



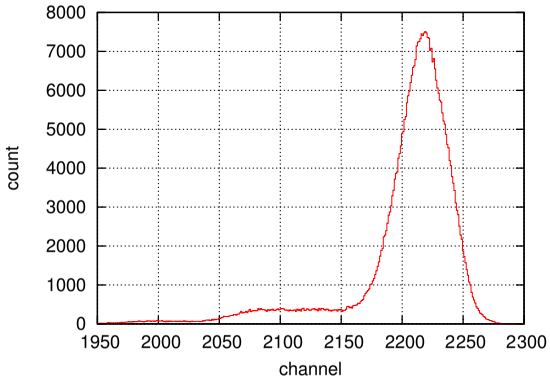
(c) Detektor 2.4



(d) Detektor 3.2

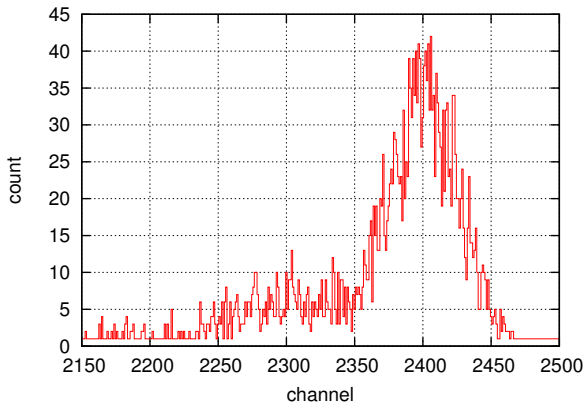


(e) Detektor 3.3

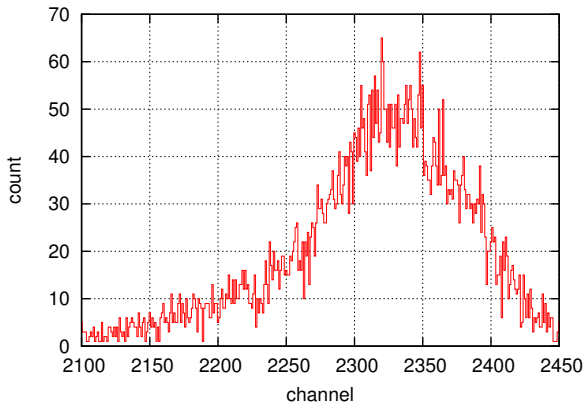


(f) Detektor 3.4

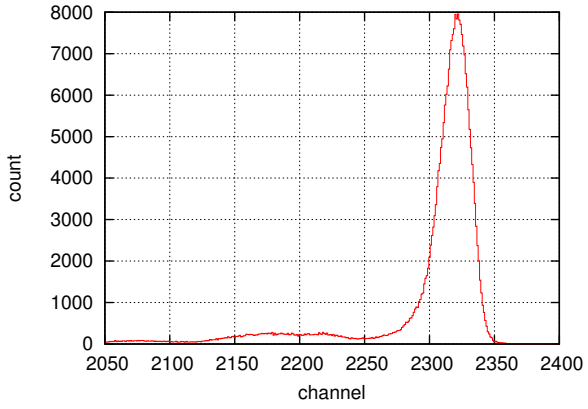
Abbildung 25.: Direkte Bestrahlung von Detektor 3.5 und benachbarten Detektoren. Dauer: $1,2\,\mathrm{h}$. Alle Detektoren außer 1.5-1.11 sind mit Spannung versorgt. Detektoren 1.3, 1.5, 2.2-2.6 und 3.2-3.6 trugen zum Trigger bei. Die anderen Plots befinden sich in Abbildung 26.



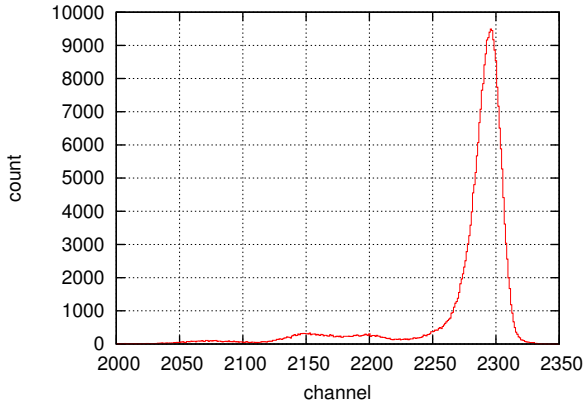
(a) Detektor 1.3



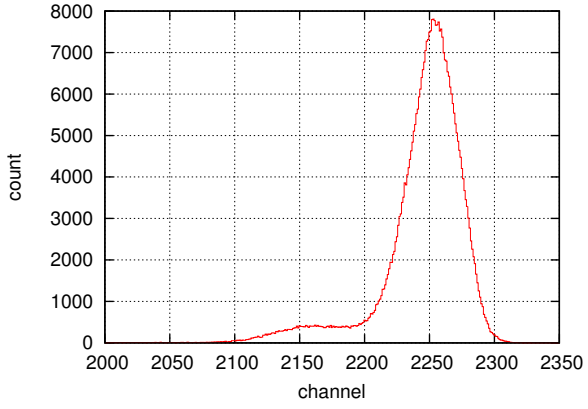
(b) Detektor 1.5



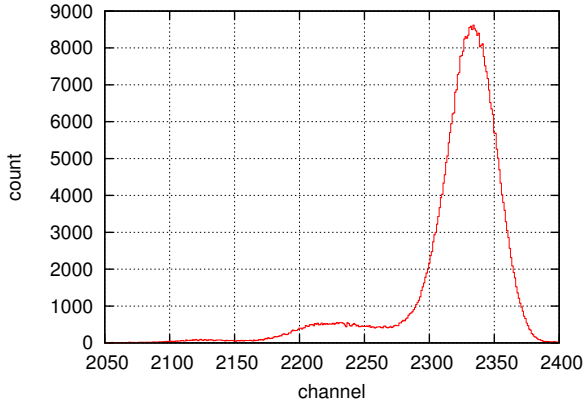
(c) Detektor 2.5



(d) Detektor 2.6

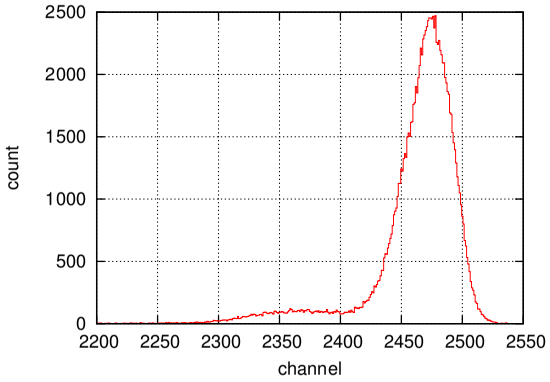


(e) Detektor 3.5

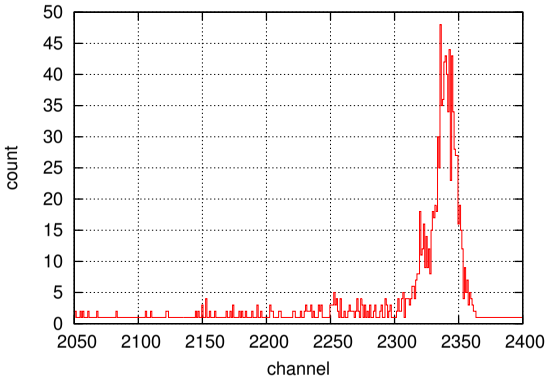


(f) Detektor 3.6

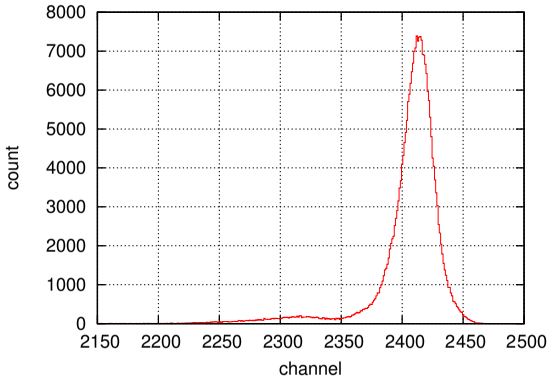
Abbildung 26.: Direkte Bestrahlung von Detektor 3.5 und benachbarten Detektoren. Dauer: $1,2\,\mathrm{h}$. Alle Detektoren außer 1.5-1.11 sind mit Spannung versorgt. Detektoren 1.3, 1.5, 2.2-2.6 und 3.2-3.6 trugen zum Trigger bei. Die anderen Plots befinden sich in Abbildung 25.



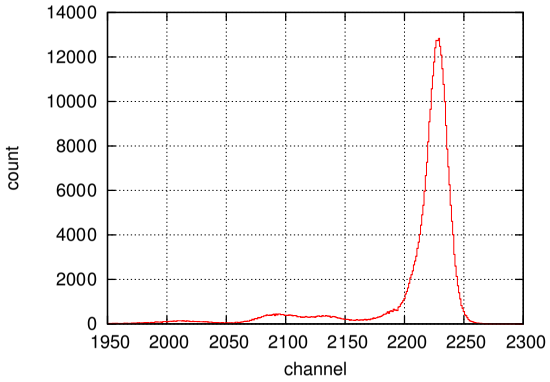
(a) Detektor 2.2



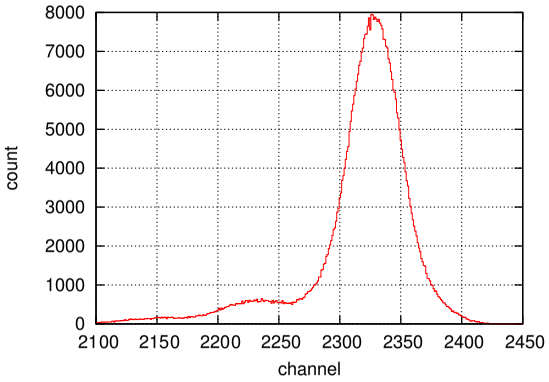
(b) Detektor 2.3



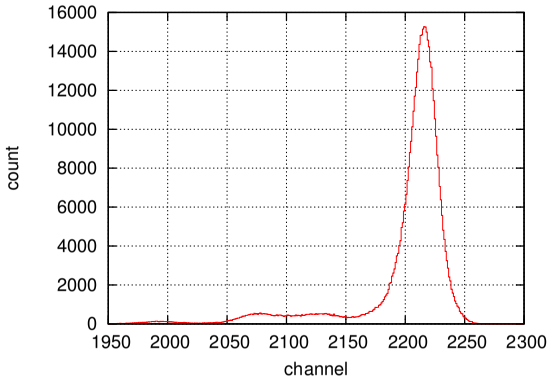
(c) Detektor 2.4



(d) Detektor 3.2

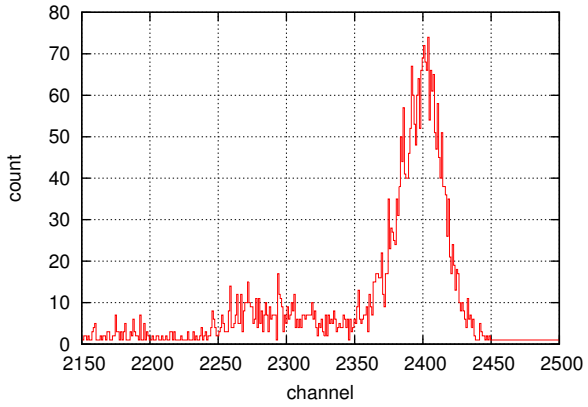


(e) Detektor 3.3

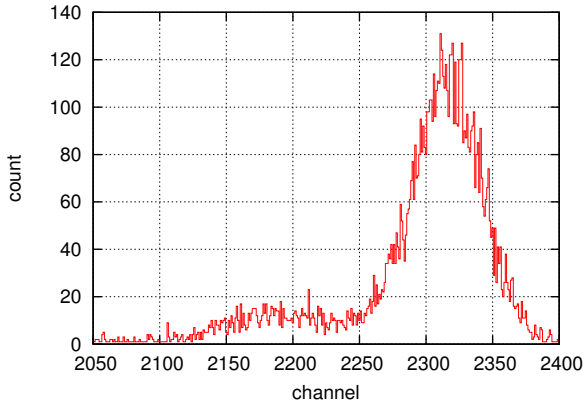


(f) Detektor 3.4

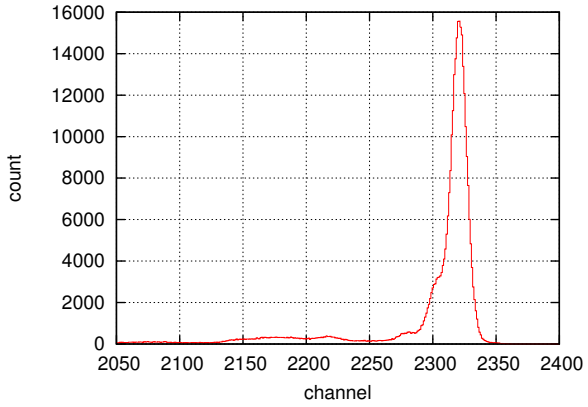
Abbildung 27.: Direkte Bestrahlung von Detektor 3.5 und benachbarten Detektoren. Dauer: $1,2\,\mathrm{h}$. Alle Detektoren außer 1.5-1.11 sind mit Spannung versorgt. Detektoren 1.3, 1.5, 2.2-2.6 und 3.2-3.6 trugen zum Trigger bei. Die anderen Plots befinden sich in Abbildung 26.



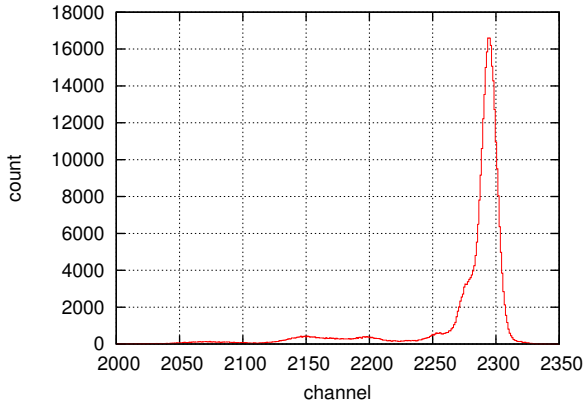
(a) Detektor 1.3



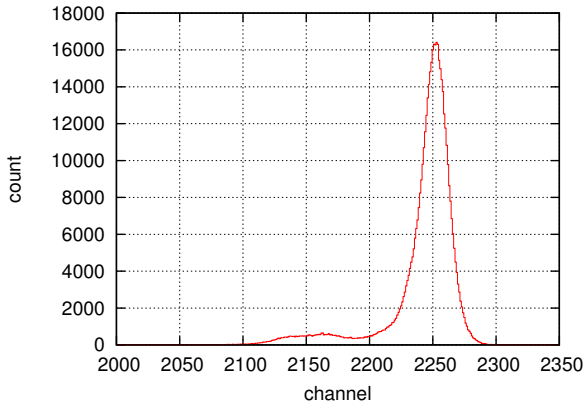
(b) Detektor 1.5



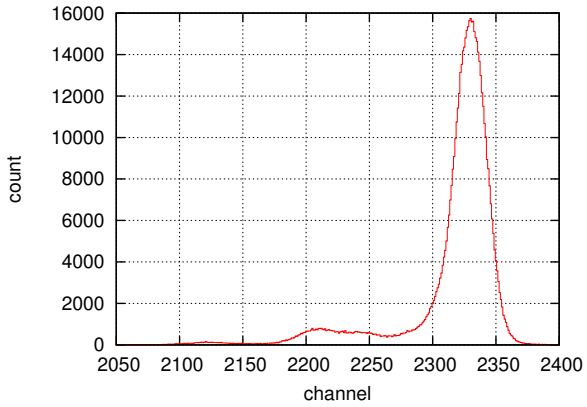
(c) Detektor 2.5



(d) Detektor 2.6

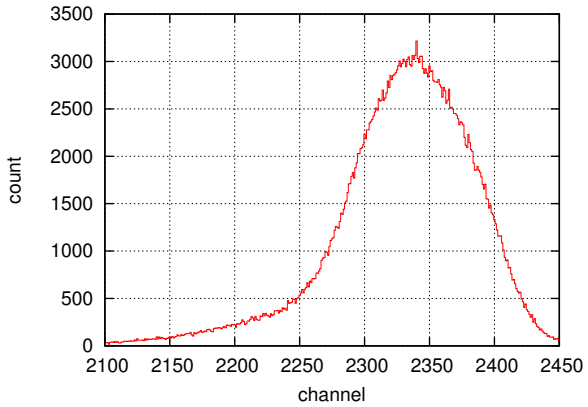


(e) Detektor 3.5

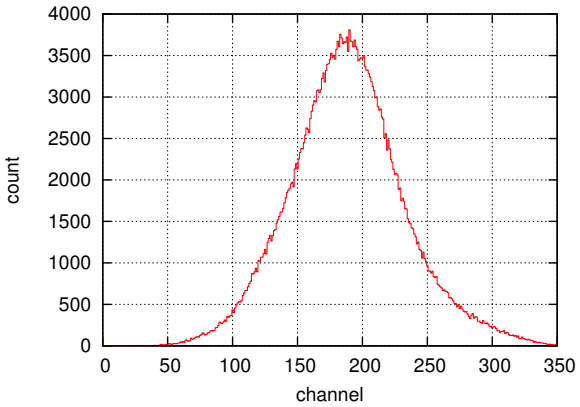


(f) Detektor 3.6

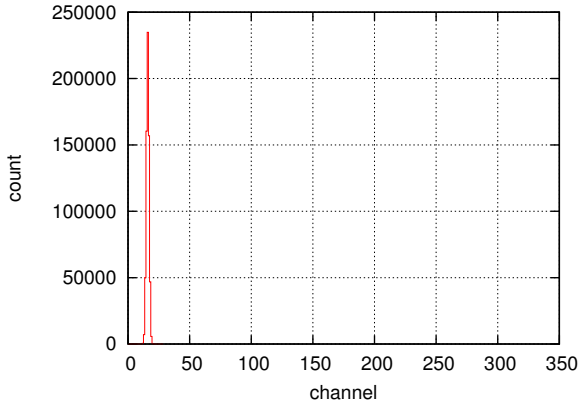
Abbildung 28.: Direkte Bestrahlung von Detektor 3.5 und benachbarten Detektoren. Dauer: $1,2\,\mathrm{h}$. Alle Detektoren außer 1.5-1.11 sind mit Spannung versorgt. Detektoren 1.3, 1.5, 2.2-2.6 und 3.2-3.6 trugen zum Trigger bei. Die anderen Plots befinden sich in Abbildung 27.



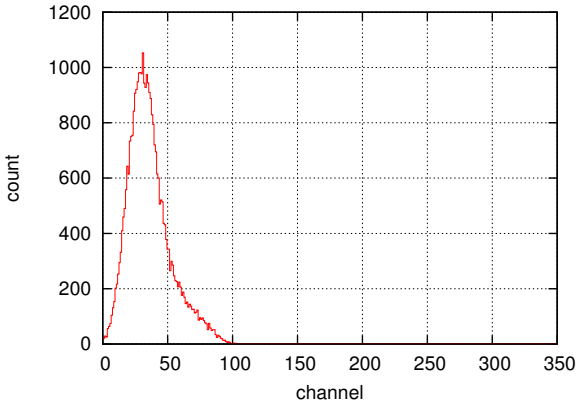
(a) α Peaks.



(b) Detektor 2.3 als Beispiel für spannungslosen Detektor.

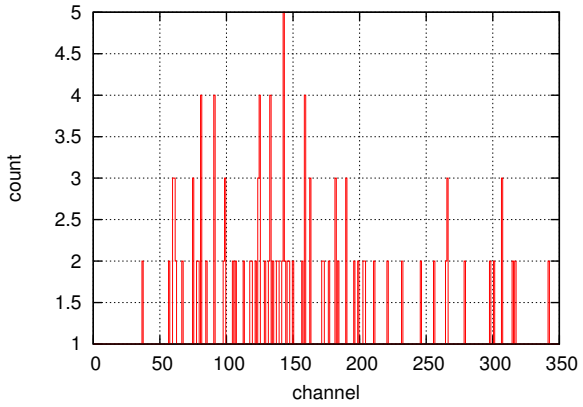


(c) Nicht angeschlossener Detektor.

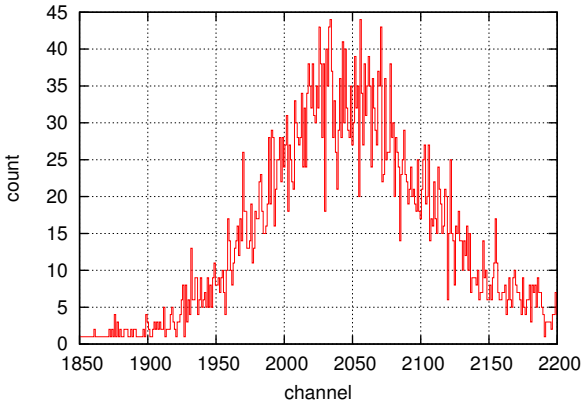


(d) Quelle weggedreht. Trigger aus Pulser mit $1\,\mathrm{kHz}$.

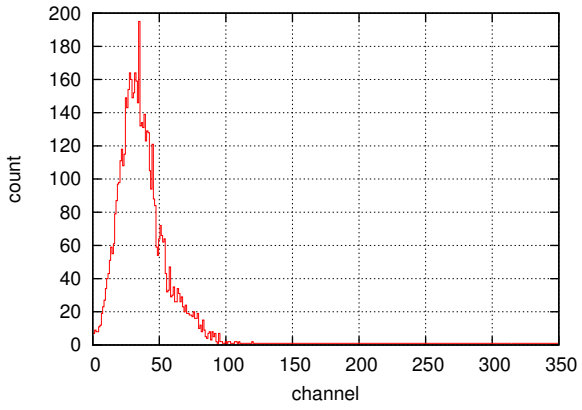
Abbildung 29.: Messungen von Detektor 3.3 zu Ereignissen in niedrigen Kanälen. Dauer: a und b $1\,\mathrm{h}$, c $2\,\mathrm{h}$, d $1\,\mathrm{min}$. Kein Threshold. Nur Detektor 3.3 ist mit Spannung versorgt und verursachte einen Trigger. Die Messung aus Plot a zeigte keine Ereignisse bei niedrigen Kanälen.



(a) Detektor 2.4, niedrige Kanäle

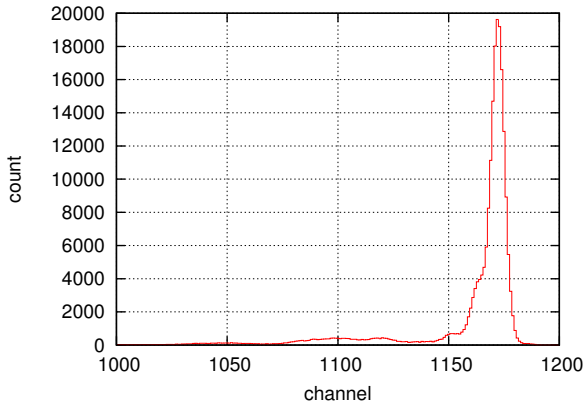


(b) Detektor 2.4, α Peak

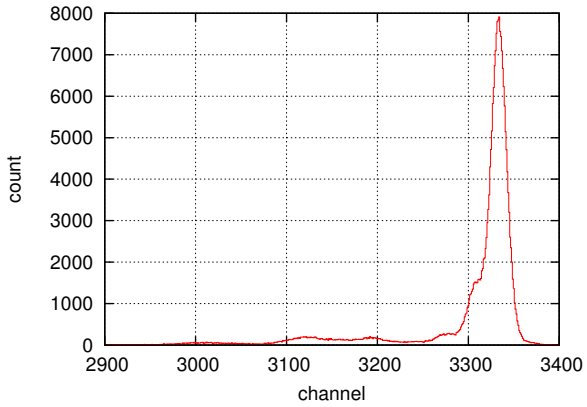


(c) Detektor 3.3

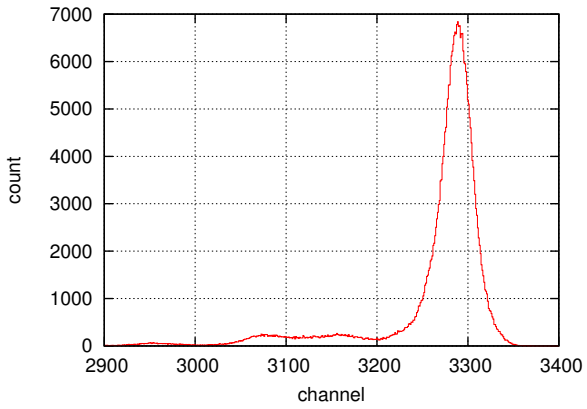
Abbildung 30.: Messungen von Detektor 3.3 zu Ereignissen in niedrigen Kanälen. Dauer: $1\,\mathrm{h}$. Kein Threshold. Nur Detektor 3.3 ist mit Spannung versorgt, der Trigger wurde von Detektor 2.4 ausgelöst.



(a) Detektor 2.5

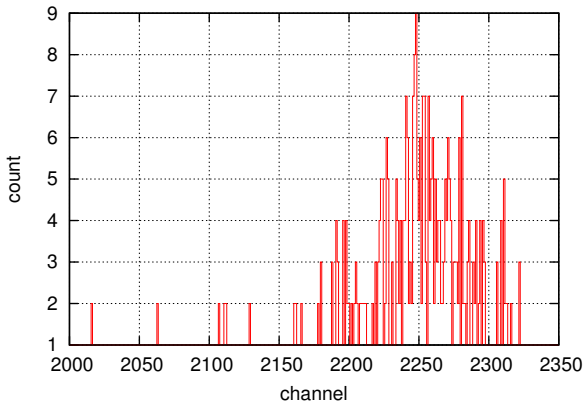


(b) Detektor 2.6

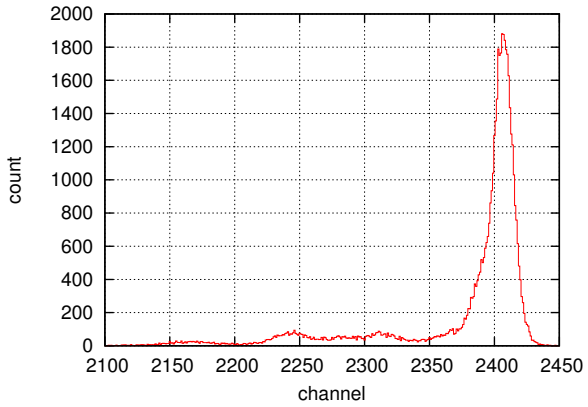


(c) Detektor 3.4

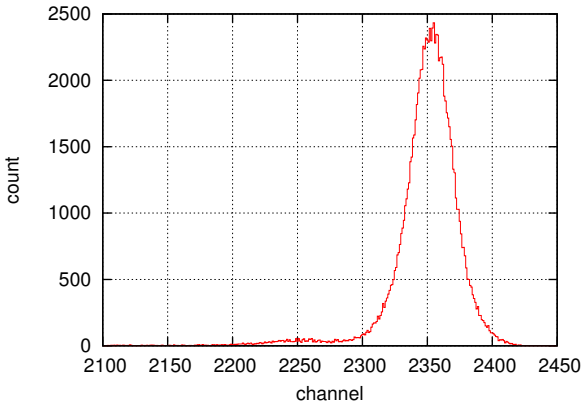
Abbildung 31.: Messungen von Detektoren mit doppelten Peaks bei anderer Verstärkung. Dauer: 1 h. Kein Threshold. Detektoren 2.6 und 3.4 wurden mit Fine Gain 230 und Coarse Gain 5 verstärkt, Detektor 2.5 mit Fine Gain 10 und Coarse Gain 4.



(a) Detektor 2.8

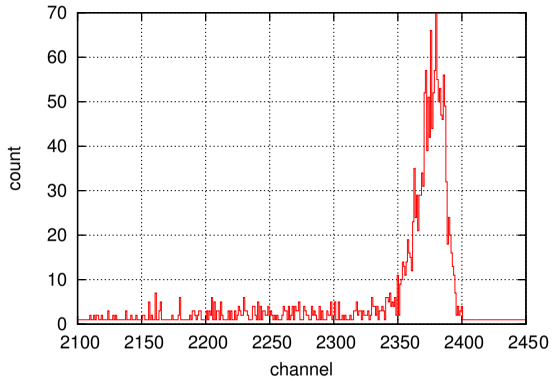


(b) Detektor 3.7

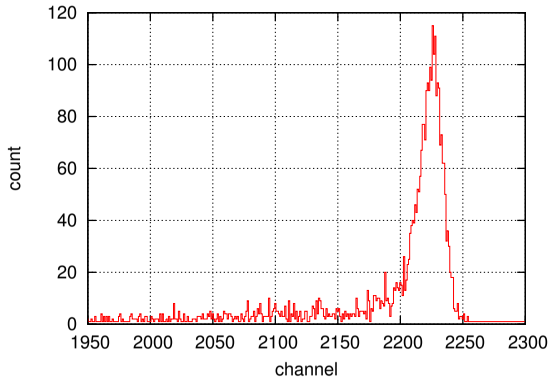


(c) Detektor 3.8

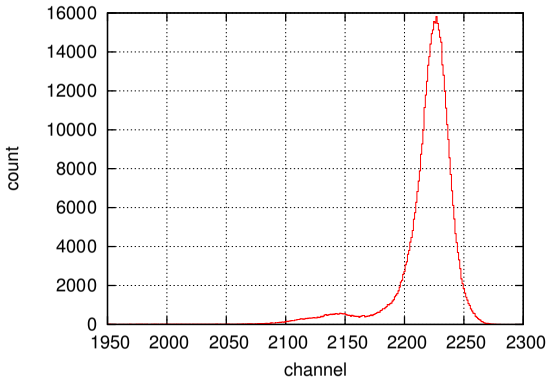
Abbildung 32.: Direkte Bestrahlung von Detektor 3.9 und benachbarten Detektoren. Dauer: $24\,\mathrm{h}$ (MBS bricht Messung nach 1GB ab). Kein Threshold. Alle Detektoren außer 1.5-1.11 sind mit Spannung versorgt. Detektoren 3.6-3.10, 2.6 und 2.7 trugen zum Trigger bei. Die anderen Plots befinden sich in Abbildung 33.



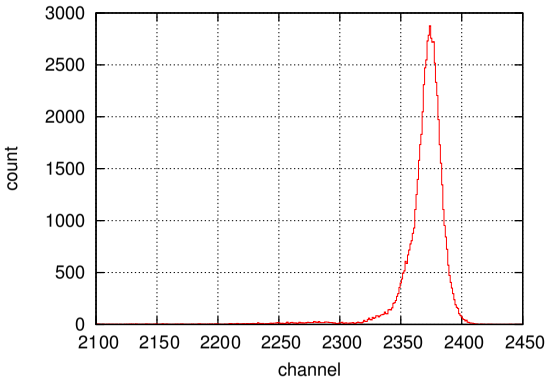
(a) Detektor 2.9



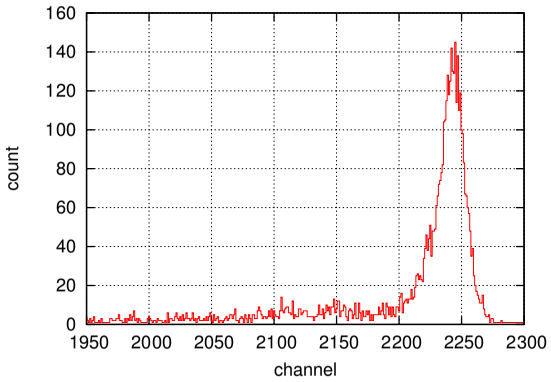
(b) Detektor 2.10



(c) Detektor 3.9

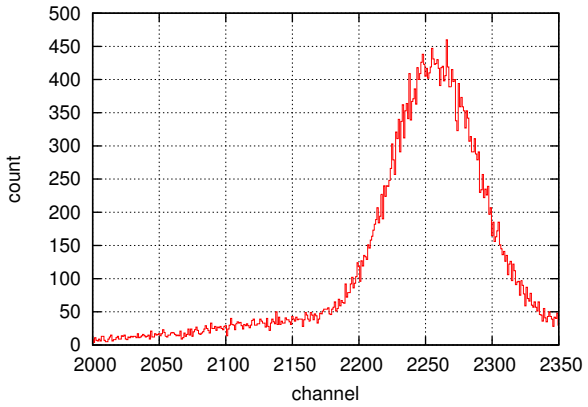


(d) Detektor 3.10

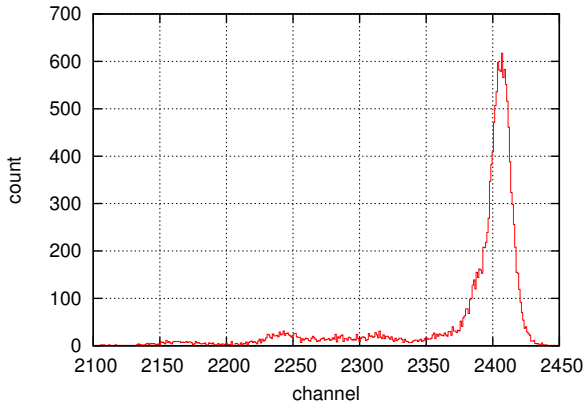


(e) Detektor 3.11

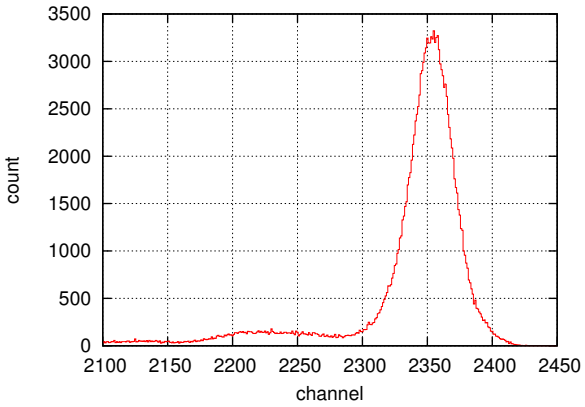
Abbildung 33.: Direkte Bestrahlung von Detektor 3.9 und benachbarten Detektoren. Dauer: $24\,\mathrm{h}$ (MBS bricht Messung nach 1GB ab). Kein Threshold. Alle Detektoren außer 1.5-1.11 sind mit Spannung versorgt. Detektoren 3.6-3.10, 2.6 und 2.7 trugen zum Trigger bei. Die anderen Plots befinden sich in Abbildung 32.



(a) Detektor 2.8

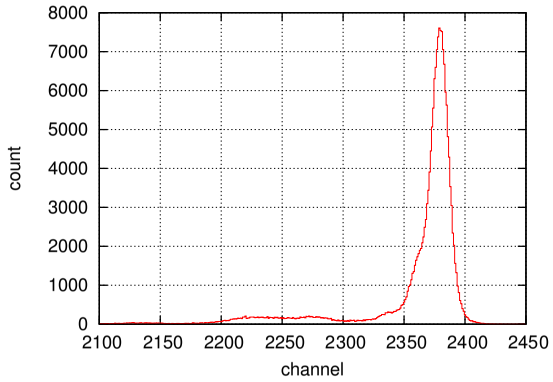


(b) Detektor 3.7

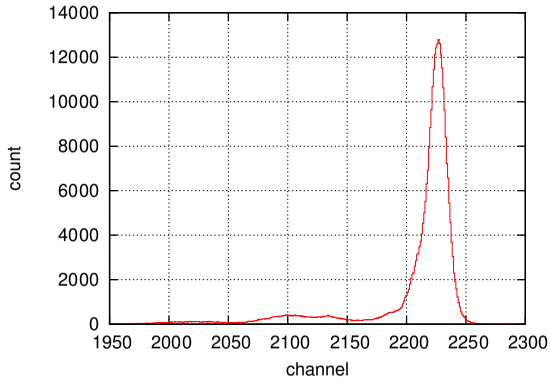


(c) Detektor 3.8

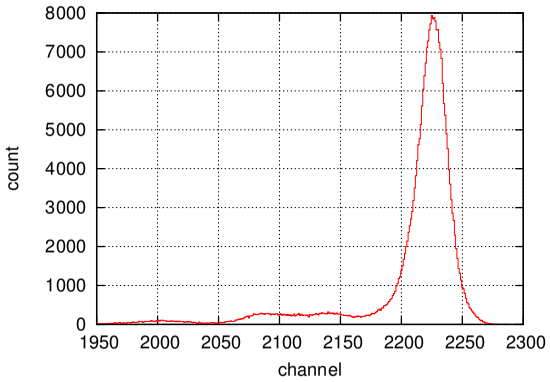
Abbildung 34.: Direkte Bestrahlung von Detektor 3.9 und benachbarten Detektoren. Dauer: $1\,\mathrm{h}$ (MBS bricht Messung nach 1GB ab). Kein Threshold. Alle Detektoren außer 1.5-1.11 sind mit Spannung versorgt. Detektoren 3.6-3.10 und 2.8-2.10 trugen zum Trigger bei. Die anderen Plots befinden sich in Abbildung 35.



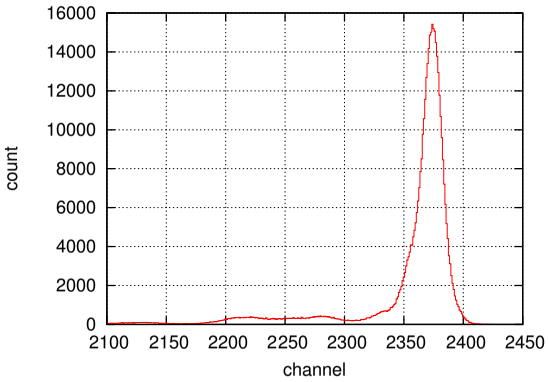
(a) Detektor 2.9



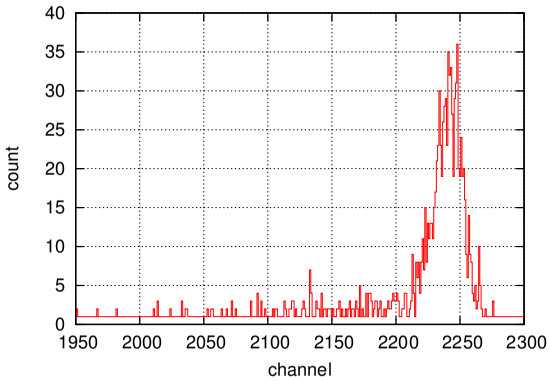
(b) Detektor 2.10



(c) Detektor 3.9



(d) Detektor 3.10



(e) Detektor 3.11

Abbildung 35.: Direkte Bestrahlung von Detektor 3.9 und benachbarten Detektoren. Dauer: 24 h (MBS bricht Messung nach 1GB ab). Kein Threshold. Alle Detektoren außer 1.5-1.11 sind mit Spannung versorgt. Detektoren 3.6-3.10 und 2.8-2.10 trugen zum Trigger bei. Die anderen Plots befinden sich in Abbildung 34.

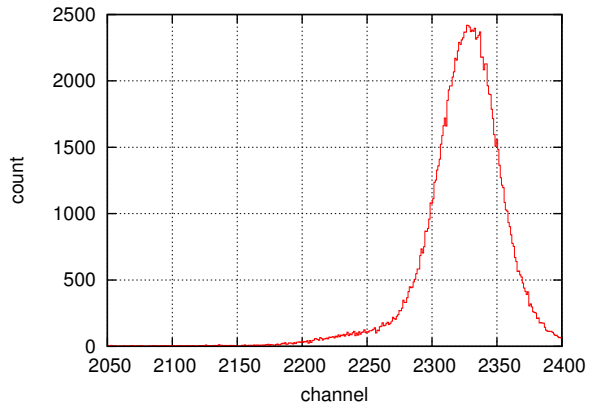


Abbildung 36.: Direkte Bestrahlung von Detektor 3.3. Dauer: 30 min. Kein Threshold. Detektor 3.3 wurde alleine gemessen.

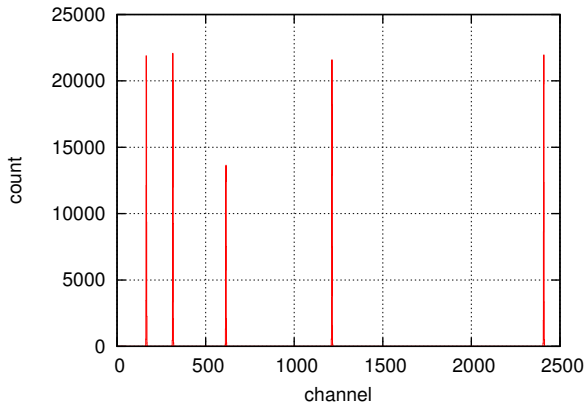
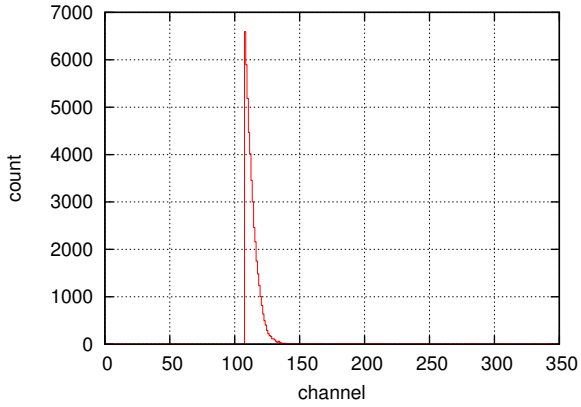
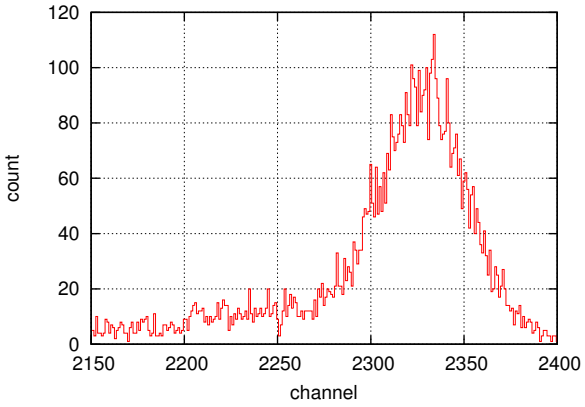


Abbildung 37.: Kalibrierungsmessung mit Signalen aus Pulser angepasst an Signal von Detektor 3.3. Kein Threshold.

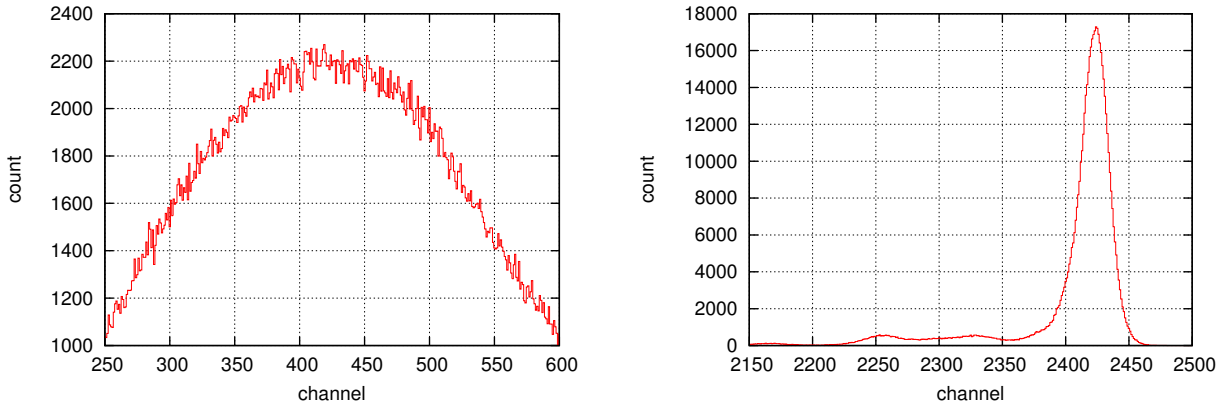


(a) Detektor 3.3, niedrige Kanäle



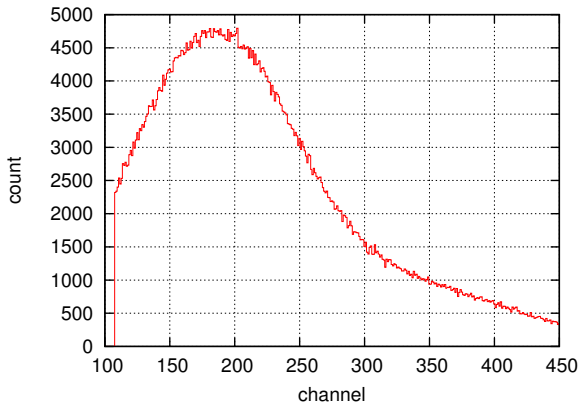
(b) Detektor 3.3, α Peak

Abbildung 38.: Direkte Bestrahlung von Detektor 3.3. Dauer unbekannt, MBS abgestürzt. Die Messung würde über einen Pulser getriggert.

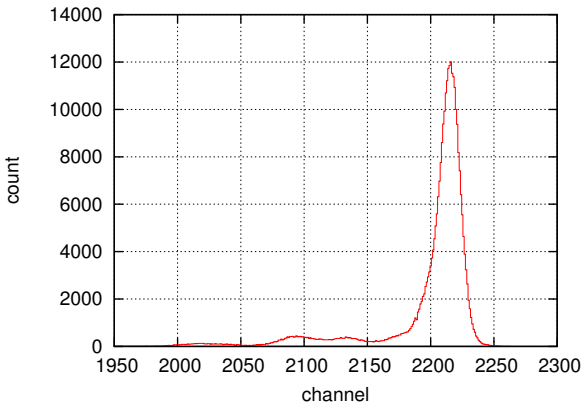


(a) Detektor 3.3

(b) Detektor 3.4

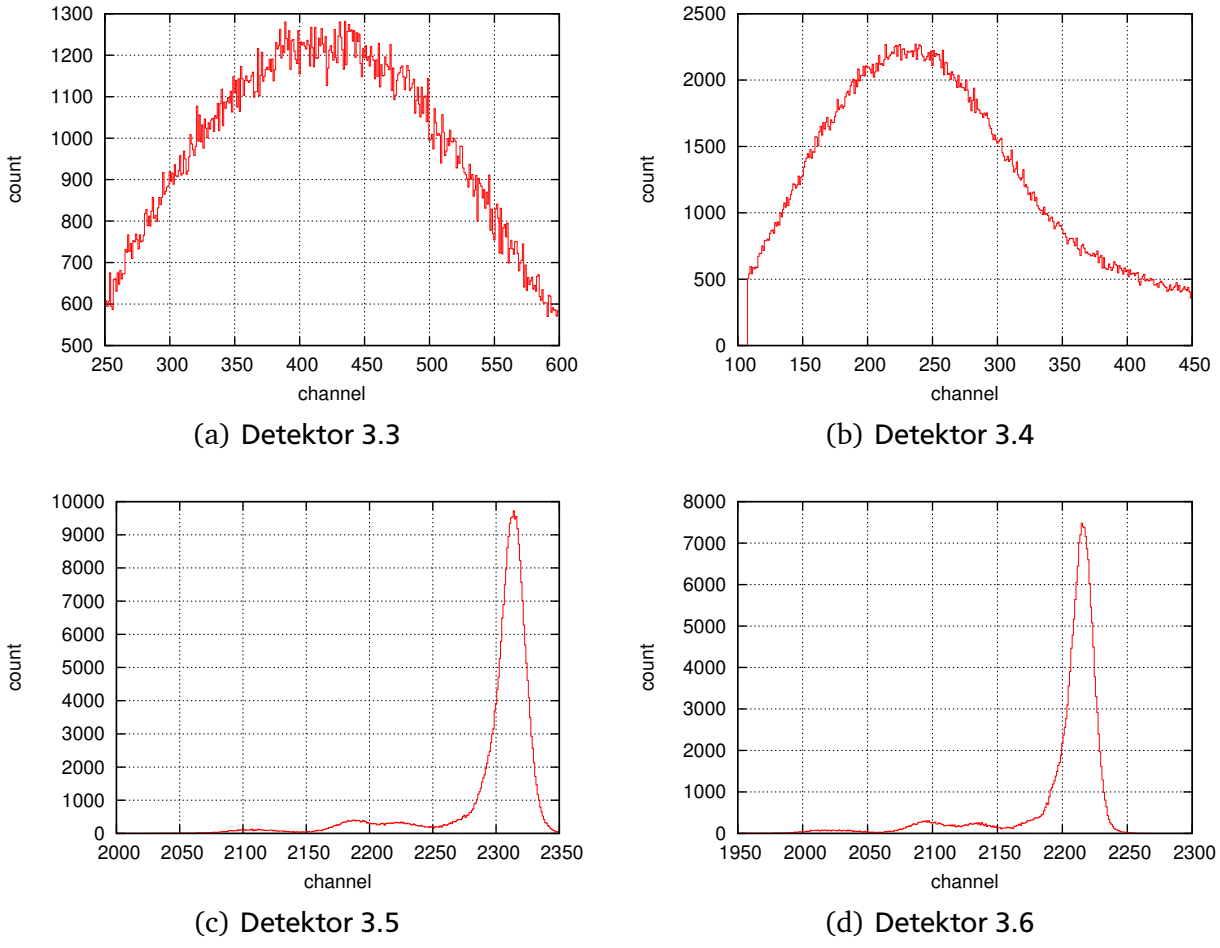


(c) Detektor 3.5



(d) Detektor 3.6

Abbildung 39.: Direkte Bestrahlung von Detektor 3.3 und benachbarter Detektoren. Dauer $1,75\,\mathrm{h}$. Detektoren 3.4 und 3.6 hatten Spannung und erzeugten Trigger.



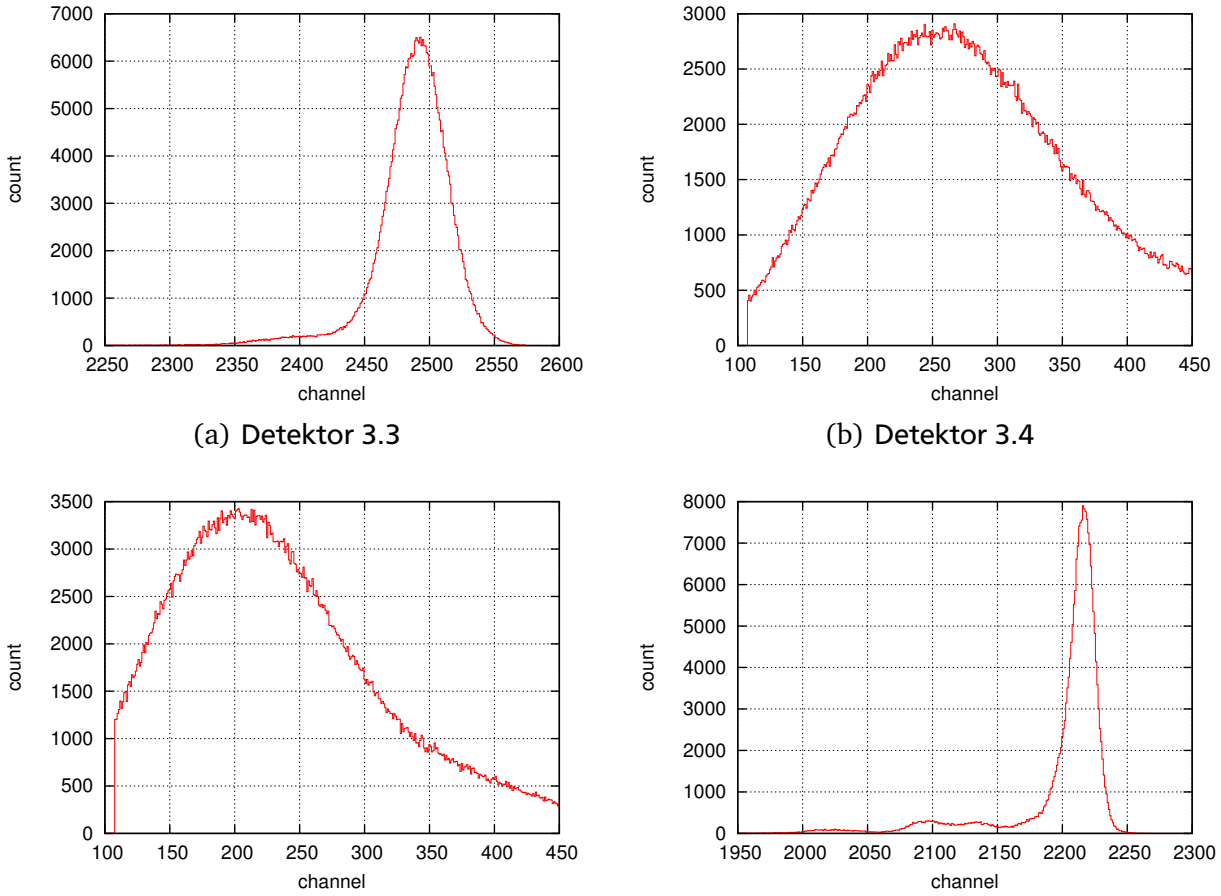
(a) Detektor 3.3

(b) Detektor 3.4

(c) Detektor 3.5

(d) Detektor 3.6

Abbildung 40.: Direkte Bestrahlung von Detektor 3.3 und benachbarter Detektoren. Dauer $1,25\,\mathrm{h}$. Detektoren 3.5 und 3.6 hatten Spannung und erzeugten Trigger.



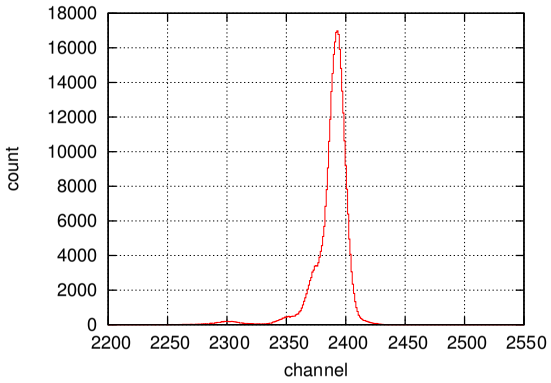
(a) Detektor 3.3

(b) Detektor 3.4

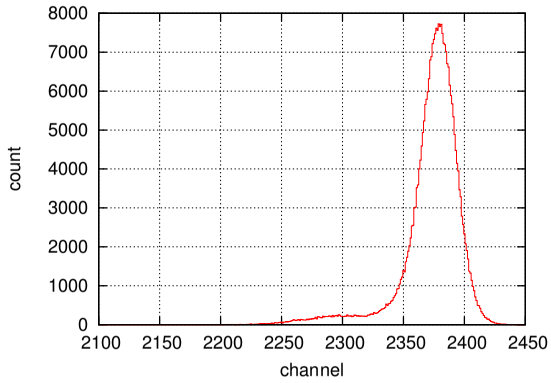
(c) Detektor 3.5

(d) Detektor 3.6

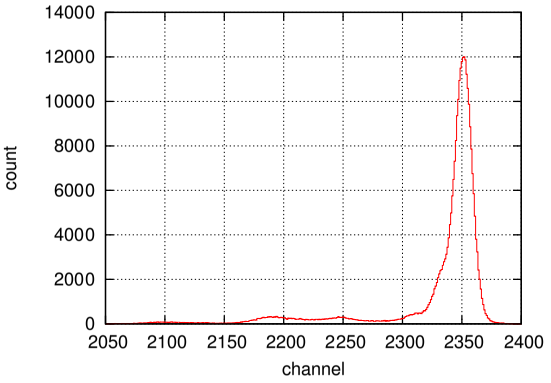
Abbildung 41.: Direkte Bestrahlung von Detektor 3.3 und benachbarter Detektoren. Dauer $1,25\,\mathrm{h}$. Detektoren 3.3 und 3.6 hatten Spannung und erzeugten Trigger.



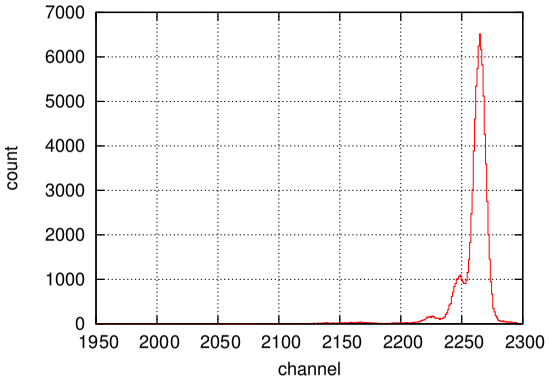
(a) Detektor 1.2



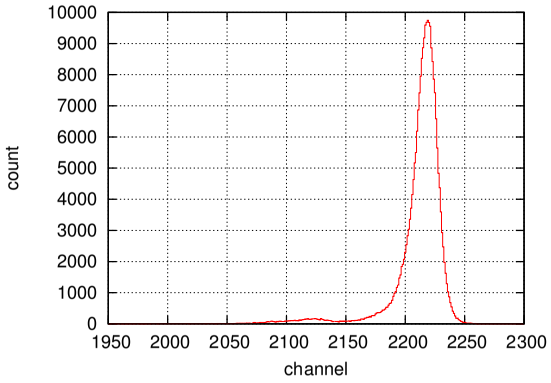
(b) Detektor 1.3



(c) Detektor 2.2

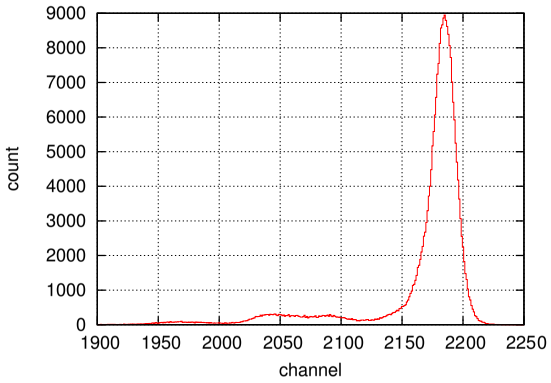


(d) Detektor 2.3

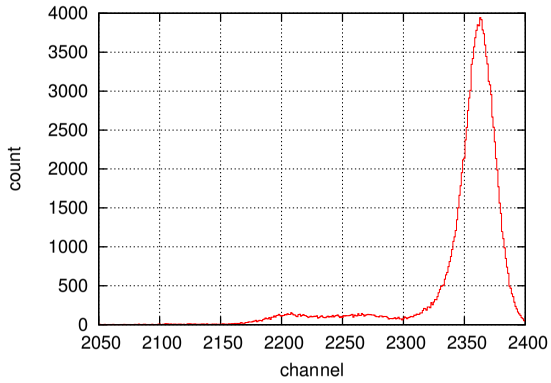


(e) Detektor 2.4

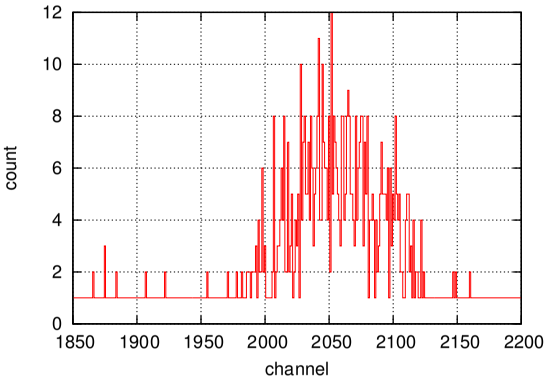
Abbildung 42.: Direkte Bestrahlung von Detektor 1.4 und benachbarten Detektoren. Dauer: 1 h. Alle Detektoren außer 3.5-3.11 sind mit Spannung versorgt. Detektoren 1.2-1.6, 2.2-2.4 und 2.6 trugen zum Trigger bei. Die anderen Plots befinden sich in Abbildung 43.



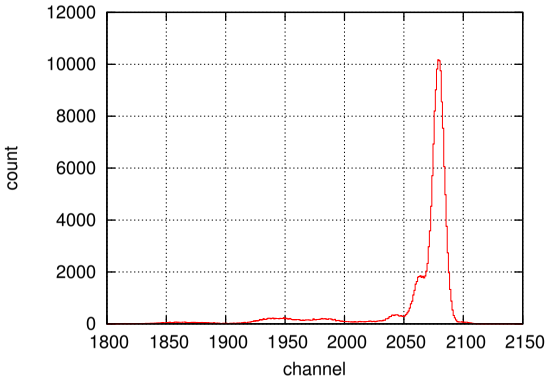
(a) Detektor 1.5



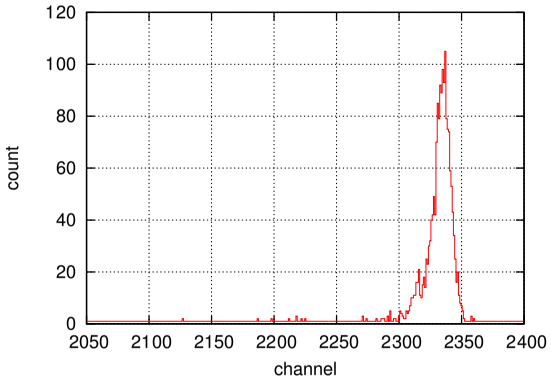
(b) Detektor 1.6



(c) Detektor 2.5

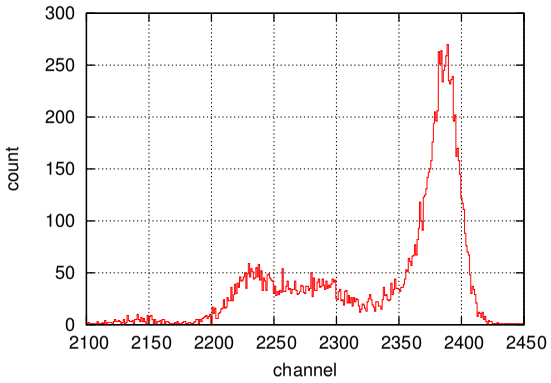


(d) Detektor 2.6

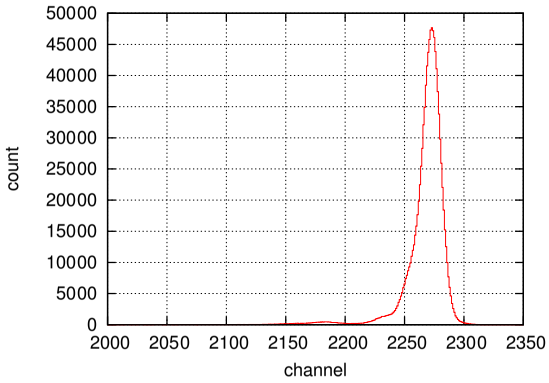


(e) Detektor 2.7

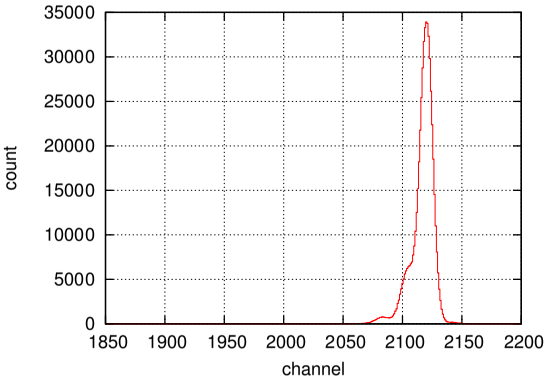
Abbildung 43.: Direkte Bestrahlung von Detektor 1.4 und benachbarten Detektoren. Dauer: 1 h. Alle Detektoren außer 3.5-3.11 sind mit Spannung versorgt. Detektoren 1.2-1.6, 2.2-2.4 und 2.6 trugen zum Trigger bei. Die anderen Plots befinden sich in Abbildung 42.



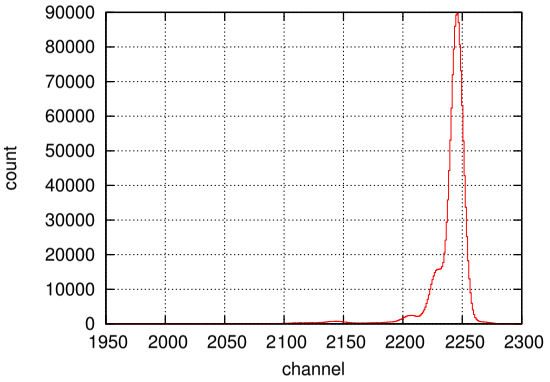
(a) Detektor 1.6



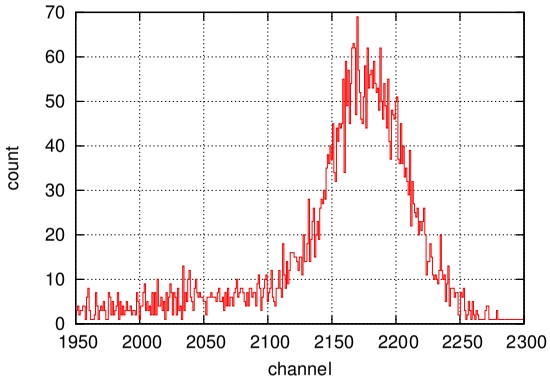
(b) Detektor 1.7



(c) Detektor 2.6

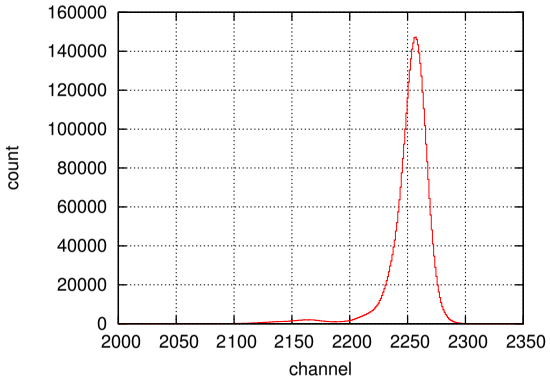


(d) Detektor 2.7

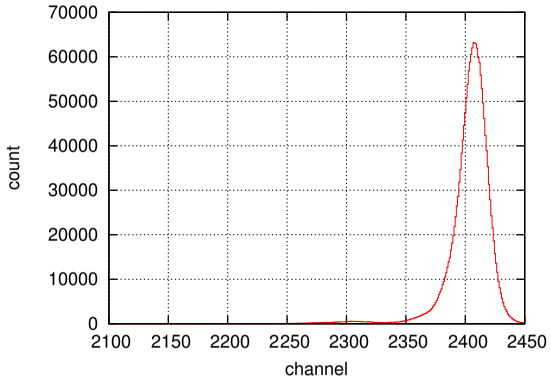


(e) Detektor 2.8

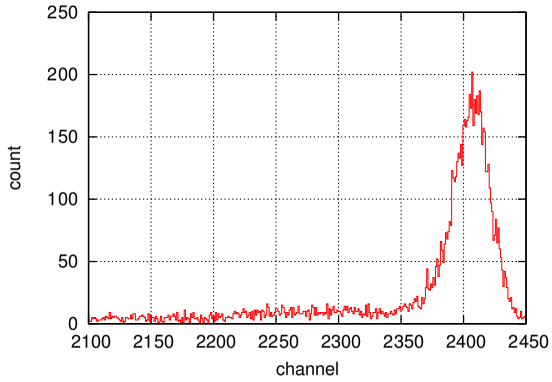
Abbildung 44.: Direkte Bestrahlung von Detektor 1.8 und benachbarten Detektoren. Dauer: $20\,\mathrm{h}$. Alle Detektoren außer 3.5-3.11 sind mit Spannung versorgt. Detektoren 1.6-1.10 und 2.6 trugen zum Trigger bei. Die anderen Plots befinden sich in Abbildung 44.



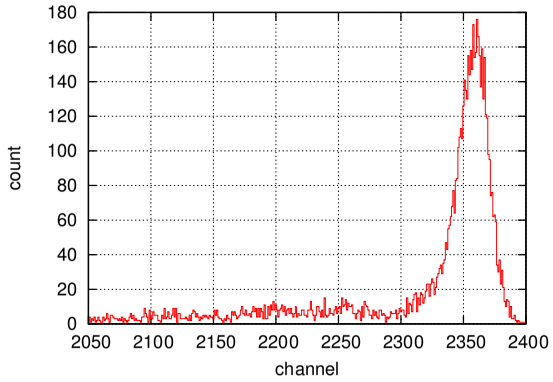
(a) Detektor 1.9



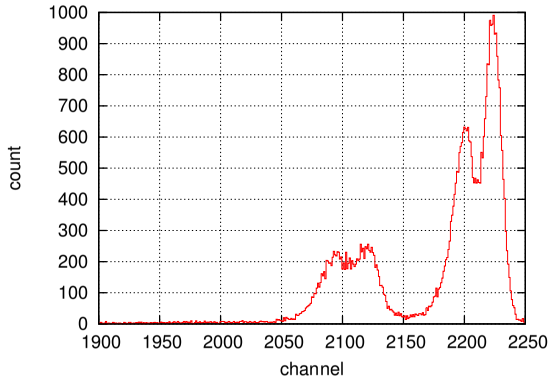
(b) Detektor 1.10



(c) Detektor 1.11



(d) Detektor 2.9



(e) Detektor 2.10

Abbildung 45.: Direkte Bestrahlung von Detektor 1.8 und benachbarten Detektoren. Dauer: $20\,\mathrm{h}$. Alle Detektoren außer 3.5-3.11 sind mit Spannung versorgt. Detektoren 1.6-1.10 und 2.6 trugen zum Trigger bei. Die anderen Plots befinden sich in Abbildung 44.