Trigger- und Energieinformation von Gammaquanten aus digitalisierten Vorverstärkerpulsen

Diplomarbeit
von

Sven Gerhard Dudeck
Physikdepartment, TU München

Dezember 2003

Zusammenfassung

Diese Arbeit beschäftigt sich mit Möglichkeiten der digitalen Signalverarbeitung für Germaniumdetektoren im Rahmen von AGATA, dem europäischen Gamma-Ray Tracking Projekt. Insbesondere sind dies Algorithmen zur Extrahierung der Energieinformation (MWD) und zur Erzeugung von Triggersignalen (SCC und MWDTrigger) aus digitalisierten Vorverstärkerpulsen. Die Arbeit umfasst neben der Implementierung in C++ und VHDL und dem Test dieser Algorithmen auch die Erarbeitung des theoretischen Hintergrundes digitaler Filter und Systeme. Daraus entstanden ist das Konzept des energiesensitiven MWDTriggers, der Einsatz des MWD-Algorithmus zur digitalen Triggergenerierung.

Sowohl in C++ als auch in VHDL wurden Testumgebungen entwickelt, die zum Test dieser Algorithmen eingesetzt wurden, aber auch für eine große Vielfalt anderer digitaler Algorithmen verwendet werden können.

Der digitale MWD-Algorithmus zur Energiebestimmung weist eine mit analogen Systemen gleichwertige Energieauflösung auf (FWHM $\leq$ 1,9 keV bei 662 keV), wobei bei dieser Auflösung theoretisch Zählraten bis zu 120 kHz möglich sind. Der Algorithmus besticht insbesondere durch seine sehr einfache Form und eignet sich ideal zur Implementierung in Hardware.

Der SCC-Triggeralgorithmus ist ebenfalls für die Umsetzung in Hardware konstruiert. Der Algorithmus beruht auf einem statistischen Ansatz zur Erkennung von an- oder absteigenden Flanken eines digitalisierten Signals und ist in der Lage, Pulsabstände unterhalb 1 μs aufzulösen.

Es ist gelungen, die beiden Algorithmen in einen Virtex-II FPGA mit der für AGATA angstrebten Datenbreite von 14 Bit und Taktfrequenz von 100 Mhz zu implementieren (bis zu vier Kanäle parallel auf einem Chip). Trotz fehlender Gleitkommaoperationen erreichen die VHDL-Umsetzungen der Algorithmen die gleiche Genauigkeit wie ihre C++-Pendants. Damit wird es möglich, die Energiebestimmung digital ohne Auflösungsverlust am Experiment vorzunehmen. Die digitale Triggergenerierung erlaubt es, den Datentransfer für nachfolgende Pulsformanalysen auf die wesentlichen Signalanteile zu beschränken.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

1.1 Einleitung

Ziel der modernen Kernspektroskopie ist es, Kerne mit extremen Eigenschaften (z.B. sehr hohem Drehimpuls, Isospin oder Anregungsenergie) zu erzeugen und zu beobachten. Anhand der daraus gewonnenen spektroskopischen Daten erhofft man sich eine Verbesserung bestehender Kernmodelle. Diese Modelle sind zwar in der Lage, Kerne nahe der Stabilität gut zu beschreiben, ihnen mangelt es aber an Vorhersagekraft für Kerne weitab der Stabilität.

Interessant sind solche Nuklide beispielsweise für die nukleare Astrophysik. Der sogenannte r-Prozess (rapid), der in Sternexplosionen stattfinden kann und für die Entstehung eines Großteils der schweren Elemente (schwerer als Eisen) im Universum verantwortlich ist, verläuft im Gebiet neutronenreicher Kerne der Nuklidkarte. Ein tieferes Verständnis dieser Kerne und ihrer nuklearen Reaktionen erlaubt es, Sternexplosionen, die einen r-Prozess aufweisen (z.B. Supernovae Typ-II), nachvollziehen zu können und daraus die Verteilung der Elemente im Universum zu erklären.

Den zu beobachtenden Kernen gemein ist ihre sehr kurze Lebensdauer. Sie existieren natürlicherweise nur unter den extremen Bedingungen explodierender Sterne (sehr hohe Dichte freier Neutronen) und müssen für die Spektroskopie auf der Erde künstlich erzeugt werden. Dies kann auf verschiedene Arten erfolgen:

- Fragmentation: Schwere Kerne werden mit hoher Energie auf ein Target geschossen. Dabei entstehen unter anderem große Fragmente, die einen hohen Neutronenüberschuss aufweisen.

- Spallation: Schwere Kerne (z.B. Uran) werden mit hochenergetischen Protonen oder thermischen Neutronen beschossen und dadurch gespalten.

- Fusion: Wählt man die kinetische Schwerpunktsenergie höher als die Coulombschwelle der Kerne, so kann es zu einer Fusion der beiden Nuklide kommen. Die dabei entstehenden Kerne sind meist neutronenarm.

Die kurze Lebensdauer und geringe Produktionswahrscheinlichkeit dieser Nuklide macht hocheffiziente Detektorsysteme nötig. Die Kerne senden bei der Abregung oder dem Zerfall Photonen im Gammabereich (keV bis MeV) aus. Anhand der Energieverteilung sowie der Polarisierung der Photonen und der zeitlichen Koinzidenz ihres Auftretens lassen sich Rückschlüsse auf die Art des Zerfalls oder der Anregung des Kerns ziehen und damit letztendlich auf seine Struktur. Ein Kernmodell muss diese Strukturen abbilden und erklären können.

1.2 Halbleiterdetektoren

Zur Detektierung der ausgesandten Photonen im Gammabereich kommen Halbleiterdetektoren zum Einsatz.

1.2.1 Funktionsweise

Die Funktionsweise von Halbleiterdetektoren beruht auf dem Prinzip der Ionisationskammer. Einfallende Teilchen wechselwirken mit der Detektormaterie, in diesem Fall dem Halbleiterkristall, und erzeugen dabei freie elektrische Ladungsträger. Drei Prozesse zeichnen sich dafür verantwortlich:

Photoelektrischer Effekt (Abb. 1.1) Das einfallende Photon gibt seine Energie vollständig an ein in einem Atom gebundenes Elektron ab. Ist diese Energie höher als die Ionisationsenergie des Elektrons, so wird es aus dem Atom entfernt und es bleibt ein positiv geladenes Loch in der Elektronenschale zurück. In diesem Fall spricht man vom photoelektrischen Effekt.

Comptonstreuung (Abb. 1.2) Das Photon wird inelastisch an einem Elektron gestreut. Dabei verliert es Energie, welche auf das Elektron übergeht.

Paarbildung (Abb. 1.3) Liegt die Energie des Photons oberhalb von 1022 keV, so kann es spontan ein Elektron-Positron-Paar bilden.

Photoelektrischem Effekt und Paarbildung gemein ist, dass sie auf Grund der Impulserhaltung nur in der Nähe von Atomkernen stattfinden können.

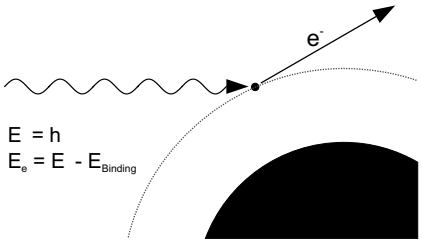


Abbildung 1.1: Photoelektrischer Effekt

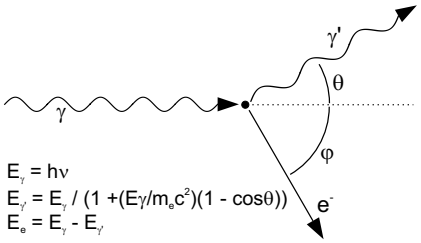


Abbildung 1.2: Comptonstreuung

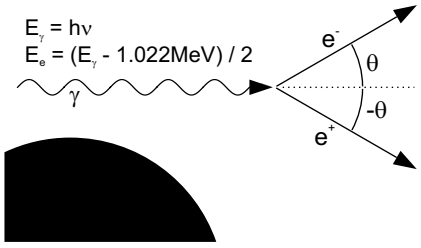


Abbildung 1.3: Paarbildung

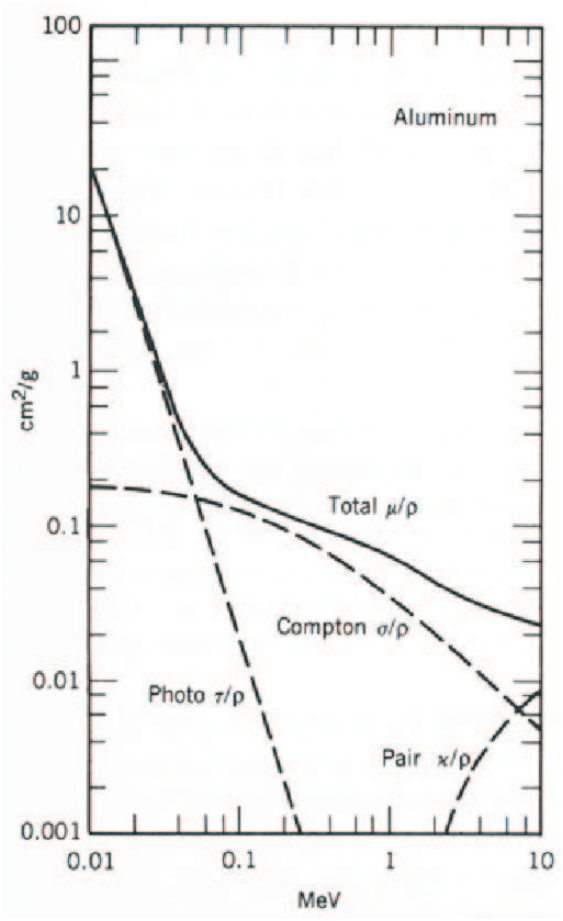


Abbildung 1.4: Wirkungsquerschnitte in Abhängigkeit von der Gammaenergie am Beispiel Aluminium.

Alle drei Prozesse erzeugen freie Elektronen oder Positronen. Die Wirkungsquerschnitte für diese Prozesse hängen stark von der Kernladungszahl und der Energie der Gammaquanten ab (siehe Abbildung 1.4).

Die Elektronen und Positronen treten ebenfalls in Wechselwirkung mit dem Detektormaterial:

Bremsstrahlung (Abb. 1.5) Die freien Ladungsträger werden im elektrischen Feld der Atomkerne abgelenkt und gebremst. Dabei geben sie ihre kinetische Energie in Form von Photonen ab, der sogenannten Bremsstrahlung.

Paarvernichtung (Abb. 1.6) Die Positronen rekombinieren, wenn sie stark

genug abgebremst wurden, mit Elektronen der Atomschalen zu zwei 511 keV Photonen. Auch hier muss augrund der Impulserhaltung ein Atomkern zugegen sein, um Impuls aufzunehmen.

Stossionisation Die freien Elektronen streuen an gebundenen Elektronen wodurch diese aus dem Atom entfernt werden können.

Es entsteht also eine Wechselwirkungskaskade aus Elektronen, Positronen und Photonen, die sich vom Ort der Primärwechselwirkung aus ausbreitet. Die Kaskade endet, wenn die Energie der einzelnen Teilchen nicht mehr ausreicht, weitere Anregungen zu erzeugen. Im Fall eines Halbleiters liegt diese Energie im unteren eV-Bereich.

Energiebänder

Durch die gleichmässige Struktur des Kristalls und den relativ kleinen Abstand der Kristallatome untereinander kommt es zu einem Überlapp der äusseren Atomorbitale. Das Pauliprinzip verbietet jedoch den Elektronen als Fermionen die gleichzeitige Besetzung eines Zustandes. Die Entartung der Orbitalzustände muss also aufgehoben werden, was zu einer Aufspaltung eines jeden Atomorbitals in N Energielevel führt. N ist dabei die Anzahl der Atome im Kristall, also sehr groß. Die Energiedifferenz zwischen diesen N Zuständen ist so gering, dass man von einem Energieband spricht. Dies geschieht mit allen äußeren Orbitalen der Kristallatome, so dass sich aus den diskreten Energiezuständen des Atoms eine Bandstruktur für den gesamten Kristall ergibt. Zwischen den Bändern entstehen "verbotene" Bereiche, die sogenannten Bandlücken.

Hauptsächlich interessant sind die beiden obersten gerade noch bzw. gerade nicht mehr besetzten Bänder. Diese werden als Leitungsband (halb oder gar nicht besetzt für T = 0 K) und Valenzband (oberstes vollständig besetztes Band für T = 0 K) bezeichnet. Aus dieser Bänderstruktur ergibt sich auch die Einordnung eines Kristalls als elektrischer Leiter, Halbleiter oder Isola-

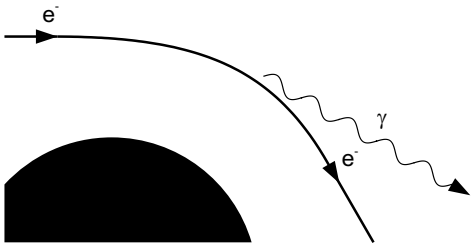


Abbildung 1.5: Bremsstrahlung

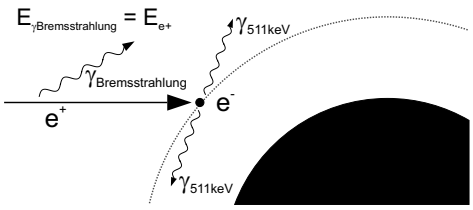


Abbildung 1.6: Paarvernichtung

tor. Entscheidend ist hier die Größe der Bandlücke zwischen Valenz- und Leitungsband. Für Temperaturen oberhalb des absoluten Nullpunktes werden Elektronen durch thermische Anregung (die thermische Energie verteilt sich gemäß der Boltzmannverteilung auf die Elektronen) vom Valenzband ins Leitungsband angehoben. Je größer die Energielücke zwischen diesen Bändern ist, desto weniger Elektronen gelangen aufgrund dieses Mechanismus ins Leitungsband. In Isolatoren ist die Bandlücke so groß, dass bei Raumtemperatur im Leitungsband nicht genügend Ladungsträger für den Stromtransport zur Verfügung stehen, im Leiter liegen Valenz- und Leitungband so nahe beieinander dass sie sich überlappen und es somit viele freie, mit geringem Energieaufwand zu erreichende Zustände gibt.

Im Halbleiter ist die Bandlücke klein genug ($\approx$ 1 eV), dass bei Raumtemperatur Elektronen durch thermische Anregung in ausreichender Anzahl in das Leitungsband angehoben werden um einen geringen Stromfluss zu ermöglichen. Dabei lassen sie im Valenzband positiv geladene Löcher zurück, die durch Nachbarelektronen besetzt werden können, was effektiv zu einer Stromleitung durch diese Löcher führt. Der Halbleiter besitzt also zwei stromtragende Ladungsträger, die negativen Elektronen im Leitungsband und die positiven Löcher im Valenzband.

Ladungswolke
Elektron-Loch-Paare können nicht nur durch thermische Anregung erzeugt werden sondern auch durch Photonen und Stöße mit anderen Teilchen. Die oben beschriebene Wechselwirkungskaskade führt letztendlich dazu, dass sich die durch das primäre Gammaphoton deponierte Energie auf eine große Anzahl von Elektron-Loch-Paaren in einem Umkreis von etwa 1 mm um den primären Wechselwirkungsort verteilt, sowie aufgrund der Impulserhaltung auf Gitterschwingungen (Phononen). Da es sich bei den oben beschriebenen Wechselwirkungen um statistische Prozesse handelt, kann sich die Energie unterschiedlich auf die Bildung von freien Ladungsträgern und der Anregung von Gitterschwingungen verteilen. Im Mittel ist die Anzahl der erzeugten freien Ladungsträger jedoch proportional zur deponierten Energie.

1.2.2 Auslese und Pulsform

Über das Detektorvolumen hinweg wird eine Spannung von einigen kV angelegt, wodurch der Halbleiterdetektor im Prinzip eine große, in Sperrrichtung betriebene Diode darstellt. Detailliertere Informationen zu Halbleiterdetektoren finden sich zum Beispiel in [23].

Durch dieses Potential werden die Elektron-Loch-Paare aufgrund ihrer unterschiedlichen Polarität getrennt und in Richtung der entsprechenden

Elektroden beschleunigt. Der dadurch in den Elektroden induzierte Strom wird von einem ladungssensitiven Vorverstärker (siehe Anhang B) integriert. Abbildung 1.7 zeigt einen solchen Puls.

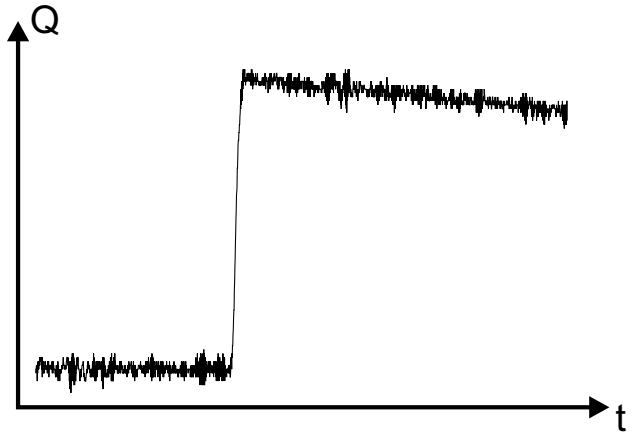


Abbildung 1.7: Beispiel für einen realen Vorverstärkerpuls.

Das dabei entstehende Signal enthält vielerlei Informationen über die Wechselwirkung. Die Amplitude ist proportional zur im Detektor erzeugten freien Ladung und damit zur Energiedeposition durch das Photon. Die Form der ansteigenden Flanke des Vorverstärkerpulses spiegelt die Driftpfade der erzeugten Elektronen und Löcher im Detektor wider und lässt somit Rückschlüsse auf den Produktionsort dieser Ladungsträger zu. Typische Pulslängen liegen im Bereich von 100 bis 500 ns.

1.2.3 Material und Geometrie

Halbleiterdetektoren werden größtenteils aus Silizium oder Germanium hergestellt. Für die Gammaspektroskopie wird heutzutage meist hochreines Germanium verwendet, da es aufgrund seiner höheren Kernladungszahl (Z = 32) einen größeren Wechselwirkungsqurschnitt als Silizium (Z = 14) für Gammaphotonen aufweist.

Es treten hauptsächlich zwei Formen auf:

- Planare Detektoren, bei denen sich die Elektroden auf Ober- und Unterseite des Kristalls befinden ([7], [24] oder [11]), und

- koaxiale bzw. halbkoaxiale Detektoren, die eine Zylindergeometrie aufweisen. Ein Loch durch die Zylinderachse dient als innere Elektrode, die Aussenseite als das Gegenstück. Bei halbkoaxialen Detektoren ist dieses Loch nicht durch den kompletten Kristall gebohrt, so dass der

Detektor auf der einen Seite geschlossen ist (closed-end) (siehe Abbildung 1.8).

Die Größe der Detektoren ist dabei durch die maximale Herstellungsgröße solcher hochreinen Germaniumeinkristalle beschränkt. Diese liegt heute bei einem Durchmesser von etwa 7 cm und einer Länge um die 10 cm.

Hochreine Germaniumdetektoren müssen vor dem Anlegen der Hochspannung auf die Temperatur von flüssigem Stickstoff (77 K) gekühlt werden. Aufgrund der relativ kleinen Bandlücke von Germanium befinden sich ansonsten zu viele thermisch angeregte Elektronen im Leitungsband und es entstehen Leckströme, die den Detektor zerstören würden.

1.3 Gammspektroskopie

Die für die Kernspektroskopie interessanten Messwerte sind die Energie und der Zeitpunkt der Wechselwirkungen der Gammaquanten im Detektor, um daraus Energiespektren und über zeitliche Koinzidenzen Niveauschemata erstellen zu können.

Energie
Die Energiebestimmung erfolgt aus der Amplitude der Vorverstärkerpulse. Diese ist proportional zur Anzahl der an den Elektroden gesammelten freien Ladungsträger, welche wiederum proportional zur im Detektor deponierten Energie sind. Um das sogenannte "ballistic deficit" (siehe Anhang B.2) zu verringern, folgen dem Vorverstärker aufwendige Pulsformschaltungen. Das Maximum dieser Pulse wird mit einem ADC (Analog-to-Digital Converter) digitalisiert und gespeichert. Analoge Schaltung zur Energiebestimmung erreichen Genauigkeiten von bis zu 2‰, limitiert durch elektrisches Rauschen und die intrinsische Auflösung der Detektors.

Trigger/Zeit
Triggergeneratoren beruhen zumeist auf der Leading-Edge- oder Constant-Fraction-Methode (siehe [23]). Diese Techniken haben den Nachteil, dass sie auch auf die Pulsform und -länge sensitiv sind und dadurch das Triggersignal zu unterschiedlichen Zeiten der Pulse generieren. Da großvolumige Halbleiterdetektoren ein breites Spektrum an Pulsformen und -längen produzieren, sind diese Methoden einigen Limitierungen, insbesondere für die genaue Zeitbestimmung, unterworfen. Ein Triggersignal dient unter anderem auch dazu, pile-ups, d.h. für die Energiebestimmung zu nah aufeinander folgende

Pulse, zu erkennen und den zugehörigen Messwert zu verwerfen.

Energie- und Zeitbestimmung stellen konträre Anforderungen an die Pulsform der Signale. Während sie für die Zeitbestimmung und zur Triggererzeugung möglichst schmal gehalten werden soll, um die absolute Differenz der Triggerzeitpunkte aufgrund der unterschiedlichen Pulsformen so gering wie möglich zu halten, ist für die Energiebestimmung eine gute Messung der Pulshöhe erforderlich. Für eine gute Pulshöhenbestimmung bei stark variierenden Pulsformen erweist sich eine lange Pulsformzeit als geeignet, da dann der Pulshöhenverlust durch das ballistic deficit am geringsten ausfällt. Für diese beiden Aufgaben werden daher Pulsformschaltungen mit sehr unterschiedlichen Pulsformzeiten eingesetzt. [15] gibt einen ausführlichen Überblick über die analoge Signalverarbeitung für Halbleiterdetektoren.

Polarisierung
Von weiterem Interesse kann die Polarisierung der ausgesandten Gammaphotonen sein, da sie zusammen mit der Winkelverteilung Rückschlüsse auf die Art (elektrisch oder magnetisch) und den Multipolcharakter der ausgesandten Strahlung erlaubt. Messen kann man die Polarisierung über die Winkelverteilung der über Detektorgrenzen hinweg comptongestreuten Gammaquanten.

1.3.1 Digitale Signalverarbeitung

Die analoge Ausleseelektronik ist insbesondere durch die für die Energiebestimmung benötigten langen Pulsdauern auf eine maximale Zählrate im Detektor von 10 kHz beschränkt. Kernspektroskopieexperimente mit sehr geringer Produktions- oder Anregungswahrscheinlichkeit erfordern aufgrund des extremen Überschusses an unerwünschten Reaktionsprodukten eine sehr hohe Gesamtzählrate im Detektor, um für den beobachteten Reaktionskanal ausreichend Statistik zu erhalten. Einen Ausweg bietet hier die digitale Verarbeitung des anfallenden Datenstromes aus dem Vorverstärker. Digitale Filter sind nicht den gleichen Einschränkungen wie ihre analogen Pendants unterworfen und erlauben es daher, neue Wege in der Datenanalyse zu beschreiten.

1.3.2 Gammaspektrometer

Moderne Gammaspektrometer umgeben die Targetregion mit einer 4π-Kugelschale aus halbkoaxialen hochreinen Germaniumdetektoren (z.B. Euroball und Gammasphere). Anti-Compton-Shields umgeben diese Einzeldetek-

toren (siehe Abbildung 1.8), um nicht vollständig absorbierte Gammaquanten, die durch den Comptoneffekt in den oder aus dem Detektor gestreut wurden, zu verwerfen und so das Peak-to-Total Verhältnis des Spektrums zu erhöhen. Der Einsatz dieser Szintillatoren zur Comptonunterdrückung verringert den durch Germanium abdeckbaren Raumwinkelbereich beträchtlich (auf ca. 50%), wodurch die erreichbare Gesamteffizienz des Detektors limitiert wird.

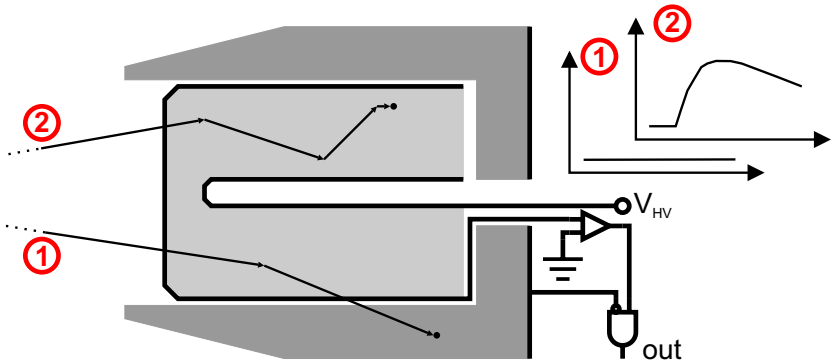


Abbildung 1.8: Halbkoaxialer Detektor mit Anti-Compton-Shield

1.3.3 γ-Tracking und Segmentierung

Eine Möglichkeit, auf die Anti-Compton-Shields zu verzichten, bietet die Technik des γ-Trackings. Sind Ort, Zeit und Energie der einzelnen Wechselwirkungen im Detektor bekannt, so kann der Pfad comptongestreuter Photonen mit Hilfe der Comptonstreuformel auch über Detektorgrenzen hinweg rekonstruiert werden. Damit wird es auch möglich, nicht vollständig im Gesamtdetektor absorbierte Photonen zu erkennen und aus dem Spektrum zu entfernen, ohne auf ein externes Signal wie das des Anti-Compton-Shields angewiesen zu sein. Voraussetzung für eine Rekonstruktion des Streupfades ist eine genaue Kenntnis der Wechselwirkungsorte (auf 1-2 mm) sowie der dabei deponierten Energie.

Die einfachste Art der Ortsbestimmung erlaubt eine Kugelschale aus vielen kleinen Detektoren. Probleme ergeben sich jedoch aus der großen Anzahl der dafür benötigten Detektoren. Wesentlich effizienter ist es, große Germaniumkristalle zu segmentieren, d.h. die äußere Elektrode in mehrere Einzelelektroden zu unterteilen (siehe Abbildung 1.9). Der Ort einer Wechselwirkung lässt sich dann anhand der ansprechenden Elektroden auf einen Bereich in der Größenordnung der Segmente (2 - 3 cm) eingrenzen. Eine genauere Analyse der Pulsformen auf allen ansprechenden Kanälen erlaubt es, den Ort der

Wechselwirkung auf einige Millimeter genau zu bestimmen.

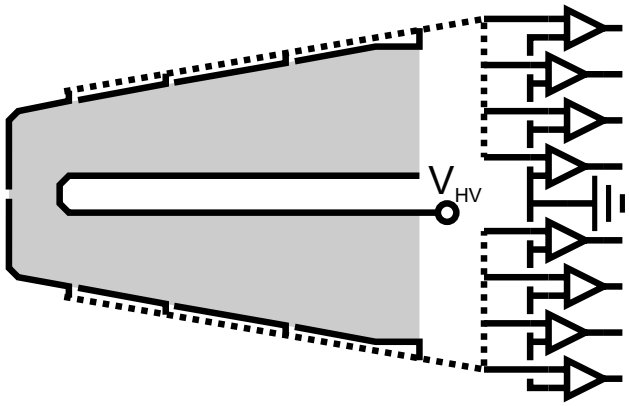


Abbildung 1.9: Halbkoaxialer segmentierter Detektor

Durch die Streupfadrekonstruktion wird es nicht nur möglich, den Comptonuntergrund zu unterdrücken, sondern auch die Polarisierung der einfallenden Photonen zu bestimmen, da diese in die Comptonformel eingeht. Außerdem ermöglicht die Kenntnis des ersten Wechselwirkungspunktes des Gammaquants im Detektor eine Korrektur der Dopplerverschiebung für Gammaquanten, die bei hohen Geschwindigkeiten emittiert werden.

Transiente Signale
Durch die Segmentierung treten neben den Nettoladungssignalen zusätzliche bipolare Stromsignale auf den Außenkontakten auf, die sogenannten transienten Signale. Die durch die Hochspannung beschleunigten Elektronen und Löcher induzieren nicht nur auf ihren Zielelektroden Ladungen, sondern auch in den benachbarten Segmenten. Effektiv wird dort jedoch keine Ladung gesammelt, so dass die induzierten Ladungen wieder abfließen müssen. Transiente Signale äußern sich im Vorverstärkersignal als Peaks im Gegensatz zu Nettoladungssignalen, die eine Stufe auf dem Signal erzeugen (siehe auch [21]).

1.3.4 AGATA

Die oben beschriebenen Techniken zur Ortsbestimmung mittels Pulsformanalyse befinden sich zur Zeit noch in der Entwicklung. Der AGATA-Detektor soll neben GRETA[1] das erste Gammaspektrometer sein, welches ausgiebigen Gebrauch von diesen Technologie macht.

[1] **G**amma **R**ay **E**nergy **T**racking **A**rray [1]

Das AGATA-Projekt [14] (**A**dvanced **GA**mma-ray **T**racking **A**rray) ist eine europäische Kollaboration von etwa 40 Instituten. Ziel ist es, ein hocheffizientes und -sensitives 4π-Gammaspektrometer zu entwickeln und zu bauen. Der AGATA-Detektor wird aus 120 oder 180 hochreinen halbkoaxialen Germaniumkristallen bestehen. Diese werden 36-fach segmentiert sein und eine vollständig digitale Ausleseelektronik besitzen.

Es soll damit eine Effizienz von 40% und ein Peak-to-Total Verhältnis von 60% bei Energien um die 1,33 MeV erreicht werden. Der Detektor wird in der Lage sein, Ereignisraten von 50 kHz und mehr pro Kristall zu verarbeiten. Zur Dopplerkorrektur soll die Winkelauflösung unterhalb von 1° liegen und die angestrebte Energieauflösung bei 5 keV für 1 MeV und einer Geschwindigkeit der emittierenden Kerns von 50% Lichtgeschwindigkeit.

1.4 Ziel der Diplomarbeit

Die vollständig digitale Ausleseelektronik von AGATA erfordert die Entwicklung und den Test neuer Algorithmen und Filter zur Datenanalyse. Seit Beginn der 90 Jahre beschäftigt sich die Kernspektroskopie mit diesen Techniken (z.B. [19], [13]), im großen Maßstab zum Einsatz gekommen ist sie bisher im Miniball-Detektor, dessen sechs- bzw. zwölffach segmentierten Germaniumdetektoren digital ausgelesen werden ([8], [22]).

Bei 14 Bit Auflösung und 100 MHz Taktfrequenz produziert jeder AGATA-ADC einen Datenstrom von knapp 200 MB/s. Zieht man in Betracht, dass jeder Detektor 37 Kanäle besitzt und es davon wiederum 120 oder 180 Stück geben soll, so sieht man, dass eine Datenreduktion unumgänglich ist. Diese soll mit Hilfe von DSPs oder FPGAs direkt nach der Digitalisierung erfolgen, zum Einen durch die Bestimmung der Energie mittels des MWD-Algorithmus, zum Anderen durch eine Selektion der relevanten Datenabschnitte mittels eines digitalen Triggeralgorithmus. Ziel der Arbeit war es daher, diese Algorithmen (MWD für die Energiebstimmung und SCC zur Triggerung) in VHDL für einen FPGA zu programmieren und die Leistungsfähigkeit dieser Hardwareimplementationen zu testen. Zu Vergleichs- und Testzwecken sollte dies ebenso in C bzw. C++ geschehen.

Kapitel 2

Algorithmen

2.1 MWD

Der *Moving Window Deconvolution*-Algorithmus (MWD) [13] von A. Georgiev und W. Gast ist ein digitaler FIR-Filter[1] zur Rekonstruktion der von einer Wechselwirkung im Detektor deponierten Energie. Wie in Kapitel 1.2 beschrieben, ist diese Energie proportional zur Anzahl der durch die Wechselwirkung erzeugten freien Ladungsträger. Der Algorithmus beruht auf der Tatsache, dass die Impulsantwort eines ladungssensitiven Vorverstärkers durch den charakteristischen exponentiellen Abfall dominiert wird. Der Algorithmus besteht ausschließlich aus Additionen und einer Multiplikation mit einem konstanten Faktor pro Takt und eignet sich daher vorzüglich zur Implementierung in einem FPGA- oder DSP-Chip.

2.1.1 Funktionsweise

Der Kernalgorithmus setzt sich aus zwei logischen Schritten zusammen,

- der Entfaltung des Vorverstärkersignals, bestehend aus Detektorsignal und Impulsantwort des Vorverstärkers (*Deconvolution*), sowie
- der Integration des dadurch zurückgewonnenen Detektorsignales (Stromsignal) über einen bestimmten sich zeitlich bewegenden Bereich (*Moving Window*).

Im Anschluss folgen ein Mittelwertfilter zur Rauschunterdrückung und ein Maximumdetektor, der letztendlich das Ergebnis liefert. Abbildung 2.1 veran-

[1] finite impulse response filter, siehe Anhang A für weiterführende Informationen zu digitalen Filtern.

schaulicht dies anhand der Ausgangssignale der jeweiligen Schritte für einen Rechteckimpuls.

Der Kernalgorithmus wird beschrieben durch die Differenzengleichung

$$y[n] = x[n] + (1-k) \cdot \sum_{m=1}^{M} x[n-m] - x[n-M] \tag{2.1}$$

mit $k = \exp(-\alpha)$, $\alpha > 0$, $\alpha = (\tau \cdot f_{sampling})^{-1}$. $x[n]$ ist das digitalisierte Ausgangssignal des Vorverstärkers, $y[n]$ das Ausgangssignal des MWD-Algorithmus. $M \in N$ ist die Breite des Integrationsbereiches, τ die Zeitkonstante (Integrationszeit) des Vorverstärkers und $f_{sampling}$ die Abtastfrequenz des ADCs. Eine Herleitung dieser Differenzengleichung findet sich in Anhang C.1.

Erläuterungen
Der zweite Schritt des Kernalgorithmus stellt im Prinzip nichts anderes dar als die digitale Nachbildung eines idealen ladungssensitiven Vorverstärkers. Auch hier wird aus einem Strompuls des Detektors ein Ladungssignal erzeugt. Die digitale Integration enstpricht dem Sammelkondensator, die endliche digitale Wortbreite dessen Kapazität in Verbindung mit der Versorgungsspannung des Vorverstärkers. Das Ladungssignal wird also auch bei der digitalen Integration irgendwann in Sättigung gehen. Um dies zu verhindern, findet der Entladewiderstand parallel zum Kondensator ebenfalls sein digitales Pendant - die Integration wird auf einen Bereich (die letzten M Werte) beschränkt. Dieser Bereich muss größer sein als die Länge eines Strompulses, um die gesamte Ladung innerhalb des Pulses zu erfassen, sollte aber idealerweise kleiner als der Abstand zweier aufeinander folgender Pulse sein. Mehr zu den Parametern des Algorithmus in Abschnitt 2.1.2.

Der Vorteil dieser digitalen Nachbildung liegt darin, dass die "Entladung" zwar immer noch kontinuierlich und nicht eventweise stattfindet, aber nur "Ladungen" aus dem Signal entfernt werden, deren Information schon vollständig verwertet wurde. Dadurch vermeidet man das *Ballistic Deficit* (siehe Anhang B.2), welches die Energieauflösung des Systems auf bis zu einige Prozent verschlechtern kann, sowie die Bildung des exponentiell abfallenden Schwanzes, welcher zu pile-ups und damit ebenfalls zu Fehlern in der Energiemessung führt.

Prinzipiell ließen sich die beiden Schritte auch einzeln durchführen. Die dafür benötigten digitalen Filter hätten die Differenzengleichungen

$$v[n] = x[n] - k \cdot x[n-1]$$

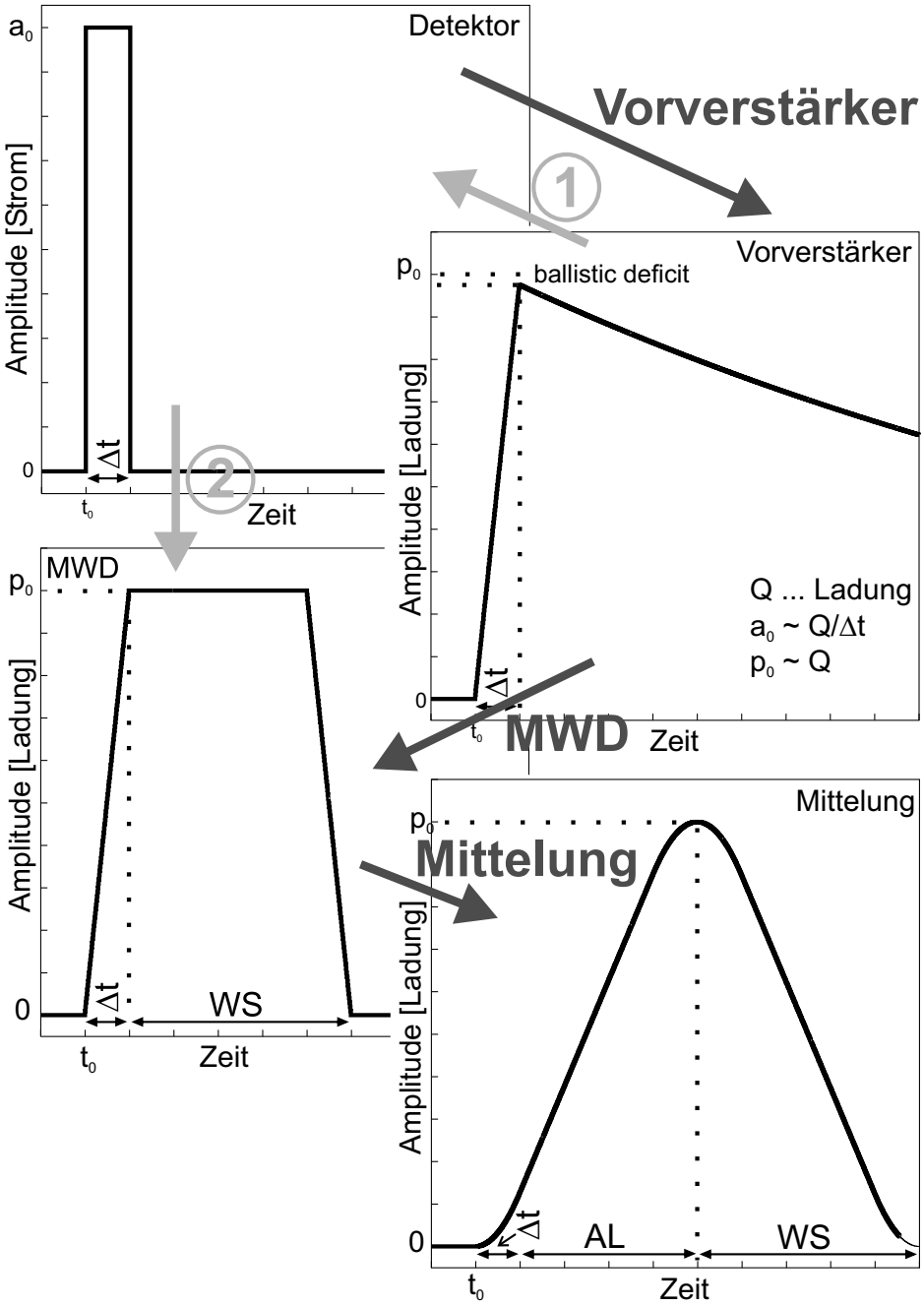


Abbildung 2.1: Ausgangssignale der verschiedenen MWD-Schritte am Beispiel eines Rechteckimpulses der Länge Δt. Das Eingangssignal des MWD-Algorithmus ist das Vorverstärkersignal. WS entspricht der Länge des verwendeten Fensters, AL der Länge des Mittelungsbereiches. 1 und 2 stellen die im Text beschriebene Entfaltung des Vorverstärkersignals sowie die Summation über das Fenster dar. Das Zustandekommen der Pulsformen wird im Abschnitt Pulsform näher erklärt.

für die Entfaltung und

$$y[n] = v[n] - v[n-M] + y[n-1]$$

für die anschließende Summation über das Fenster der Breite M.

Der Grund für die Zusammenfassung liegt in der Implementierung in Hardware. Da der FPGA keine Fließkommaberechnungen zulässt, muss k durch einen Integerwert repräsentiert werden. Dabei ist

$$\begin{aligned} k &= \exp(-\alpha) && \text{mit } \alpha = (\tau f_{sampling})^{-1} \approx 10^{-4} \\ &\approx 1 - (1 \cdot 10^{-4}) \end{aligned}$$

Um k auf 10^{-6} genau abzubilden, was in etwa einem Fehler von 1% für α entspricht, muss dessen Integerrepräsentation eine Unterteilung des Intervalls $[0, 1)$ im Bereich von 10^6 ermöglichen, also eine Datenwortbreite von 20 Bit ($2^{20} \approx 10^6$) und mehr besitzen. Dies erschwert zum einen die Multiplikation von k mit $x[n-1]$, zum anderen ist das Ausgangssignal der Entfaltungsstufe dadurch ein sehr großes Datenwort (Datenwortbreite x + Datenwortbreite k). Die anschließende Summation dieser großen Datenworte über das Fenster benötigt unnötigerweise große Resourcen des Chips.

Die Zusammenfassung von Entfaltung und Summation zu Gleichung 2.1

$$y[n] = x[n] + (1-k) \cdot \sum_{m=1}^{M} x[n-m] - x[n-M]$$

erlaubt es, die benötigten Wortbreiten innerhalb des Algorithmus minimal zu halten. $(1-k)$ bewegt sich in der Größenordnung von 10^{-4} und lässt sich daher durch die untersten 8 Bit der obigen 20 Bit Repräsentation des Intervalls $[0, 1)$ in ausreichender Genauigkeit abbilden. Dies beschleunigt die Multiplikation und erleichtert die Bildung der Summe. Für weitere Erklärungen siehe Abschnitt 4.1.2.

Die gerade beschriebene Auftrennung in zwei Einzelschritte macht es allerdings möglich, mehrere Entfaltungsstufen vor der Summation einzusetzen, um der Komplexität des Vorverstärkers besser gerecht zu werden.

Pulsform

Wie aus Abbildungen 2.2 und 2.3 ersichtlich, hat das Ausgangssignal des MWD-Filters annähernd Trapezform. Dabei sind ansteigende und abfallende Flanke punktsymmetrisch zueinander, und ihre Form spiegelt den Verlauf des Stromsignales wieder. Die Breite der Flanken entspricht der Strompulslänge Δt, die Länge des gesamten MWD-Pulses ist $(M \cdot T_{sampling}) + \Delta t$. Interessant

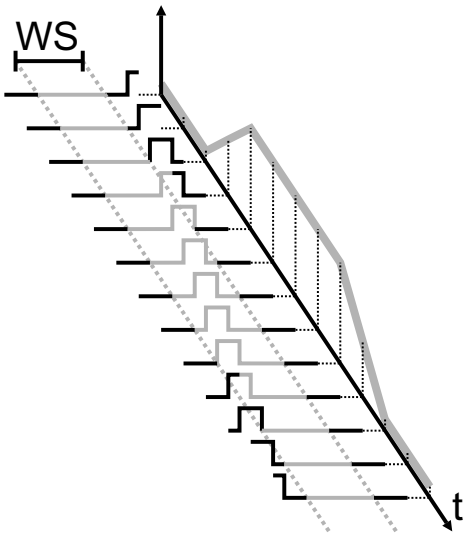


Abbildung 2.2: Die Summation über die jeweils letzten M Werte (WS = $M \cdot T_{sampling}$) führt zu einem trapezförmigen Ausgangssignal.

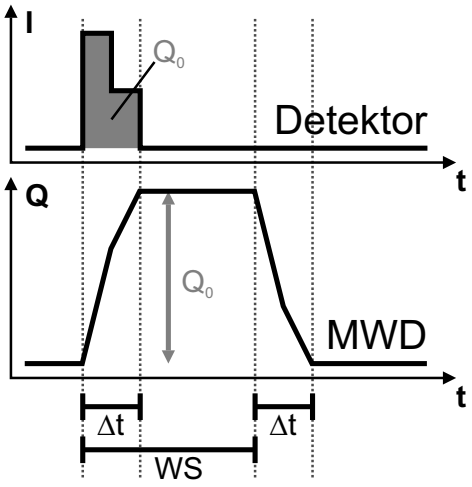


Abbildung 2.3: Schematische Darstellung der Ausgangspulsform des MWD-Filters.

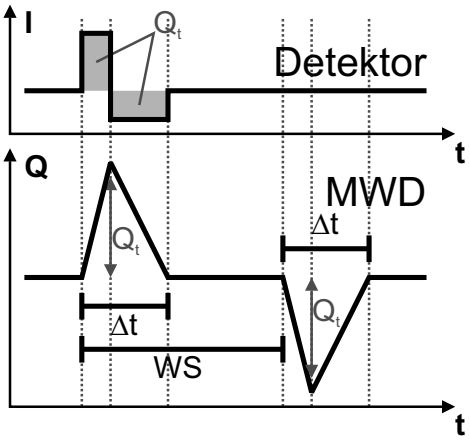


Abbildung 2.4: Schematische Darstellung der Ausgangspulsform des MWD-Filters für ein transientes Signal.

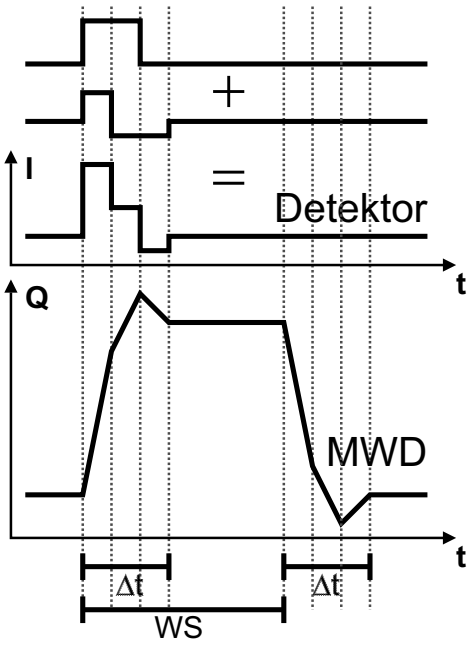


Abbildung 2.5: Schematische Darstellung eines Nettoladungssignales überlagert von einem transienten Signal unterschiedlicher Länge.

für die Energiebestimmung ist die Höhe des Trapezes, welche die gesammelte Ladung widerspiegelt. Da die Länge des Strompulses nur einen Teil des Integrationsbereichs ausfüllt, repräsentiert jeder Wert des Plateaus die gesamte Ladung innerhalb des Pulses sowie das über diesen Bereich integrierte Rauschen auf dem Signal.

Transiente Signale erzeugen im MWD-Algorithmus einen bipolaren Puls (siehe Abbildung 2.4). Die Länge ist auch hier $(M \cdot T_{sampling}) + \Delta t$ für die gesamte Struktur. Eine Überlagerung von transientem und Nettoladungssignal (Abbildung 2.5) führt, da der MWD-Algorithmus ein linearer Filter[2] ist, zu einem Ausgangssignal welches eine Überlagerung der Ausgangssignale der Einzelpulse darstellt.

Mittelwertbildung zur Rauschunterdrückung

Wie man anhand Abbildung 2.6 erkennen kann, beinhaltet jeder Wert des Plateaus neben der Ladung auch das über das jeweilige Fenster integrierte Rauschen. Eine Mittelung über das Plateau verringert somit den statistischen Rauschanteil in der Maximumbestimmung. Lässt man diese Mittelung kontinuierlich über das gesamte Signal laufen, so liefert das Maximum der dabei entstehenden Kurve die gemittelte Höhe des Trapezplateaus. Die Differenzengleichung für solch einen Filter lautet

$$z[n] = \sum_{l=0}^{L-1} w[l] y[n-l]$$

L ist die Mittelungslänge, $w[l]$ (die Impulsantwort dieses Filters) eine Gewichtungsfunktion mit der Bedingung $\sum_{l=0}^{L-1} w[l] = 1$. $w[l]$ ist je nach Art des zu säubernden Signales zu wählen. Die einfachste Art der Rauschunterdrückung stellt das arithmetische Mittel dar. Hierbei ist $w[l] = \frac{1}{L}$. Es werden also die letzten L Ausgangswerte des MWD-Kernalgorithmus aufsummiert und durch L geteilt:

$$z[n] = \frac{1}{L} \cdot \sum_{l=0}^{L-1} y[n-l] \tag{2.2}$$

2.1.2 Parameter

Der Algorithmus besitzt zwei Parameter, über die sich die Qualität des Ausgangssignales steuern lässt. Dies sind die Länge des Summationsbereiches

[2] FIR-Filter sind eine Untergruppe der LTI-Filter (linear time-invariant). Siehe Anhang A.

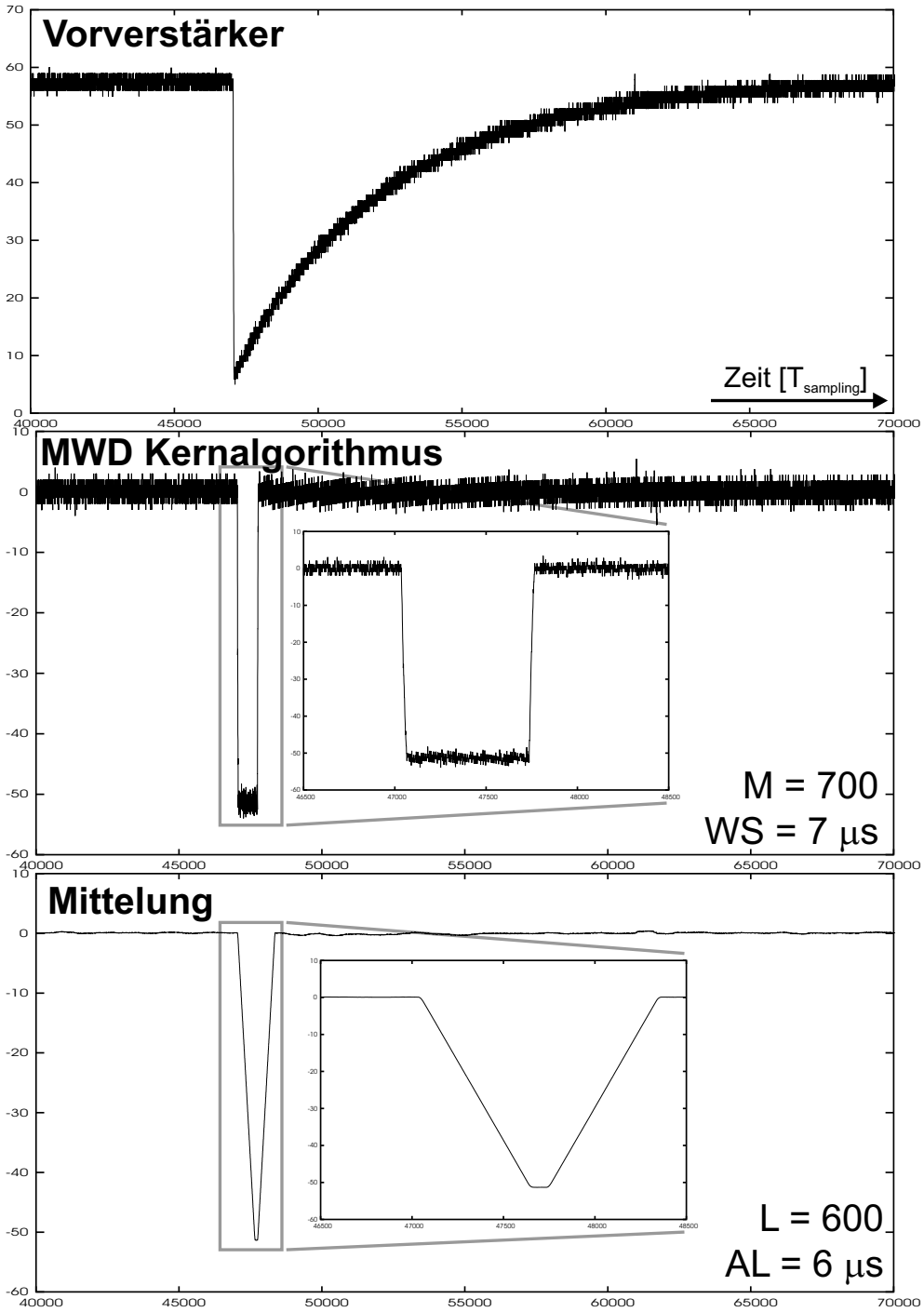


Abbildung 2.6: MWD-Algorithmus für ein reales Signal, aufgenommen mit dem MARS-Detektor (100 MHz, 8 Bit, 300 μs, siehe Abschnitt 3.3). Das Signal stammt von einem der Frontsegmente. Die Vorverstärkerpulse des MARS-Detektors haben negative Polarität.

M und die Länge der arithmetischen Mittelung L. Hinzu kommt ein dritter Wert $1 - k$, der die Zeitkonstante des Vorverstärkers im Verhältnis zur Abtastpariode des ADCs widerspiegelt.

Bedingungen

M und L sind einigen prinzipiellen Bedingungen unterworfen. Es sei Δt_{max} die maximale Dauer eines Strompulses und ΔT_{min} der minimale Abstand zweier Pulse sowie WS $= M \cdot T_{sampling}$ (window size) und AL $= L \cdot T_{sampling}$ (averaging length):

- $WS > \Delta t_{max}$, d.h. die Fenstergröße muss länger als die Pulsdauer des langsamsten zu erfassenden Pulses sein,
- $WS < \Delta T_{min}$, d.h. die Fensterlänge sollte kürzer als der Abstand zweier Pulse sein, und
- $AL \leq WS - \Delta t_{max}$, d.h. die Mittelungslänge darf die Länge des Plateaus des MWD-Ausgangssignales nicht überschreiten.

Zur graphischen Veranschaulichung siehe Abbildung 2.7.

Bedingung eins und zwei gleichzeitig zu erfüllen, ist nicht immer möglich, da die Pulslänge für koaxiale Germaniumdetektoren zwischen 100 ns und 500 ns liegen kann, aufeinanderfolgende Pulse teilweise aber enger beisammen liegen können. Die Begrenzung von AL auf die minimale Plateaulänge rechtfertigt außerdem die Wahl einer konstanten Gewichtungsfunktion zur Rauschunterdrückung, da alle Werte des Plateaus unter dem Rauschen die gleiche Information enthalten.

Auswirkungen

Innerhalb der obigen Grenzen können WS und AL beliebig gewählt werden. Wie erwähnt, wirken sich die beiden Parameter auf die Rauschunterdrückung aus. Je größer der Integrationsbereich WS gewählt wird, umso besser lässt sich der statistische Rauschanteil der Plateauhöhe herausmitteln, und desto

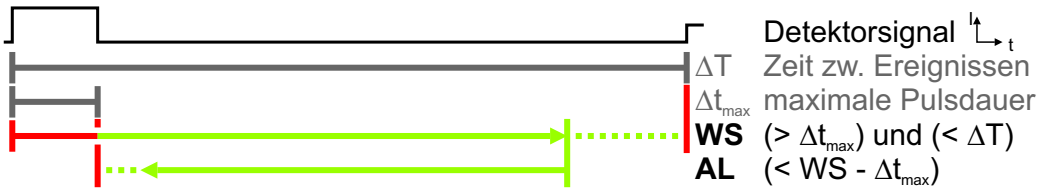


Abbildung 2.7: Bedingungen an die Parameter des MWD-Algorithmus.

genauer lässt sich daher die gesammelte Ladung bestimmen. Idealerweise entspricht AL der Länge des Plateaus, also

$$AL = WS - \Delta t$$

Da aber Δt nicht ohne größeren Aufwand bestimmt werden kann, muss man, um Fehler in der Energiemessung für lange Pulse zu vermeiden, hierfür die Pulsdauer des längsten auftretenden Pulses benutzen, also

$$AL = WS - \Delta t_{max}$$

Dadurch weisen auch die gemittelten Pulse noch Plateaus der Länge $\Delta t_{max} - \Delta t$ auf (siehe Abbildung 2.8).

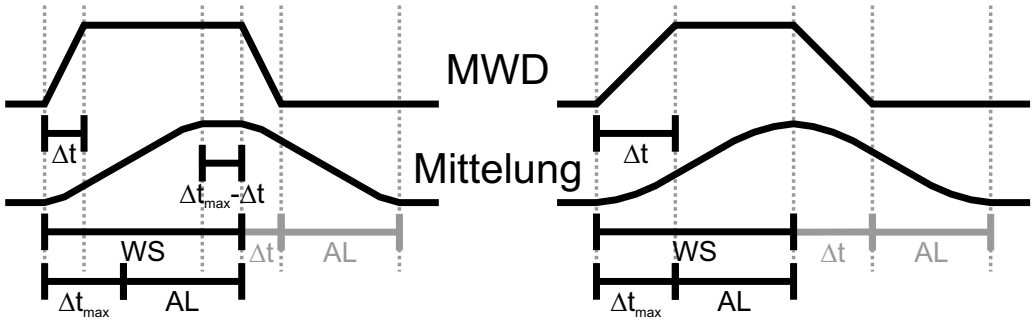


Abbildung 2.8: Schematische Darstellung der Mittelung. Pulse mit einer Pulsdauer $\Delta t < \Delta t_{max}$ erzeugen auch auf dem gemittelten Signal ein Plateau.

Ein großes WS verbessert zwar die Rauschunterdrückung, führt aber tendenziell zu größerem Überlapp der MWD-Pulse (pile-up), was die Auflösung wiederum reduziert[3]

Es gilt also, diese beiden Effekte - verbesserte Rauschunterdrückung vs. pile-up - in Abhängigkeit vom geplanten Experiment (Eventrate, angestrebte Genauigkeit der Energiemessung) gegeneinander abzuwägen und die Integrationzeit WS dementsprechend zu wählen.

Eine Betrachtung des Gesamtfilters, also des MWD-Algorithmus mit anschließender Mittelung, zeigt, dass eine Vertauschung der Werte der Parameter L und M das Ergebnis nur um einen konstanten Faktor ändert (Beweis siehe Anhang C.2). Bedingung 3 kann also umformuliert werden zu

$$|WS - AL| \geq \Delta t_{max} \tag{2.3}$$

[3]Zu Beachten ist jedoch, dass es sich hierbei um pile-ups bei Pulsabständen von typischerweise 5-7 μs handelt, während ein Vorverstärker mit einer Integrationszeit von 50 μs pile-ups schon bei um eine Größenordnung längeren Abständen produziert.

2.1.3 Eichung

Die Eichung der Ergebnisse des MWD-Algorithmus ist unabhängig von den Parametern M und L. Die MWD-Pulshöhe vor der Mittelung ist nicht abhängig von der Wahl der Fenstergröße M. Diese wirkt sich nur auf die Länge des Plateaus aus. Die Mittelung über L Werte im Anschluss führt bei Beachtung der Bedingungen an diesen Parameter zu keiner Veränderung der maximalen MWD-Pulshöhe.

Geeicht werden müssen daher nur die Eigenschaften des ADCs sowie des Detektorsegmentes und seiner analogen Vorverstärkerelektronik. Daher kann eine Eichung mit maximaler Rauschunterdrückung, also möglichst großen Parametern M und L durchgeführt werden, auch wenn diese Fensterbreite und Mittelungslänge später für das eigentlichen Experiment zu lang sein sollten.

Ebenfalls bestimmt werden muss die Integrationszeit τ des Vorverstärkers, z.B. anhand des exponentiellen Abfalls der Antwort des Vorverstärkers auf Deltapulse. Die Auswirkungen einer falsch bestimmten Integrationszeit werden in Abschnitt 4.2.2 näher betrachtet.

2.1.4 Ausblick: Adaptiver Algorithmus

Die Leistungsfähigkeit des Algorithmus kann weiter verbessert werden, indem man den Parameter M an die aktuelle Eventrate im Detektor anpasst, d.h. für jeden einzelnen Puls eine maximale Fenstergröße bestimmt. Bevor die Daten in den MWD-Algorithmus eingespeist werden, muss also eine Online-Analyse des Datenstroms nach Pulsen stattfinden und der Integrationsbereich M dem Abstand zum nächsten Puls angepasst werden. In diesem Fall sollten die ermittelten Energiewerte jeweils mit einem Qualitätsmerkmal für die Rauschunterdrückung gekennzeichnet werden, um die verschiedenen Rauschniveaus bei der Erstellung eines Spektrums berücksichtigen zu können.

2.2 MWDTrigger

Mit einem angepassten Parametersatz lässt sich der MWD-Algorithmus auch zur Triggererzeugung bzw. zum Aufspüren von Ereignissen im Datenstrom einsetzen, z.B. für den adaptiven MWD-Energiefilter. Die Anforderungen an einen solchen Algorithmus sind, dass

- niederenergetische Ereignisse (niedrige Pulshöhe)
- mit langer Pulsdauer (flacher Anstieg)

- bei hoher Ereignisrate (kleiner Abstand der Pulse voneinander)

zuverlässig erkannt werden. Der MWD-Algorithmus bietet sich hierfür an, da er ein von der Eingangspulsform (und damit Pulslänge) unabhängiges Ergebnis (MWD-Pulshöhe) liefert.

2.2.1 Funktionsweise

Im Prinzip ist die Vorgehensweise die Gleiche wie für die analoge Triggergenerierung. Dort wird für diese Aufgabe, anders als für die analoge Energiemessung, ein Pulsformer mit möglichst kleiner Integrationszeit τ verwendet. Dies führt zwar aufgrund des ballistic deficit zu einer starken Verzerrung der Pulshöhe, verringert aber die Anzahl von pile-ups, generiert also für möglichst viele Ereignisse eigenständige Signale.

Dementsprechend werden auch die Parameter des MWD-Triggers möglichst klein gewählt. Die Länge WS ($\hat{=}$ M) des Integrationsfensters wird auf die maximale Länge eines Pulses ($\approx$ 500ns) gesetzt.

$$M = \frac{\Delta t_{max}}{T_{sampling}}$$

Damit ist sichergestellt, dass das Maximum des MWD-Pulses für jedes Ereignis der gesamten erzeugten Ladung entspricht, langsame Pulse also nicht diskriminiert werden. Auch hier ist es sinnvoll, zur Rauschunterdrückung zu mitteln, um ein Auslösen des Triggers durch Spitzen im Rauschen zu verhindern. Unter Beachtung von Gleichung 2.3 bedeutet dies $L = 0$ oder $L \geq 2 \cdot M$. Setzt man

$$L = 2 \cdot M \ ,$$

so entspricht dies der maximalen Länge eines MWD-Pulses, wenn $WS = \Delta t_{max}$ gewählt wurde. Aufgrund der Punktspiegelsymmetrie der ansteigenden und abfallenden Flanke des MWD-Pulses ist die Gesamtfläche unter einem solchen Puls immer WS $\cdot Q$ (siehe Abb. 2.9). Mittelt man einen solchen Puls über $L = 2 \cdot M$ Werte, so ist das Maximum dieses gemittelten MWD-Pulses proportional zur Energie des Ereignisses.

Mittels eines anschließenden Schwellendiskriminators ist es somit möglich, relativ genau auf Ereignisse mit einer definierten Minimalenergie zu triggern. Die zeitliche Trennung zweier Pulse kann durch einen zusätzlichen Maximumdetektor verbessert werden.

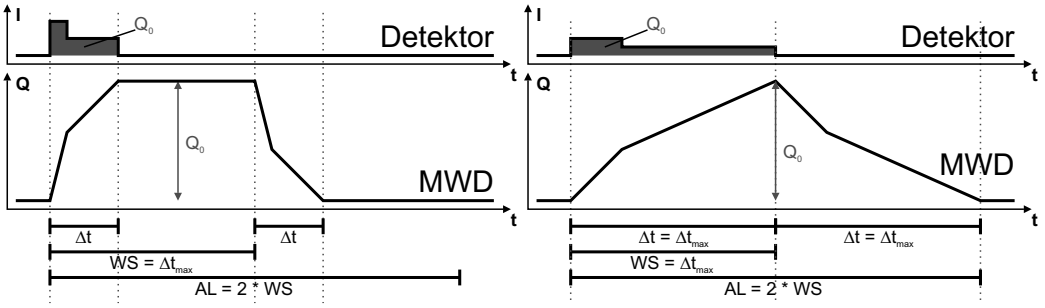


Abbildung 2.9: Schematische Darstellung zweier Pulse. Der linke Puls hat eine Länge von $\Delta t = \frac{1}{3}\Delta t_{max}$, der rechte die Maximallänge von $\Delta t = \Delta t_{max}$.

2.3 SCC

Der *Slope Condition Counter* (SCC) von W. Gast ([12], [10]) verfolgt einen statistischen Ansatz zur Generierung von Triggersignalen. Ziel ist es, ansteigende oder abfallende Flanken im Signal zuverlässig auch bei hohem relativem Rauschniveau herauszufiltern, und zwar unabhängig von der Signalform.

2.3.1 Funktionsweise

Der Algorithmus überprüft für jeden Datenpunkt in einem umliegenden Bereich der Länge $2M+1$ mittels einfacher Größer-Kleiner-Vergleiche, ob dieser Bereich eine ansteigende bzw. abfallende Flanke enthält. Der Mittelpunkt des momentan betrachteten Bereichs ist der Referenzwert. Dieser wird mit allen Werten vor und nach ihm innerhalb des Fensters verglichen und die gewichteten $2M$ Ergebnisse dieser Vergleiche summiert. Für den Bereich vor und hinter dem Referenzpunkt gelten jeweils unterschiedliche Kriterien für eine ansteigende(abfallende) Flanke:

- Die dem Referenzpunkt vorangehenden M Werte gehen mit 1 gewichtet in die Summe ein, wenn sie größer(kleiner) als dieser sind, ansonsten mit 0,
- die dem Referenzpunkt nachfolgenden M Werte gehen mit -1 in die Summe ein, wenn sie kleiner(größer) sind, alle anderen wiederum mit 0.

Man erhält also für jedes Fenster als Ergebnis eine Summe von $2M$ Werten aus der Menge $\{-1, 0, 1\}$. Graphisch veranschaulicht ist dieser Vorgang in Abbildung 2.10.

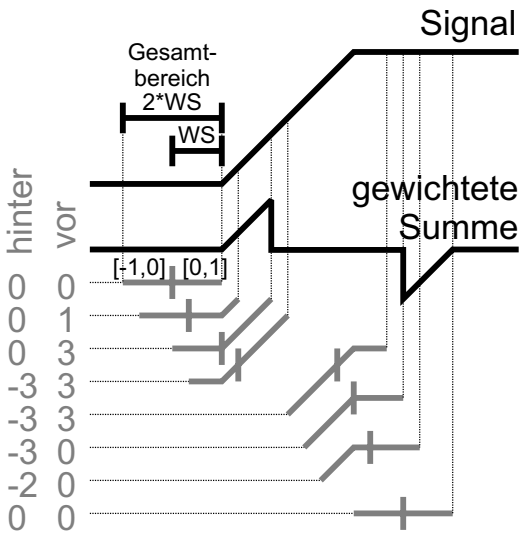


Abbildung 2.10: Durch eine Addition der gewichteten Vergleichsergebnisse eines Fensters erhält man eine Sägezahnstruktur. Es ist $WS = M \cdot T_{sampling}$.

Wie aus Abbildung 2.11 hervorgeht, verschwindet das gesuchte Signal zunächst komplett im Rauschen, da der Algorithmus insensitiv für die Länge einer Flanke ist, und somit auch herausragende Deltapeaks als Anstieg oder Abfall gewertet werden und einen Ausschlag maximaler Amplitude erzeugen können. Charakteristisch für das gewichtete Signal einer zeitlich endlichen Flanke ist das punktsymmetrische Sägezahnprofil. Durch eine Integration des gesamten gewichteten Signals wird diese Struktur deutlich aus dem Rauschen hervorgehoben.

Wie beim MWD-Algorithmus wird das Signal anschließend zur Rauschunterdrückung arithmetisch gemittelt und dann über einen Maximumdetektor in Kombination mit einem Schwellendiskriminator das Triggersignal erzeugt. Abbildung 2.12 zeigt den SCC-Triggeralgorithmus, angewendet auf ein reales Signal.

2.3.2 Pulsform

Das gewichtete Signal für eine Flanke hat zunächst eine punktsymmetrische Sägezahnform mit einer Länge von $(2 \cdot WS) + \Delta t$, wobei Δt die Anstiegszeit der Flanke ist (siehe Abb. 2.11) und $WS = M \cdot T_{sampling}$. Integriert man über diese Struktur, so ergibt sich ein spiegelsymmetrischer Peak der selben Breite mit einem Plateau der Länge Δt und parabelförmigen Flanken der Länge WS. Die Amplitude F dieses Peaks entspricht der Fläche unter der Sägezahnstruktur, kann also einen maximalen Wert von $F_{max} = 0.5 \cdot M^2$

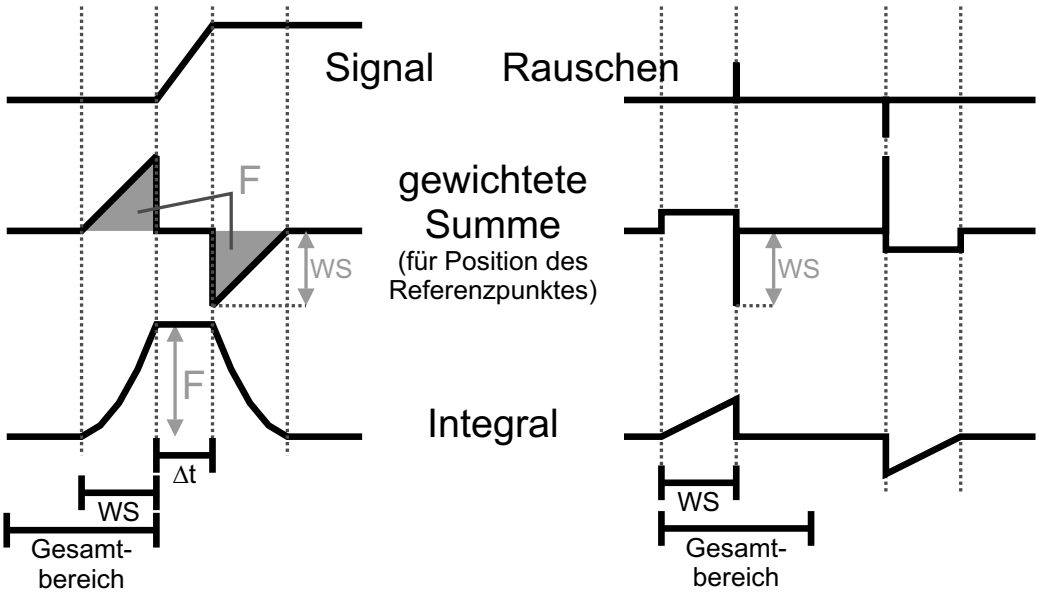


Abbildung 2.11: Schematische Darstellung der Pulsformen für eine ansteigende Flanke ohne Rauschen sowie für Deltapeaks. Es ist zu erkennen, dass die Amplitude der gewichteten Summe für beide Arten gleich ist. Daher ist eine Integration über dieses Signal zur Verbesserung der Unterscheidbarkeit notwendig. Die Ausgangssignale der beiden Stufen des SCC-Triggers sind hier im Gegensatz zu Abbildung 2.10 um $WS = M \cdot T_{sampling}$ verschoben. Das Ergebnis ist hier für den Referenzpunkt aufgetragen, um zeitliche Korrelationen und Längen deutlich zu machen.

erreichen. Die anschließende Mittelung verbreitert den Peak um die Mittelungslänge AL. Der Ausgangspuls des SCC-Algorithmus hat also insgesamt eine Breite von $(2 \cdot WS) + AL + \Delta t$.

Transiente Signale erzeugen je nach Länge und Form recht unterschiedliche gewichtete Summen, wie man anhand Abbildung 2.13 erkennen kann. Vereinfacht gesagt stellen die transienten Signale für den SCC-Triggeralgorithmus je nach gewählter Fensterlänge WS ein Mittelding zwischen Deltarauschpeak und Stufensignal[4] dar. Das gilt auch nach der Integration, da die dabei entstehenden Peaks oberhalb des Rauschens liegen, aber nicht die Maximalhöhe von $0.5 \cdot M^2$ erreichen.

[4]Dem SCC-Trigger erscheinen alle Signale mit einer Pulsdauer Δt länger als $2 \cdot WS + 1$ als einzelne Flanken.

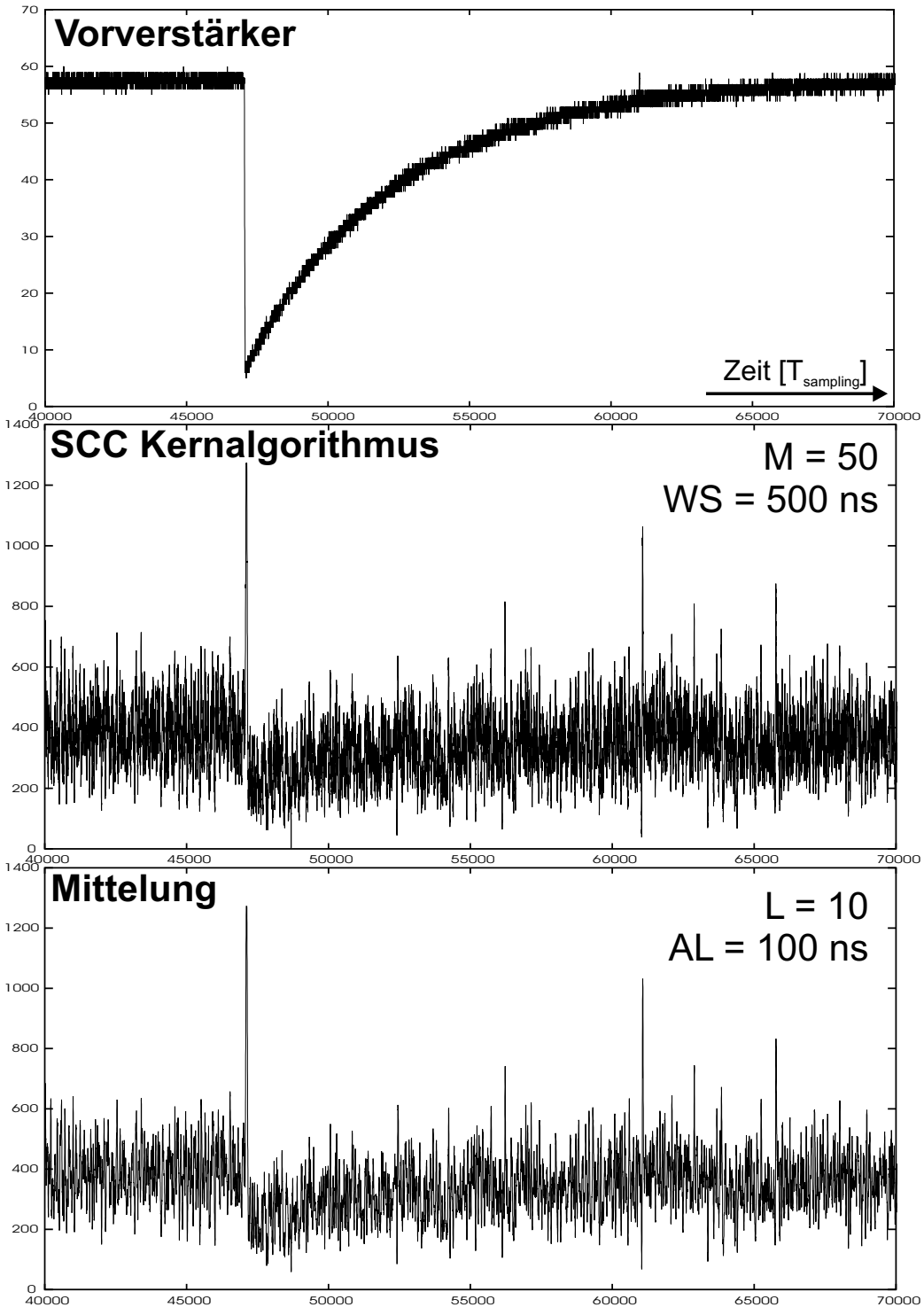


Abbildung 2.12: Der SCC-Trigger für ein reales Signal des MARS-Detektors. Es handelt sich um denselben Ausschnitt wie in Abbildung 2.6.

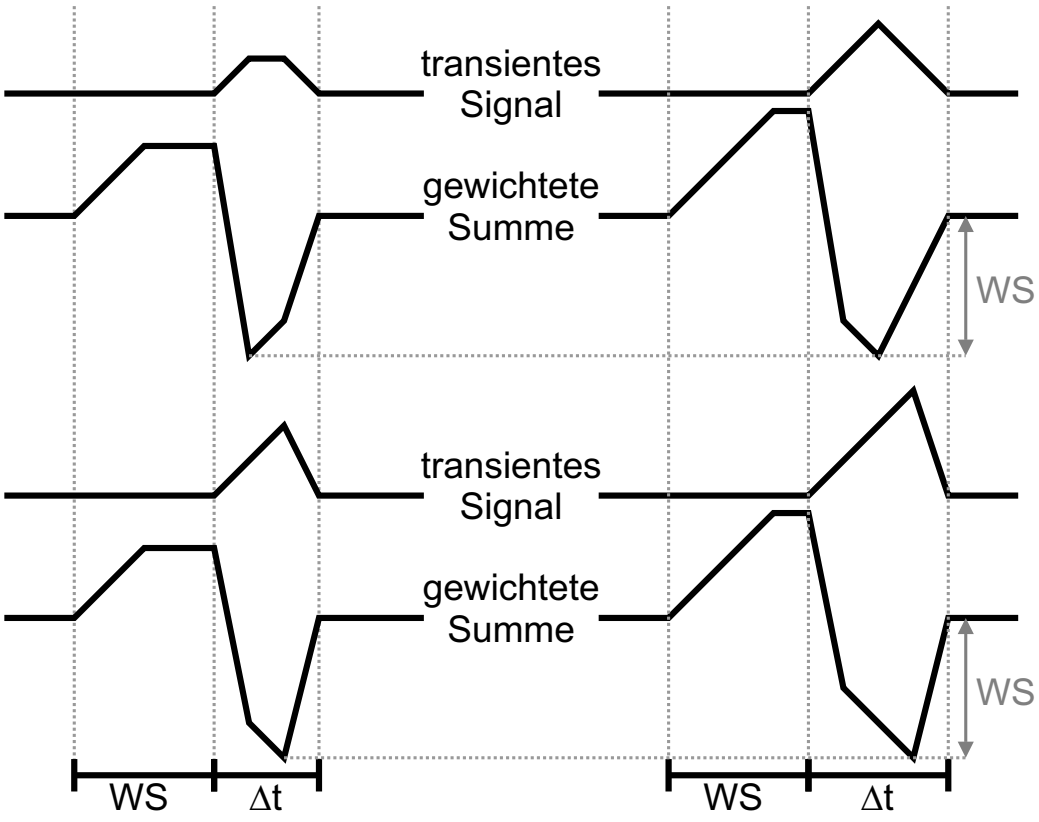


Abbildung 2.13: Schematische Darstellung der Pulsformen für vier verschiedene transiente Signale. Die Amplituden der gewichteten Summen ereichen dabei die gleichen Werte wie im Falle von Nettoladungssignalen. Transiente Signale mit negativer Polarität erzeugen punktgespiegelte gewichtete Summen.

2.3.3 Parameter

Der SCC-Kernalgorithmus besitzt wie der MWD-Algorithmus die zwei Parameter M und L, also die Breite des zu betrachtenden Bereichs und eine Mittelungslänge. Im Gegensatz zum MWD sind diese aber hier keinen prinzipiellen Einschränkungen unterworfen.

M und L können unabhängig voneinander gewählt werden. M beeinflusst, wie stark die Sägezahnsignatur im gewichteten Signal ausgeprägt ist, was sich nach der Integration in der Pulshöhe ausdrückt (siehe Abb. 2.14). Nachteilig wirkt sich ein großes M allerdings auf die zeitliche Trennbarkeit zweier Pulse aus, da die Pulsbreite des integrierten Signals $(2 \cdot WS) + \Delta t$ mit $WS = M \cdot T_{sampling}$ beträgt, Pulse also mindestens um diese Zeit auseinander liegen müssen, um sauber getrennt zu werden. Außerdem erfordert jeder zusätzliche

Vergleichspunkt in der späteren Hardwareimplementierung mehr Resourcen, im Gegensatz zum MWD-Algorithmus, dessen benötigte Rechenleistung von der Fenstergröße unabhängig ist[5].

Die Mittelung dient dazu, eine saubere Maximumerkennung zu ermöglichen. Es reicht also aus, die Spitzen der Peaks zu glätten. Die Peaks besitzen, wie oben erwähnt, ein unter dem Rauschen verstecktes Plateau, dessen Breite der jeweiligen Flankenlänge Δt entspricht. Eine Mittelung über eine Länge von 100 bis 500 ns, ein Bereich, der den typischen Pulslängen eines koaxialen Germaniumdetektors entspricht, sollte also zu einer ausreichenden Glättung dieser Spitzen führen. AL = 100 ns lässt die Pulshöhe dabei unverändert. Wird AL zu groß gewählt, so verändert dies die Pulshöhen nach unten, was sich auf Grund des Einsatzes des Schwellendiskriminators negativ auf Triggereffizienz auswirkt (siehe Abb. 2.15).

2.4 Maximumdetektor und Schwellendiskriminator

Sowohl für den MWD- wie auch den SCC-Algorithmus ist in einem letzten Schritt die Bestimmung der Amplituden der von ihnen erzeugten Pulse notwendig. Für die beiden Trigger (SCC und MWDTrigger) reicht es aus zu verifizieren, ob der betrachtete Puls eine gesetzte Schwelle überschreitet. Im Gegensatz dazu ist der MWD-Energiealgorithmus auf eine möglichst genaue Bestimmung der maximalen Amplitude angewiesen.

Dies kann auf mehrere Arten geschehen. Möglichkeit eins ist, den Zeitpunkt des Erreichens des Maximalwertes zu bestimmen und den zugehörigen Datenwert auszulesen. Eine andere Möglichkeit, die Amplitude zu bestimmen ist, ohne Berücksichtigung der Form des gemittelten MWD-Ausgangssignales einfach den Maximalwert aus dem Bereich dieses MWD-Pulses zu verwenden. Der gemittelte MWD-Puls erreicht spätestens nach einer Zeit WS ab Beginn des ursprünglichen Signals den auszuwertenden Extremwert, so dass es genügt, das Maximum in einem Fenster der Breite WS nach Auslösen eines Triggers zu bestimmen.

2.4.1 Maximumdetektor

Die exakte Maxima- oder Minimadetektierung kann auf verschieden Arten erfolgen:

[5]Das gilt nicht für den Datenpuffer im MWD-Algorithmus, der die Eingangswerte über die Länge des Fensters hinweg speichert.

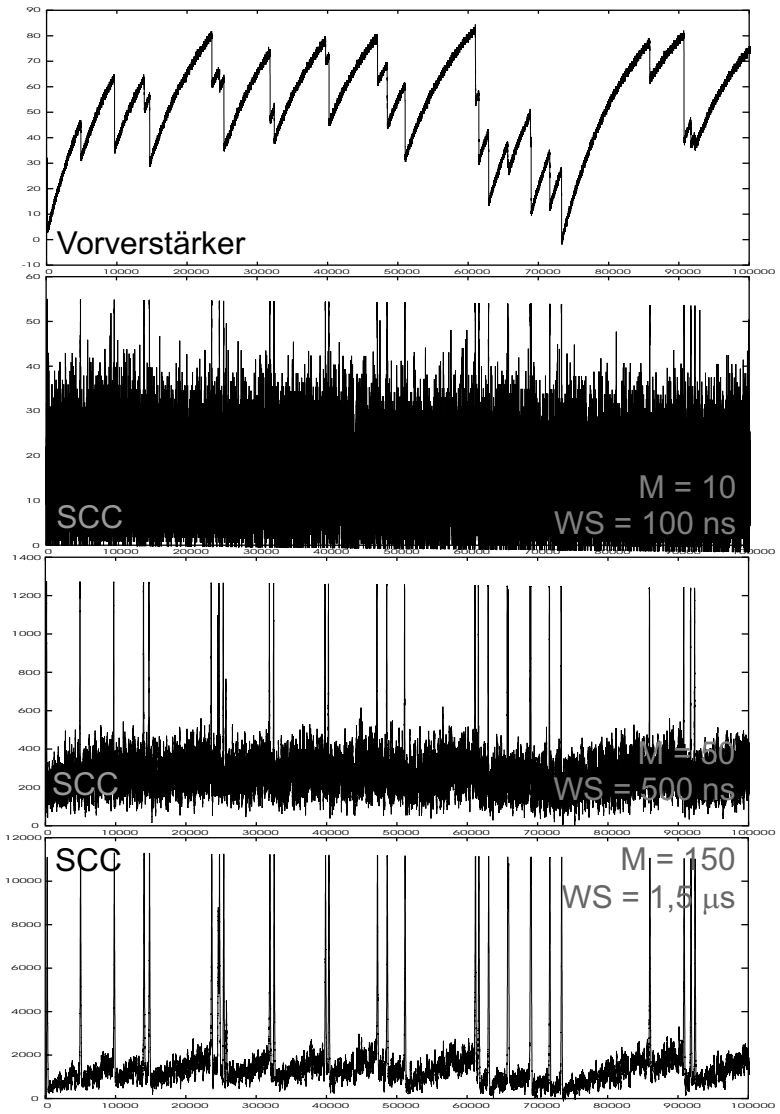


Abbildung 2.14: Drei verschiedene Werte für den SCC-Parameter WS. Je größer WS gewählt wird, desto deutlicher treten die Peaks zum Vorschein. Keine anschließende Mittelung.

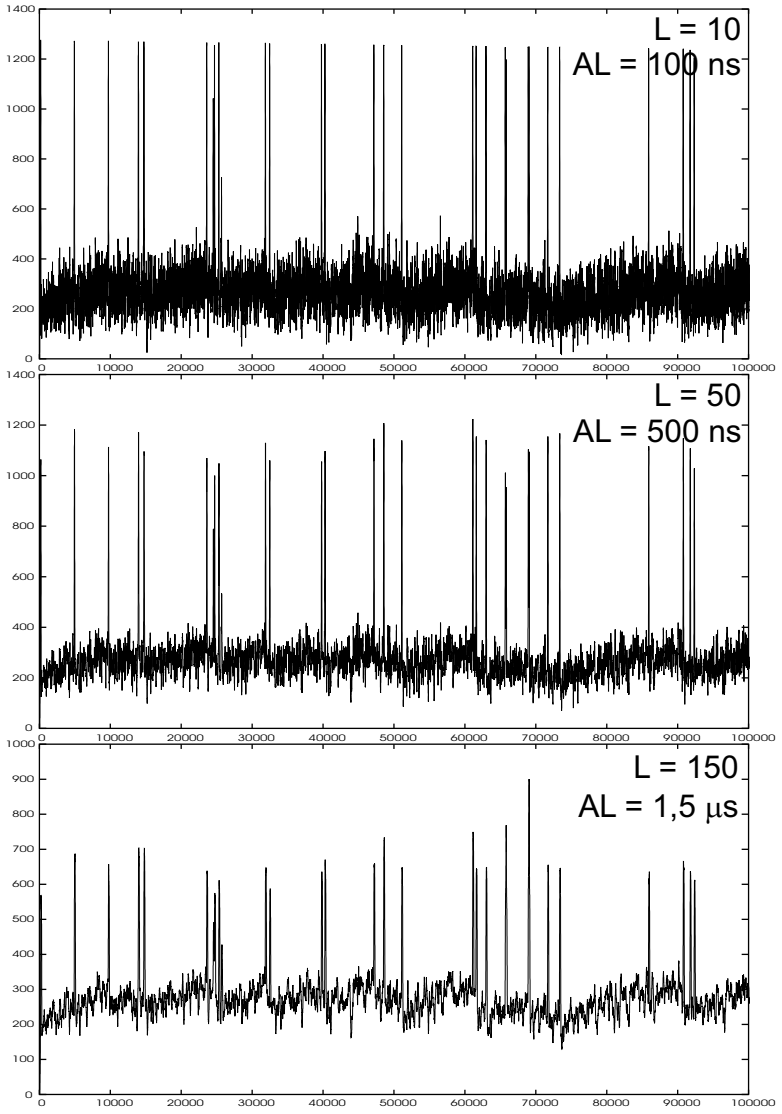


Abbildung 2.15: Drei Werte für den Parameter AL, WS beträgt 500 ns (siehe Abb. 2.14). Je größer AL ist, umso weiter sinken die Spitzen der Peaks ab und ihre Breite nimmt zu.

Zum einen statistisch wie beim SCC-Trigger, d.h. man betrachtet einen Bereich vor und nach einem Referenzpunkt und vergleicht die Werte in diesem Bereich mit dem Referenzpunkt. Eine Gewichtung der Vergleichsergebnisse und die Summation dieser Gewichtungswerte erlauben eine Wahrscheinlichkeitsaussage darüber, ob der Referenzpunkt ein lokales Maximum oder Minimum darstellt.

Ist das Rauschniveau des betrachteten Signals gering, so kann man die Position der maximalen Amplitude auch durch Differentiation und Nulldurchgangsbestimmung ermitteln.

Schwierigkeiten bereiten beiden Detektionsarten Pulse mit Plateau.

2.4.2 Schwellendiskriminator

Der Ausgang des Schwellendiskriminators ist binär. Der Filter erzeugt eine 1, wenn der momentane Datenwert oberhalb oder unterhalb einer zuvor gesetzten Schwelle liegt. Mittels einer Differenzenbildung ($y[n] = x[n] - x[n-1]$) dieses Ausgangssignales lässt sich dann ein eindeutiger Triggerpuls generieren (siehe Abbildung 2.16).

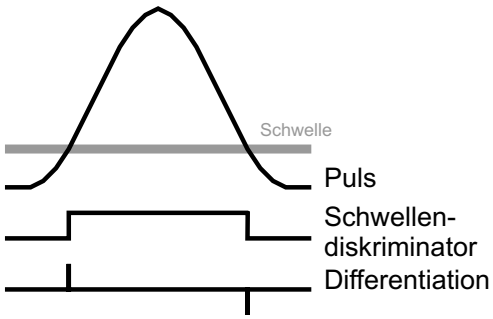


Abbildung 2.16:

Kapitel 3

Experimentierumgebungen

3.1 C++ Programmierung

3.1.1 Grundstruktur

Den im Rahmen der Diplomarbeit entwickelten C++-Programmen zur Analyse und zum Test der verschiedenen Algorithmen liegt ein hochmodularisierter, objektorientierter generischer Code zugrunde. Dies erlaubt eine schnelle Anpassung an neue Datenformate und Algorithmen.

Der Programmcode besteht zum größten Teil aus aufeinander aufbauenden Klassen. Ausgehend von einer Basisklasse teilt sich die Klassenstruktur in drei Hauptäste, den

- Filterklassen, die die Basis der zu testenden Algorithmen bilden, den
- Datenklassen, die für die Verwaltung und Speicherung der Daten zuständig sind und den
- Datenquellenklassen, die Schnittstellen zu verschiedenen externen Datenformaten implementieren.

Die Struktur der Filterklassen und deren Einsatz orientiert sich an der modularen Beschreibung des VHDL-Codes. Die Filterbasisklassen stellen einen Operator zur Verfügung, der ein unkompliziertes Verbinden der Filter und Daten zu Filterketten erlaubt. Alle Filter sind, soweit sinnvoll, in Templates definiert, wodurch sie flexibel für verschiedene (auch selbstdefinierte) Datentypen - z.B. Integer, Fließkomma oder komplexe Zahlen - verwendet werden können, soweit diese Datentypen die für den jeweiligen Algorithmus benötigten Rechenoperationen erlauben.

Die Datenklassen dienen dazu, den Zugriff auf die Daten im Speicher zu vereinfachen und bieten vielfältige Möglichkeiten, die Daten umzuorganisieren und zu bearbeiten. Daneben erlauben sie den Export der Daten in andere Formate, z.B. als eps- oder ROOT-Datei. Dafür binden sie einige Klassen des ROOT-Frameworks[1] ein. Auch diese Klassen sind generisch gehalten und erlauben so eine hohe Wiederverwendbarkeit.

Die Datenquellenklassen überlappen bezüglich ihrer Funktionalität teilweise mit der der Datenklassen. Sie definieren eine datentypunabhängige Schnittstelle für externe Daten zum restlichen Programmcode. Die Daten können zweidimensional organisiert sein, z.B. mehrere Messungen auf mehreren Kanälen enthalten.

Der Programmcode erlaubt es, Debuginformationen je nach Bedarf gezielt für einzelne Klassen oder für das gesamt Projekt beim Kompilieren einzufügen und verfügt über eine automatische Speicherüberwachung.

Einen detaillierten Überblick über den Klassenbaum gibt Anhang E.

3.1.2 Werkzeuge

Aufbauend auf dem PSAClass-Klassenbaum sind neben den eigentlichen Filtern eine ganze Reihe von kleineren Programmen zur Datenkonvertierung und -auswertung entstanden, unter anderem, um einzelne Kanäle oder Ereignisse aus den MARS-Datensätzen zu extrahieren, mehrere Datensamples in einer Struktur zusammenzufassen, Daten in andere Formate (ROOT, eps, ...) zu konvertieren oder am Bildschirm anzuzeigen.

Die Programme lassen sich sowohl unter Linux wie auch Windows kompilieren. Unter Windows stehen allerdings die ROOT-Routinen nicht zur Verfügung.

3.2 VHDL Programmierung

Für die Programmierung in VHDL wurde die ISE Design Tool Software der Firma Xilinx verwendet. Für die Synthetisierung des Codes kam der darin enthaltene Compiler zum Einsatz. Zur Simulation wurde das ModelSim XE/Starter-Paket[2] der Firma Model Technology eingesetzt.

[1] http://root.cern.ch

[2] ModelSim XE/Starter ist eine von Xilinx unterstützte Version. Die Starter-Lizenz führt dazu, dass Simulationen je nach Modulgröße durch die Software künstlich in die Länge gezogen werden.

3.2.1 VMEbus-FPGA-Karte

Die VHDL-Algorithmen werden auf einer am Lehrstuhl E18 des Physikdepartments der TUM entworfenen VMEbus-Karte [20] getestet. Diese Karte ist ursprünglich zur Triggersteuerung und -generierung im COMPASS[3]-Experiment am CERN[4] konzipiert worden, bietet aber für einen ersten Test der Algorithmen alle erforderlichen Komponenten. Erklärungen zu einigen Abkürzungen und Begriffen befinden sich in Anhang D.

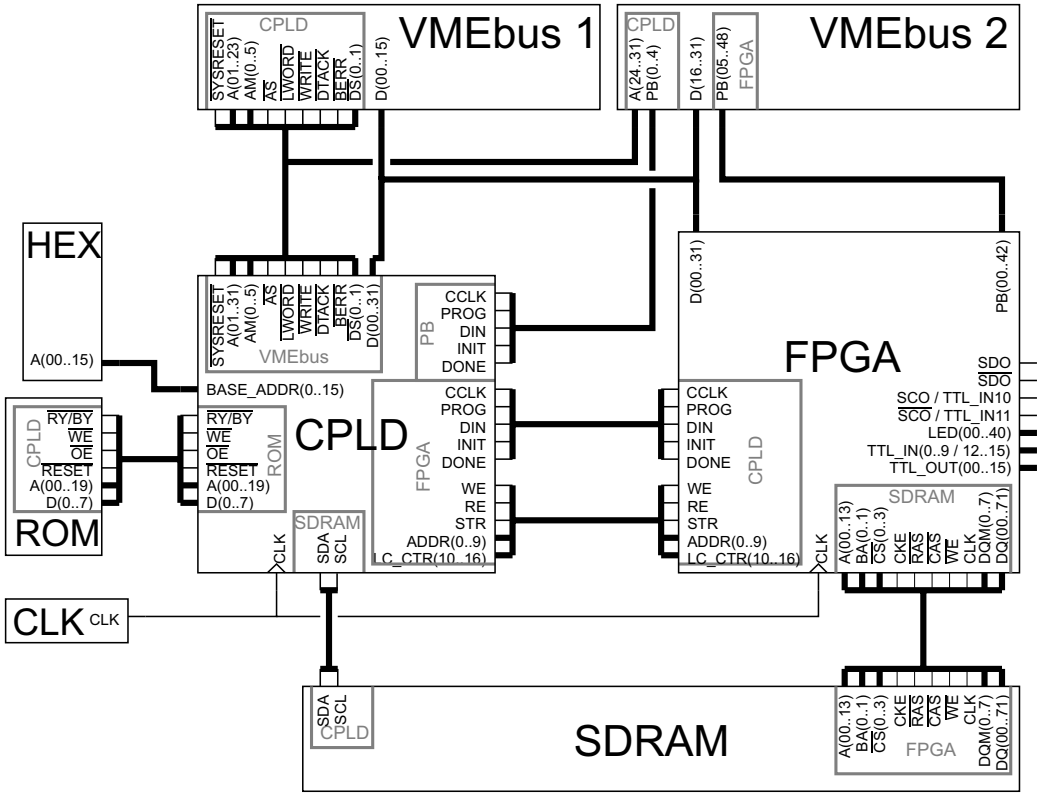


Abbildung 3.1: Skizze der TCL3.0-Karte

Die Karte besitzt 16 Ein- und 16 Ausgänge für NIM-Signale sowie einen optischen Eingang auf der Frontseite. Diese sind, nach einer Umsetzung auf TTL-Signale, mit einem FPGA-Baustein - dem Herz der Karte - verbunden. Dem FPGA zur Seite gestellt ist ein CPLD, der ein rudimentäres VMEbus-Interface für die Karte bereitstellt. Über diesen Chip kann der FPGA nach dem Einschalten der Spannungsversorgung der Karte entweder über den

[3]COmmon Muon Proton Apparatus for Structure and Spectroscopy, http://wwwcompass.cern.ch/

[4]European Organization for Nuclear Research, http://www.cern.ch/

VMEbus oder einen (momentan nicht bestückten) PROM-Baustein auf der Karte initialisiert werden. Daneben befinden sich ein Sockel für einen 168pin SDRAM DIMM, verbunden mit dem FPGA, sowie ein Frequenzgenerator mit 38,88MHz zur Taktgenerierung für die beiden Logik-Bausteine auf der Karte. Die VMEbus-Basisadresse lässt sich über vier Hexadezimal-Drehschalter (momentan nur drei aktiv), verbunden mit dem CPLD, einstellen. Zusätzlich hat der FPGA Zugriff auf die im VMEbus-Standard nicht definierten Leitungen sowie den VMEbus-Datenbus auf der Rückwandplatine.

	Hersteller	Typ	Bezeichnung
CPLD	Xilinx	XC9500XL	xc95288xl-10fg256
FPGA	Xilinx	Virtex-II	xc2v3000-4fg676
SDRAM	Infineon		HYB39S256800CT-7.5

Tabelle 3.1: Verwendete Bausteine auf der TCL3.0-Karte

Zwei Besonderheiten der Karte sind erwähnenswert:

- Da die Karte zur Triggersteuerung gebaut wurde, sind die 16 NIM-Eingänge sowie der optische Eingang mit den Clock-Eingängen des FPGAs verbunden, um eine optimale Flankenerkennung des Triggersignales zu ermöglichen. Dadurch ist der Frequenzgenerator der Karte an einen Standard-I/O-Pin des FPGAs angeschlossen und muss somit chip-intern auf eines der Takt-Verteilungs-Netzwerke gelegt werden.
- Die SPD-Leitungen[5] SDA und SCL des SDRAM-Modules zur Auslese der Modulspezifikationen sind im Gegensatz zu den restlichen SDRAM-Leitungen mit dem CPLD und nicht mit dem FPGA verbunden.

Die für die Implementierung der Algorithmen und deren Test wichtigen Elemente der Karte sind der FPGA und der SDRAM-Speicher sowie die Programmierung des VMEbus-Interfaces auf dem CPLD.

FPGA

Der auf der Karte verwendete FPGA ist ein Virtex-II der Firma Xilinx, Inc. bestehend aus 3.000.000 Logik-Gattern. Daneben befinden sich 96 18x18 Bit Multipliziereinheiten sowie dieselbe Anzahl an frei konfigurierbaren 18 kBit Block-RAM-Modulen auf dem Chip. Hinzu kommen 12 DCM-Blöcke (Digital Clock Management), die Möglichkeiten zur Laufzeitkompensation der Clock sowie zur Frequenzsynthese bieten. Für detailliertere Informationen siehe [4].

[5]SPD - Serial Presence Detect

Speicher

Das verwendete Speichermodul ist ein 512MB PC133 ECC Registered SDRAM-DIMM. Aufgrund des zusätzlichen neunten ECC-Bits pro Byte beträgt die gesamte verfügbare Speichergröße 576 MB (entspricht 603.979.776 Byte), und der Datenbus zum Modul ist 72 Bit breit.

CPLD-Programmierung

Die momentane Programmierung[6] des CPLDs erlaubt VMEbus-Transfers im A24D32-Modus, d.h. mit einer Adressbreite von 24 Bit bei einer Datenbreite von 32 Bit. Das Interface arbeitet als slave-device am Bus. Die Basisadresse der Karte wird mittels drei der vier mit dem CPLD verbundenen Hexadezimal-Schalter eingestellt. Das Interface wird aktiv, wenn zum Zeitpunkt eines Adressimpulses $\overline{\text{AS}}$

- die oberen 12 Bit A12 bis A23 der VMEbus-Adresse mit der Basisadresse der Karte übereinstimmen,
- der Adressmodifizierer AM0 bis AM5 den Wert 0x39 oder 0x3D (111X01) hat, was dem A24D32-Modus entspricht,
- $\overline{\text{IACK}}$ auf eins liegt, d.h. kein Interrupt am Bus anliegt, und
- A01 auf 0 liegt, der Master also einen vollständigen 4-Byte-Transfer verlangt.

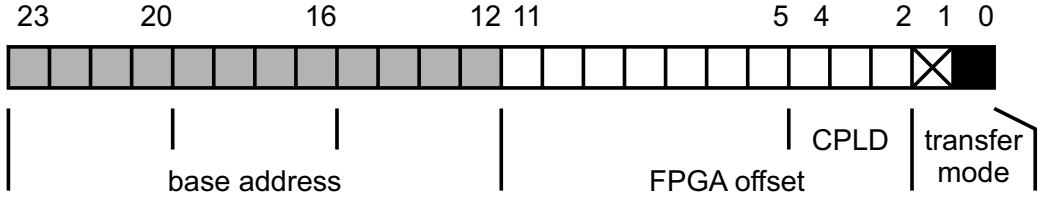


Abbildung 3.2: VMEbus-Adressbelegung der TCL3.0-Karte.

Es bleibt aktiviert bis zum nächsten Adressimpuls. Solange die Karte über den Bus angesprochen wird, analysiert der CPLD den Adressoffset A02 bis A11 zur Basisadresse (siehe auch Abb. 3.2). Die momentane Programmierung bearbeitet drei der 1024 möglichen Adressen CPLD-intern:

[6] von Igor Koronov, E18, Physikdepartment, TU-München

0x14 zur Initialisierung des FPGAs auf der Karte im slave-serial-mode[7]. Die Konfigurationspins DONE, PROG, CCLK, $\overline{\text{INIT}}$ und DIN des FPGAs werden dafür auf die VMEbus-Datenleitungen D00 bis D02 abgebildet.

0x10 um einen externen Xilinx-Chip im serial-slave-mode auf einer anderen Karte zu programmieren. Dafür werden einige der nicht standardisierten VMEbus-Leitungen benutzt.

0x0C ermöglicht das Lesen und Beschreiben eines CPLD-internen Testregisters.

Auf alle drei obigen Adressen kann lesend und schreibend zugegriffen werden. Offsets größer als 0x1C werden vom CPLD ignoriert und auf den FPGA abgebildet. Dafür werden die VMEbus-Adressleitungen A02 bis A11 im CPLD zum FPGA durchgeschleift. Dem FPGA stehen daher 1016 Adressen zwischen 0x020 und 0x3FF zur Verfügung.

Im Falle einer Schreibaufforderung durch einen VMEbus-Master (d.h. es sollen Daten an die Karte gesendet werden) erzeugt der CPLD synchron zum Kartentakt ein write-enable- und data-strobe-Signal für den FPGA, wenn der Adressoffset höher oder gleich 0x20 ist. Eine Leseanforderung mit einer solchen Adresse ergibt ein read-enable-Signal, jedoch wird kein Datenimpuls generiert. Der CPLD gibt dem FPGA vier Kartentakte Zeit, gültige Werte an die Datenleitungen anzulegen und erzeugt dann automatisch einen data-acknowledged-Puls auf dem VMEbus um den Transferzyklus damit abzuschließen.

3.2.2 Genereller Aufbau der FPGA-Programmierung

Neben der Implementierung der reinen Algorithmen in VHDL war zum Test derselben auf dem FPGA die Programmierung weiterer Module zur Datenzu- und abfuhr nötig. Die Zielvorgabe, 14 Bit mit 100 MHz zu verarbeiten, entspricht einer Datenrate von etwas unter 200 MB/s, die der Algorithmus als Input benötigt. Daneben ist es sinnvoll, zur Überprüfung der Algorithmen in der Lage zu sein, Teilergebnisse, die zur Erreichung einer höheren Genauigkeit teilweise um bis zu 20 Bit aufgeweitet sind, speichern und analysieren zu können. Dies führt dazu, dass Gesamttransferraten von bis zu 600 MB/s notwendig sind. Die maximale vom VMEbus zur Verfügung gestellte Kapazität von 80 MB/s ist hier bei weitem zu gering.

[7]Der slave-serial-mode ist eine von mehreren Möglichkeiten der Initialisierung der Xilinx-FPGAs.

Daher werden die Daten zunächst in einem als FIFO[8]-Speicher angesteuerten SDRAM-Modul auf der Karte zwischengespeichert. Bei einer Speicherkapazität von insgesamt 576 MB sowie gleichzeitiger Entnahme der Rohdaten und Speicherung der Ergebnisse können damit im Fall das MWD-Algorithmus Datensätze von bis zu 1,3 s Länge auf der Karte mit 100 MHz verarbeitet werden, was für einen realistischen Test ausreicht.

Die im Rahmen der Diplomarbeit entwickelte Programmierung des FPGAs besteht in der obersten Ebene aus vier Blöcken (siehe Abb. 3.3):

- einer *CPLD-VMEbus-Schnittstelle*, welche den Datentransfer von und zum VMEbus in Zusammenarbeit mit dem CPLD abwickelt,
- einer als FIFO ausgeführten *SDRAM-Schnittstelle* zur Zwischenspeicherung der Rohdaten und Ergebnisse,
- den zu testenden Algorithmen, die mit dem SDRAM-FIFO verbunden sind, und
- einer Steuereinheit, die den Datenfluss zwischen den Modulen je nach Arbeitsschritt steuert.

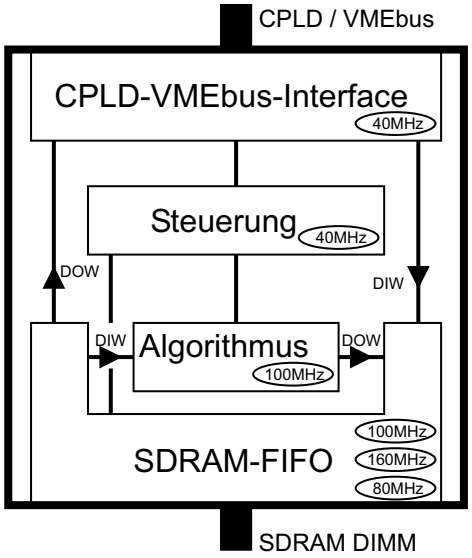


Abbildung 3.3: Oberste Hierarchieebene der FPGA-Programmierung.

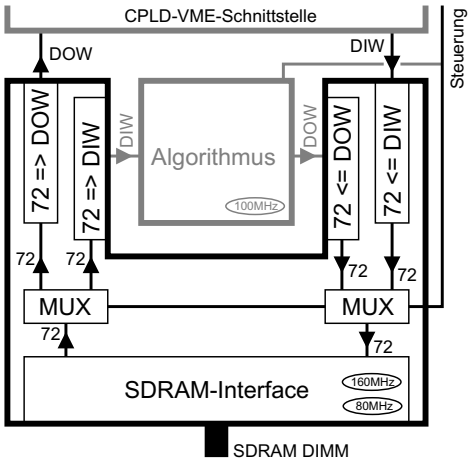


Abbildung 3.4: Schematische Darstellung des SDRAM-FIFO-Moduls.

[8]first-in-first-out

Um die Wiederverwendbarkeit des VHDL-Codes zu erhöhen, sind sämtliche Komponenten modular aufgebaut. Die Verwendung generischer Konstanten innerhalb dieser Module und zur Definition ihrer Ein- und Ausgänge erlaubt eine schnelle Anpassung des Codes an sich ändernde Anforderungen wie Wortbreite und Taktfrequenz oder die einfache Übernahme in bzw. Anpassung an andere Projekte. Xilinx-spezifischer Code, der Hardwaremodule des Virtex-II direkt einbindet, ist durch chipunabhängige Schnittstellen so gekapselt, dass er im Falle einer Portierung auf andere Systeme schnell und unkompliziert ersetzt werden kann.

3.2.3 CPLD-VMEbus-Schnittstelle

Die CPLD-VMEbus-Schnittstelle wickelt die Kommunikation des FPGAs mit dem VMEbus ab. Dabei baut sie auf dem vom CPLD zur Verfügung gestellten Interface auf (Abschnitt 3.2.1). Da dieses Interface dem FPGA maximal vier Takte zur Bereitstellung von Daten auf dem VMEbus Zeit lässt, müssen diese schon vor dem Lesezugriff dem CPLD-VMEbus-Modul direkt zur Verfügung stehen. Eine Anforderung an das SDRAM-Modul dauert hierfür zu lange. Die Schnittstelle bietet Zugriff auf das Steuermodul (Testregister, Karten-ID, Arbeitsmodus, ...) und bewerkstelligt den Datentransfer zwischen SDRAM-FIFO und VMEbus zum Laden der Rohdaten sowie Abrufen der Ergebnisse.

3.2.4 SDRAM-FIFO

Das SDRAM-FIFO-Modul besteht aus sechs Untermodulen, und zwar

- der eigentlichen SDRAM-Schnittstelle,
- einem Adressgenerator für die Speicheradressverwaltung,
- sowie vier Datenbreitenkonvertern, die für eine Umsetzung der Ein- und Ausgangsdaten auf die 72 Bit Wortbreite des Speichermoduls sorgen.

Je nach Modus werden die Datenein- und -ausgänge der SDRAM-Schnittstelle über die jeweiligen Konverter mit dem Algorithmen-Block oder der CPLD-VMEbus-Schnittstelle verbunden.

SDRAM-Controller
Der SDRAM-Controller beherrscht nicht alle Funktionen des SDRAM-Standards, unterstützt aber Blocktransfers von ein bis acht Worten sowie den Page-Modus, d.h. eine Seite des Speichers wird solange ausgelesen, bis ein Abbruchbefehl vom Controller kommt. Die Datenein- und -ausgabe wird

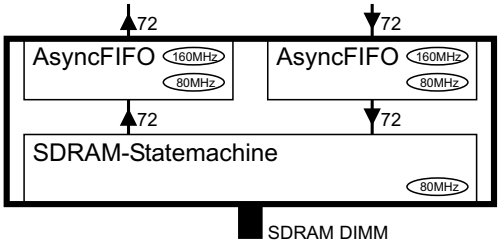


Abbildung 3.5: Schematische Darstellung des SDRAM-Controller-Moduls.

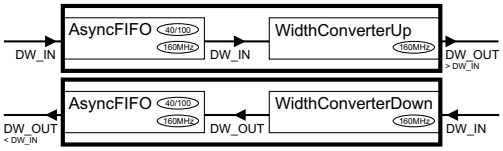


Abbildung 3.6: Schematische Darstellung der Datenbreitenkonverter-Module.

über asynchrone FIFOs zwischengepuffert. Dies ermöglicht es, das eigentliche Kontrollmodul, welches in Form einer Statemachine realisiert ist, und den SDRAM-Baustein mit einer eigenen Taktfrequenz laufen zu lassen. Abbildung 3.5 zeigt diesen Aufbau.

Datenbreitenkonverter

Die Datenbreitenkonverter sind nötig, um die verschiedenen Module effektiv miteinander koppeln zu können, d.h. möglichst wenig Bandbreite der jeweiligen Stufe zu vergeben. Zum Einen müssen die Wortbreiten der Module aufeinander umgesetzt werden, zum Anderen laufen sie mit unterschiedlichen Taktfrequenzen. Die Datenbreitenkonverter bestehen jeweils aus einem asynchronen FIFO und dem eigentlichen Konvertierungsmodul, welches entweder eine bestimmte Anzahl von Wörtern aus dem FIFO entnimmt und zu einem langen Datenwort zusammensetzt oder ein solches in kleine Teilstücke unterteilt und diese hintereinander in den FIFO schreibt. Der FIFO-Speicher sitzt also immer auf Seiten der kleineren Wortbreite (siehe Abb. 3.6).

Synchroner FIFO-Adressgenerator

Der synchrone FIFO-Adressgenerator erzeugt die SDRAM-Zugriffsadressen zum Schreiben und Lesen. Er setzt momentan am Ein- und Ausgang des SDRAM-Controller-Moduls an, so dass die Füllstandsanzeigen dieses Moduls durch die Verzögerungen die die Datenbreitenkonverter erzeugen, für Module außerhalb des SDRAM-FIFOs wertlos sind. Es muss daher beim Laden des SDRAMs zu Beginn eines Testdurchlaufs darauf geachtet werden, nicht zu viele Daten auf das Modul zu schieben. Daten, die nicht in den SDRAM-Speicher passen, verbleiben ohne Fehlermeldung in den Datenbreitenkonvertern und laufen nicht durch den zu testenden Algorithmus.

3.2.5 Taktfrequenzen der Hauptblöcke

Ziel ist es, das Algorithmusmodul konstant mit 14 Bit Datenwörtern bei 100 MHz aus dem SDRAM-Speicher zu versorgen, sowie das Ergebnis, welches im Fall des MWD-Algorithmus 34 Bit Wortbreite hat, dort zu speichern. Läuft der SDRAM-Speicher mit einem Takt von 80 MHz sowie im page-mode mit 128 Wörtern pro Zugriff, so erreicht man die erforderliche Datenrate. Die Datenbreitenkonverter laufen intern mit 160 MHz.

Eine überschlagsmäßige Berechnung dieser Werte unter Berücksichtigung der Timing-Parameter des SDRAM-Moduls erfolgt mittels einer Microsoft-Excel Arbeitsmappe.

3.2.6 Basismodule

Obige Hauptblöcke sind wiederum aus Untermodulen aufgebaut. Die Wichtigsten sind

- das asynchrone FIFO-Speichermodul *AsyncFIFO*, welches für die Schnittstelle zwischen den verschiedenen Frequenzdomänen benötigt wird, und

- die Taktfrequenzgeneratoren, welche aus dem Basistakt des FPGAs von 38,88 MHz die anderen benötigten Frequenzen z.B. für das SDRAM-Modul erzeugen.

AsyncFiFo

Das asynchrone FIFO-Speichermodul dient zur Schnittstellenpufferung zwischen zwei Frequenzdomänen. Ein- und Ausgang können mit unterschiedlichen Taktfrequenzen betrieben werden. Auch das AsyncFIFO-Modul besteht wieder aus Untermodulen: Einem Adressgenerator-Modul, welches die Speicherverwaltung abwickelt, sowie einem Speichermodul, welches den eigentlichen Speicherplatz bereitstellt. Von beiden Modulen gibt es jeweils zwei Ausführungen, welche mittels generischer Variablen im Programmcode ausgewählt werden können.

- **Adressgeneratoren**

  async_fifo_addr_gen_fast ist die schnellere Variante dieses Moduls. Sie ist in der Lage, neben den Adressen für die Speicheransteuerung die Flags *empty*, *almost empty*, *almost full* und *full* zu erzeugen. Das Modul arbeitet mit vier Zählern sowie Komparatoren.

async_fifo_addr_gen_levflags erzeugt neben den oben genannten Adressen und Flags noch zusätzlich eine Füllstandsanzeige in 2er-Potenzen. Dafür wird unter anderem eine asynchrone 10 Bit-Subtraktion durchgeführt, so dass dieses Modul nur bei langsamen Taktfrequenzen zuverlässig funktioniert.

- **Speichermodule**

  async_fifo_memory_selectRAM verwendet Virtex-II selectRAM-Blöcke als Speicher für den FIFO. Die Adressbreite des Moduls beträgt maximal 9 Bit, die Datenwortbreite kann beliebig sein, das Modul ist aber von der Art der Einbindung der selectRAM-Blöcke vor allem für große Wortbreiten, z.B. die 72 Bit des SDRAM-Moduls, ausgelegt.

  async_fifo_memory_small erzeugt ein Registerarray zur Speicherung und eignet sich daher vor allem für schmalbandige und kurze FIFOs.

Taktfrequenzgeneratoren

Die Taktfrequenzgeneratoren kapseln die Clock-Management Schaltkreise des Virtex-II. Da der FPGA die Taktfrequenz für das SDRAM-Modul erzeugt, ist hier eine aufwändigere Konstruktion notwendig (*clock_gen2x_ext*). Details zu den Taktfrequenzgeneratoren und den vielfältigen Möglichkeiten, die der Virtex-II Chip bietet, finden sich in der Virtex-II Platform FPGA User Guide [3] und Virtex-II Platform FPGA Advance Product Specification [4].

3.2.7 Momentaner Status

Auf dem FPGA getestet wurden bisher das CPLD-VMEbus-Interface sowie das SDRAM-Controller-Modul. Damit war es möglich, Daten über den VMEbus in den SDRAM-Speicher zu schreiben und wieder zu lesen. Die Datenbreitenkonverter konnten bisher nur in der Simulation erprobt werden.

3.3 MARS-Daten

Die zum Testen der Algorithmen verwendeten Detektordaten stammen vom MARS-Projekt.

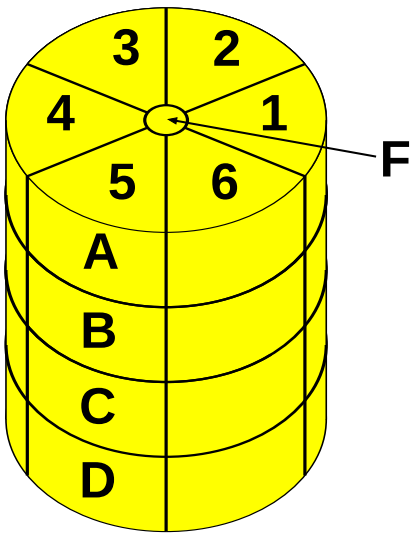


Abbildung 3.7: Schematische Darstellung des MARS-Detektors und seiner Segmente.

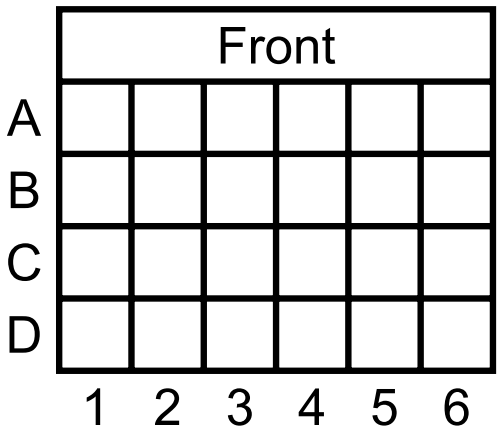


Abbildung 3.8: Zweidimensionale Anordnung der 25 Segmente des MARS-Detektors.

3.3.1 Das MARS-Projekt

Das MARS-Projekt [2] (MARS: **M**ini **A**rray di **R**ivelatori **S**egmentati bzw. **M**ini **AR**ray of **S**egmented detectors) dient der Erforschung und Entwicklung neuer (digitaler) Technologien im Einsatz und Umgang mit hochsegmentierten Germaniumdetektoren für die Gamma-Spektroskopie. Besonderes Augenmerk liegt dabei auf dem Gamma-Ray Tracking. Der Prototyp-Detektor ist ein 25-fach segmentierter halbkoaxialer Germaniumdetektor. Die Segmente sind dabei in vier Ringen mit jeweils sechs Abschnitten angeordnet. Zusätzlich befindet sich eine weitere Elektrode auf der geschlossenen Frontseite des Detektors. Mit dem Zentralkontakt sind also insgesamt 26 Kanäle auszulesen und zu digitalisieren. Dies geschieht momentan mit Hilfe von sieben digitalen 4-Kanal-Oszilloskopen von LeCroy, da sich eine spezielle Ausleseelektonik noch in der Entwicklung befindet. Die Oszilloskope können die Daten mit bis zu 200 MHz und 8 Bit Auflösung digitalisieren und erlauben eine Zwischenspeicherung von 100 kByte pro Kanal [32].

3.3.2 Die MARS-Daten

Für den Test der Algorithmen standen zwei verschiedene Datensätzen zur Verfügung.

- Datensatz eins besteht aus 20.000 Einzelereignissen, gemessen mit einer ^{137}Cs-Quelle. Getriggert wurde auf Ereignisse mit der vollen Gamma-Energie von 662 keV am Zentralkontakt. Für jedes dieser Ereignisse wurde ein 10 μs langer Bereich um den Triggerzeitpunkt herum auf allen Kanälen aufgezeichnet. Digitalisiert wurde mit 200 MHz Abtastfrequenz und 8 Bit Datenbreite. Abbildung 3.10 zeigt alle Kanäle eines Beispielereignisses aus diesem Datensatz. Aufgetragen ist jeweils der Ausgangswert des Vorverstärker-ADCs über der Zeit, wie auch für alle anderen Vorverstärkersignalabbildungen. Jedes 50ste Datensample enthält kein Ereignis (insgesamt etwa 400). Um auf allen Segmenten Signale in größerer Anzahl zu erhalten, wurde die Quelle während der Messung um den Detektor herum bewegt.

- Datensatz zwei wurde mit einer ^{60}Co-Quelle gewonnen, die direkt auf der Frontseite des Detektors angebracht war. Die mittlere Ereignisrate auf dem Zentralkontakt beträgt 38 kHz. Aufgezeichnet wurden parallel alle 26 Kanäle mit 100 MHz Abtastfrequenz. Jeder Abschnitt ist 1 ms lang (entspricht der maximalen Speicherkapazität des Oszilloskops bei 100 MHz und 8 Bit), der gesamte Datensatz beinhaltet 54 solcher Abschnitte - d.h. insgesamt etwa 2.000 Ereignisse unterschiedlicher Energie.

Zum Ausgleich der Nichtlinearität der ADCs [31] existierten Umsetzungstabellen für alle 26 Kanäle. Die Abweichung des Eingangswertes als Funktion des Ausgangskanals ist exemplarisch für Segment A1 in Abbildung 3.9 dargestellt.

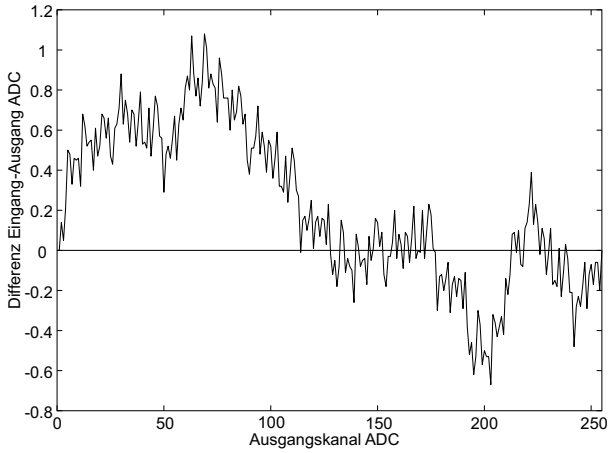


Abbildung 3.9: Abweichung des Eingangssignals vom Ausgangssignal des ADCs für Segment A1.

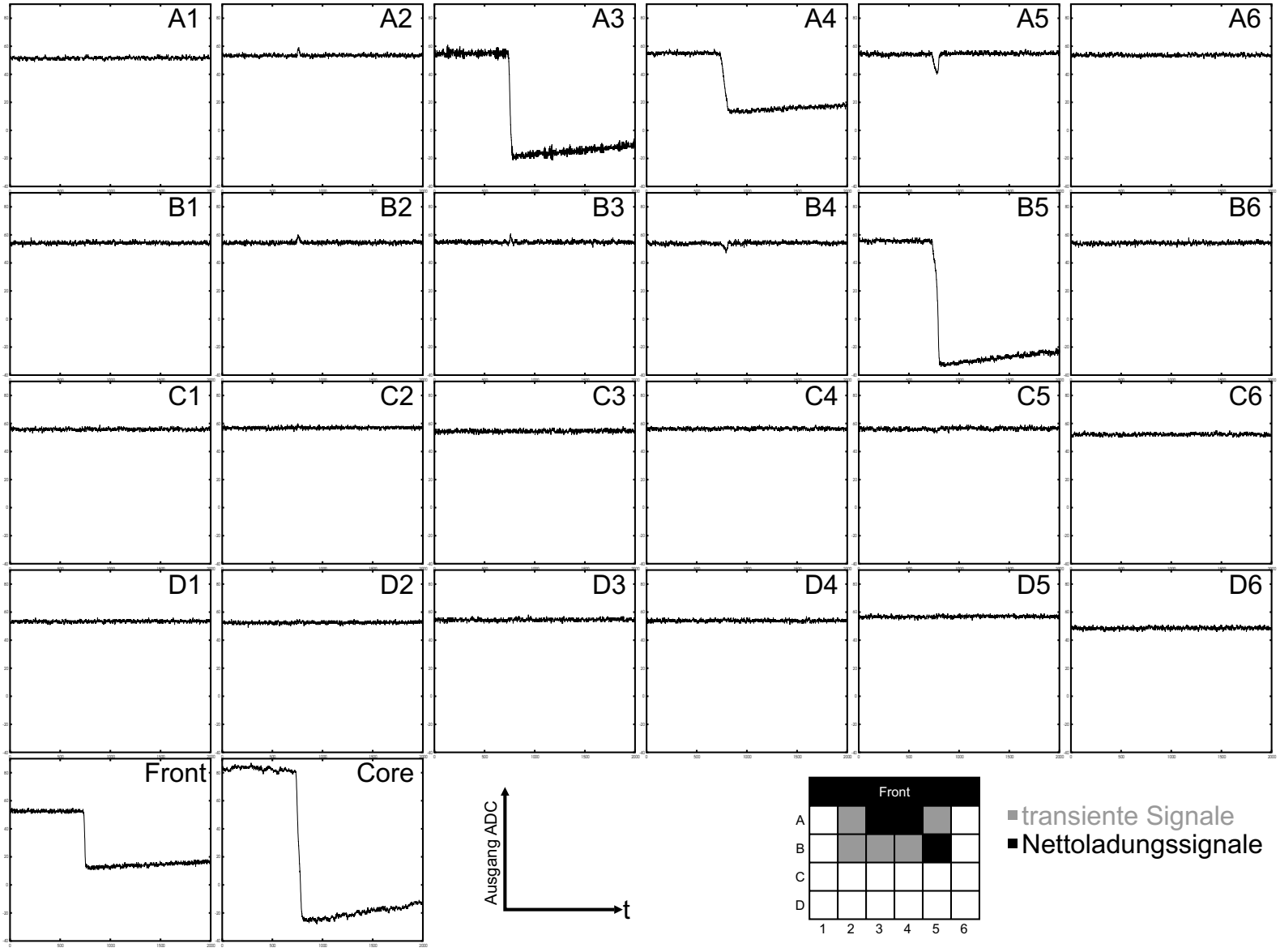


Abbildung 3.10: Alle 26 Kanäle für ein Ereignis aus dem MARS-Datensatz 1 (200 MHz, 8 Bit, 10 μs).

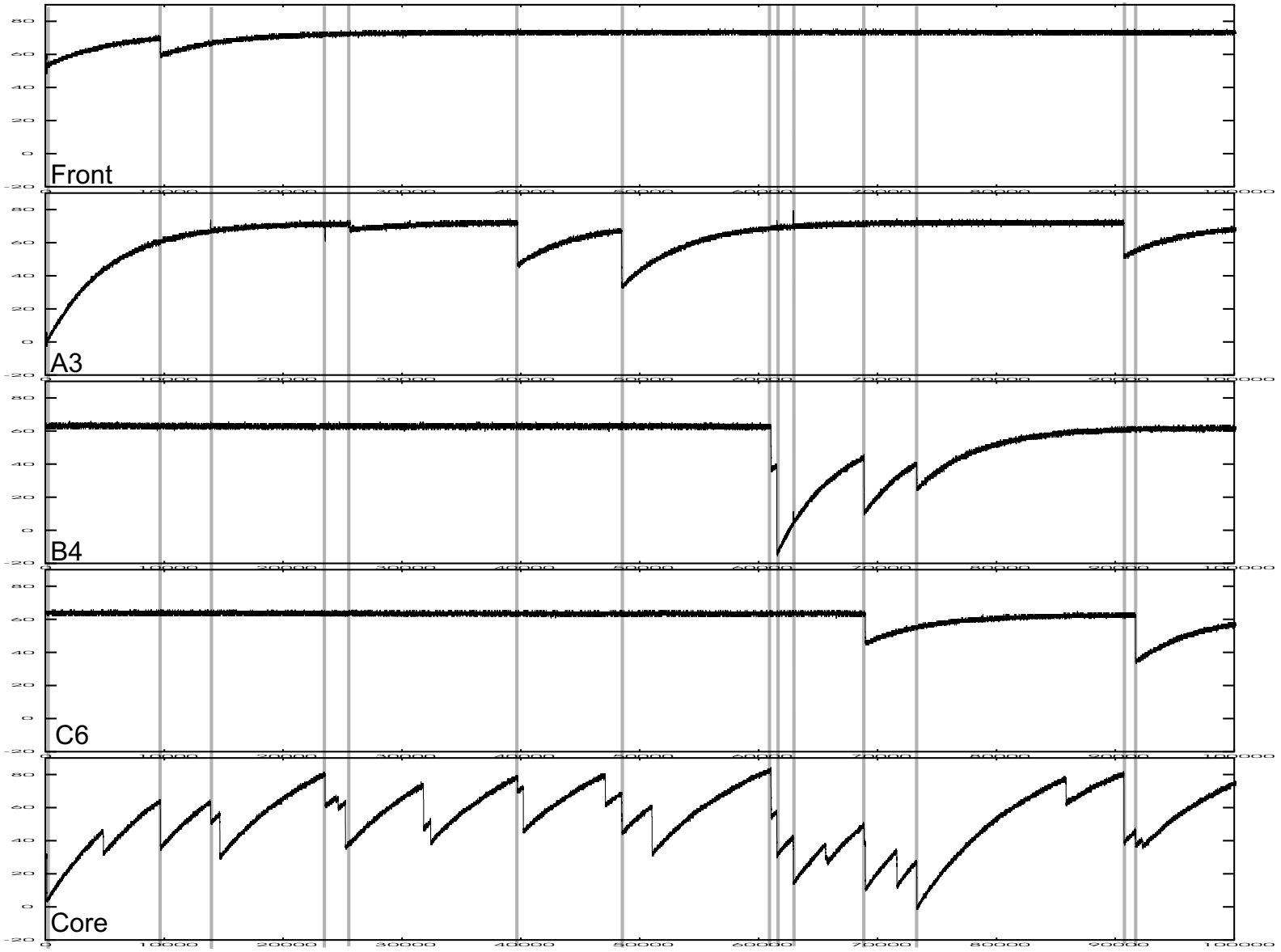


Abbildung 3.11: Ausgewählte Segmente eines Abschnitts aus dem MARS-Datensatz 2 (100 MHz, 8 Bit, 1 ms).

Kapitel 4

Ergebnisse

4.1 Implementierung

4.1.1 C++

Prinzipielle Beschreibungen der hier verwendeten Klassen sowie der Gesamtstruktur finden sich in Anhang E.

MWD

Kernalgorithmus
Die Differenzengleichung für den MWD-Algorithmus hat die Form (Glg. 2.1)

$$y[n] = x[n] + (1-k) \cdot \sum_{m=1}^{M} x[n-m] - x[n-M] \ .$$

Eine Reduktion der benötigten Rechenschritte ergibt sich aus der rekursiven Repräsentation

$$y[n] = y[n-1] + x[n] - kx[n-1] - x[n-M] - kx[n-M-1] \ . \qquad (4.1)$$

Die konkrete Implementation erfolgt in einer Mischform dieser beiden Gleichungen. Basis ist die nicht-rekursive Form Glg. 2.1, wobei die Summe jedoch rekursiv berechnet wird

$$\begin{aligned} \mathrm{SUM}[n] &= \mathrm{SUM}[n-1] + (1-k)(x[n] - x[n-M]) \text{ und} \\ y[n] &= x[n] - x[n-M] + \mathrm{SUM}[n-1] \ . \end{aligned} \qquad (4.2)$$

Dadurch reduziert sich die Anzahl der benötigten Rechenschritte pro Takt auf vier Additionen bzw. Subtraktionen und eine Multiplikation.

Der C++-MWD-Kernalgorithmus ist als PSAFilter-Klasse ausgeführt. Als digitaler Filter mit endlicher Impulsantwort ist er von der Klasse `PSAFilterFIR` abgeleitet. Der Konstruktor der Klasse bekommt die Fenstergröße und den Exponenten $\alpha = -(f_{sampling} \cdot \tau)^{-1}$ übergeben. Diese werden in den privaten Variablen `window_size` und `response_coefficient` $(= (1 - \exp(\alpha))$ abgelegt.

Die Funktion `run()` der `PSAFilter_MWD`-Klasse hat folgende Form:

```
template <class gtype>
gtype PSAFilter_MWD<gtype>::run(const gtype& dv) const {
    this->buffer_data.PullPush(dv);
    gtype output = dv \\
        + (this->response_coefficient * this->input_sum) \\
        - this->buffer_data.GetConstReferenceAt(this->window_size);
    this->input_sum = this->input_sum + dv \\
        - this->buffer_data.GetConstReferenceAt(this->window_size);
    PSAFilterObject_1t1<gtype, gtype>::run();
    return output;
}
```

`buffer_data` ist eine geschützte Variable der Klasse `PSAFilterFIR` und dient der Aufnahme der letzten M Datenwerte. `input_sum` wird zur Pufferung der Summe zwischen zwei Schritten verwendet, so dass in jedem Schritt nur noch die Differenz $x[n] - x[n - M]$ addiert werden muss. Die `run()`-Funktion des FIR-Filters wird umgangen und stattdessen direkt die darunter liegende Funktion des `PSAFilterObject_1t1` aufgerufen, um den Schrittzähler der `PSAFilterObject`-Klasse zu erhöhen.

Mittelung
Die Zusammenfassung mit dem Mittelungsfilter `PSAFilter_Average` erfolgt über ein `PSAFilterChainObject`-Objekt. Der Mittelungsfilter `PSAFilter_Average` ist eine aus dem Summationsfilter `PSAFilter_Sum` und `PSAFilter_Gain` zusammengesetzte Filterkette.

Offset
Zusätzlich enthält diese Filterkette vor dem MWD-Filter einen Additions- bzw. Subtraktionsfilter `PSAFilter_RemoveOffset` zur Entfernung eines eventuellen Offsets der Daten. Prinzipiell ist ein Offset der Nulllinie irrelevant für den MWD-Algorithmus, da der erste Schritt, die Entfaltung, sich wie eine Differentiation auswirkt. Der Speicher der M vergangenen Werte ist allerdings zu Beginn mit Nullen gefüllt, so dass ein von Null verschiedener Wert des ersten Datenpunktes wie eine Stufe auf dem Signal behandelt wird,

also einen MWD-Puls erzeugt. Dadurch werden die ersten M Werte des ungemittelten MWD-Ausgangssignales verfälscht. Befindet sich der Algorithmus in einem kontinuierlichen Betrieb, so stellt dies kein Problem dar. Im verwendeten MARS-Datensatz 1 befindet sich der zu untersuchende Puls jeweils bei etwa 4 μs, so dass ohne eine Entfernung des Offsets Fenster mit einer Länge größer als 4 μs falsche Ergebnisse liefern. Dem Offset-Filter kann eine Länge N übergeben werden, über die dann zunächst gemittelt wird, um den Offset des Datensamples genauer zu bestimmen.

`PSAFilter_RemoveOffset` demonstriert einen Teil der Möglichkeiten, die die `PSAFilterChainObject`-Klasse bietet. Diese Filterkette verwendet zwei Filterklassen, die während der Laufzeit ausgetauscht werden. Zunächst wird für die ersten N Samples der Mittelwert mit dem Filter `PSAFilter_Mean`[1] berechnet, im Anschluss das Ergebnis dieser Mittelung für die Initialisierung des Additionsfilters `PSAFilter_Shift` verwendet und diese beiden Filter ausgetauscht. Für den Anwender der `PSAFilter_RemoveOffset`-Klasse geschieht dies völlig transparent.

Zwischen Offsetentfernung und MWD-Filter befindet sich noch ein `PSAFilter_Cut`-Filter, der den Datenstrom während der Mittelungsphase des Offset-Filters maskiert.

`PSAFilter_Average` und `PSAFilter_Mean` unterscheiden sich insofern, als dass ersterer die Mittelung nur über ein Fenster vornimmt, während zweiterer den Mittelwert der seit dem letzten Reset angefallenen Daten berechnet.

Abbildung 4.1 stellt diese Verknüpfungen der Filter grafisch dar.

SCC

`PSAFilter_SCC_FallingEdge` und `PSAFilter_SCC_RisingEdge` sind von der Klasse `PSAFilter_SCCObject` abgeleitet, welche wiederum auf `PSAFilter_CCObject` aufbaut.

`PSAFilter_CCObject` (CC steht für condition counter) stellt die Basisfunktionalität für den Vergleich eines Bereichs mit einem Referenzwert zur Verfügung. Der Filter definiert die virtuelle Funktion `CalcLocalSC()`, welcher der Referenzwert sowie zu vergleichende Werte vor und hinter diesem übergeben werden. Diese Funktion muss von den abgeleiteten Klassen implementiert werden. Die `run()`-Funktion des Filters ruft diese Funktion für den gesamten zu betrachtenden Bereich auf und addiert die Rückgabewerte, erstellt also die gewichtete Summe.

`PSAFilter_SCCObject` fügt die Integrationsstufe im Anschluss an die Bildung der gewichteten Summe hinzu.

[1] PSAFilter_Mean ist seinerseits aus PSAFilter_Integrate und PSAFilter_InvCycleGain zusammengesetzt.

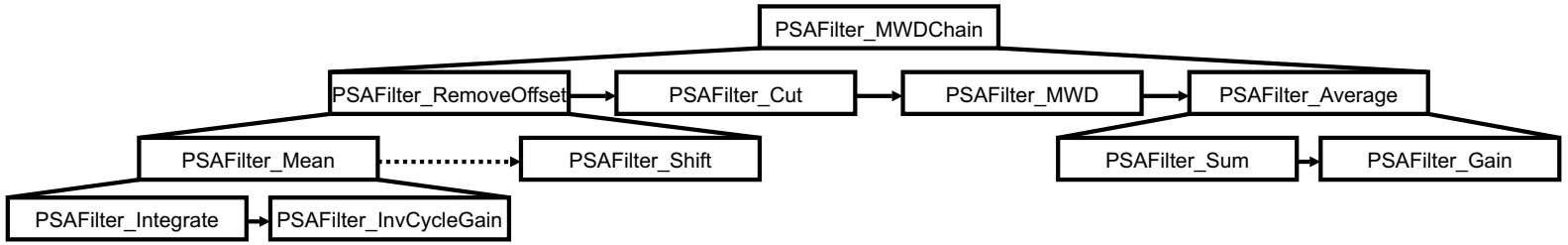


Abbildung 4.1: Struktur des MWDChain-Filters. In Anlehnung an die Modulbauweise des VHDL-Codes sind auch hier die Einzelschritte als eigene Klassen ausgeführt.

In PSAFilter_SCC_FallingEdge und PSAFilter_SCC_RisingEdge wird CalcLocalSC() mit den gewünschten Gewichtungsfaktoren implementiert. Als Beispiele seien hier die CalcLocalSC()-Funktion des Triggers für abfallende Flanken angeführt sowie die run()-Funktionen von PSAFilter_SCCObject und PSAFilter_CCObject:

```
template <class gtype>
int PSAFilter_SCC_FallingEdge<gtype>::CalcLocalSC( \\
        const gtype& middle, \\
        const gtype& former, \\
        const gtype& later) const {
     return ((middle > later) - (middle < former));
}

template <class gtype>
int PSAFilter_SCCObject<gtype>::run(const gtype& dv) const {
    this->acc += PSAFilter_CCObject<gtype>::run(dv);
    return this->acc;
}

template <class gtype>
int PSAFilter_CCObject<gtype>::run(const gtype& dv) const {
    unsigned int size = this->GetSize();
    int counter = 0;
    data_buffer.PullPush(dv);
    for (unsigned int i = 1; i <= size; ++i) {
         counter += this->CalcLocalSC( \\
            this->data_buffer.GetConstReferenceAt(size), \\
            this->data_buffer.GetConstReferenceAt(size + i), \\
            this->data_buffer.GetConstReferenceAt(size - i));
    }
    return counter;
}
```

4.1.2 VHDL

MWD

Kernalgorithmus

Der VHDL-MWD-Algorithmus basiert wie der C++-Code auf der Mischform (Glg. 4.2) aus rekursiver und nicht-rekursiver Repräsentation des Algorithmus. Bei der Implementierung in Hardware gibt es jedoch einige zusätzliche Punkte zu beachten.

- Die Ausführung aller Rechenoperationen (vier Additionen, eine Multiplikation) hintereinander in einem Takt ist bei 100 MHz und 14 Bit Datenwortbreite nicht möglich. Daher muss die Berechnung auf mehrere Takte verteilt werden in Form einer Pipelinestruktur in der die Rechenoperationen parallel für die aufeinander folgenden Datenwerte vorgenommen werden. Die dadurch auftretende Verzögerung des Ausgangswertes beträgt 4 Takte.

- Der FPGA beherrscht ausschließlich Integerrechenoperationen. Eine Abbildung des Wertes (1-k) auf ganzzahlige Werte erzwingt daher eine Erweiterung der Datenwortbreite.

  Fordert man für α ($\approx 10^{-4}$) bei der Umrechnung eine maximale Abweichung von 1%, so folgt daraus für $(1-k) = (1-\exp(-\alpha)) \approx 10^{-4}$ eine Genauigkeit von 10^{-6}. Um also Fließkommawerte mit dieser Genauigkeit auf den Ganzzahlbereich abzubilden, muss man eben diese Werte mit 10^6 multiplizieren. Dies geschieht am einfachsten durch Anhängen zusätzlicher Leerbits an das Ende des Datenwortes bzw. eine Linksverschiebung der Datenbits. Diese Operation entspricht der Multiplikation mit Potenzen von zwei.

  $$2^p \cdot y[n] = 2^p \cdot (x[n] - x[n-M]) + \underbrace{2^p \cdot (1-k)}_{\approx 100} \cdot \sum_{m=1}^{M} x[n-m]$$

  $2^{20} = 1048576$, also $p = 20$, kommt 10^6 dabei am nächsten. $2^p \cdot (1-k)$ liegt dann in der Größenordnung von 10^2 bis 10^3, einem Wertebereich, der sich mit 8-10 Bit Zahlen abdecken lässt. Diesen Wert repräsentiert im VHDL-Programmcode die Variable EXP_DECAY. Für einen Vorverstärker mit einer Zeitkonstanten von 50 μs und bei einer Abtastfrequenz $f_{sampling}$ von 100 MHz erhält man für EXP_DECAY einen Wert im Bereich von 200, d.h. EXP_DECAY kann auf 8 Bit beschränkt werden.

  Die Datenwortbreite des Ergebnisses wird dadurch beträchtlich vergrößert, im obigen Fall um 20 Bit. Das MWD-Ausgangssignal hat also dann bei einer Eingangsbreite von 14 Bit insgesamt 34 Bit.

Abbildung 4.2 zeigt die daraus resultierende Struktur des MWD-Algorithmus.

Das VHDL-MWD-Modul `psa_filter_mwd` ist aus zwei Untermodulen aufgebaut:

- Das Algorithmusmodul beinhaltet die eigentlichen Rechenoperationen. Es hat zwei Eingänge für den aktuellen Datenwert x[n] und den um M

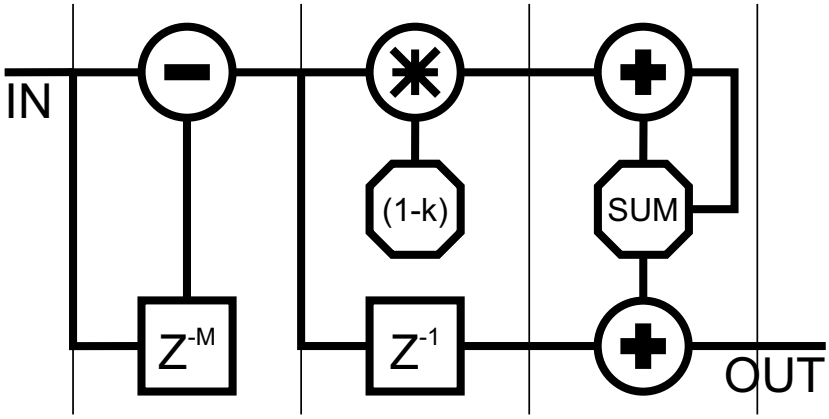


Abbildung 4.2: Schema der Hardwareimplementation des MWD-Algorithmus.

Takte verzögerten Datenwert x[n-M]. Das Ausgangssignal ist je nach Implementation drei bis vier Takte verzögert. Es gibt zwei verschiedene Implementationen des MWD-Algorithmusmodules.

Zum einen das Modul `psa_filter_mwd_algorithm_direct`, welches alle fünf Rechenoperationen in einem Prozess zusammenfasst. Der Quelltext dieses Moduls findet sich in Anhang F.

Das zweite Modul, `psa_filter_mwd_algorithm`, ist aus weiteren Untermodulen zusammengesetzt. Das Summations-Algorithmus-Modul `psa_filter_sum_algorithm` berechnet die aktuelle Summe, ein Multiplikationsmodul kapselt die Multiplikation. `psa_filter_mwd_algorithm` implementiert dieses Modul zweifach, da es die Werte x[n] und x[n-M], anders als in Gleichung 4.2 dargestellt, vor der Subtraktion multipliziert. Je nach Chipresourcen kann es von Vorteil sein, die Multiplikation mit einer möglichst geringen Datenwortbreite durchzuführen[2].

- Das Verzögerungsmodul dient der Verzögerung der Eingangssignale um M Takte. Es hat einen Eingang für x[n] und einen Ausgang für x[n-M].

Die Auswahl des zu verwendenden Algorithmusmoduls erfolgt mittels einer generischen Variablen bei der Einbindung von `psa_filter_mwd`.

Die Programmierung erlaubt es, die Fensterbreite WS und den Multiplikationsfaktor EXP_DECAY während der Laufzeit zu verändern. Eine Veränderung der Fensterbreite hat jedoch einen Reset des Algorithmus zur Folge, da

[2]Die Addition oder auch Subtraktion zweier vorzeichenbewehrter Werte mit einer Breite von DW führt zu einem Ausgangswert der Breite DW+1.

die Summenbildung neu gestartet werden muss um ein korrektes Ergebnis zu erzielen.

Mittelung

Wegen der fehlenden Gleitkommaoperationen wird die Mittelung durch eine Summenbildung ersetzt. Dies führt zu einer weiteren Datenwortverbreiterung. Das Summationsmodul ist wie das MWD-Modul aus zwei Untermodulen zusammengesetzt, dem Modul `psa_filter_sum_algorithm` und einem Verzögerungsmodul. Es bietet ebenfalls die Möglichkeit, die Summationslänge während der Laufzeit neu zu setzten, wobei auch hier die bisherige Summe verloren geht.

MWD und die anschließende Summation werden durch `psa_filter_mwdchain2` zu einem Block zusammengefasst. Eine Entfernung des Offsets findet hier nicht statt, so dass die ersten M + L Ausgangswerte nicht verwendbar sind.

Verzögerungsmodule

Auch die Verzögerungsmodule setzten sich wieder aus zwei Untermodulen zusammen, einem Speichermodul und einem Adressgenerator der die Schreib- und Leseadresse für das Speichermodul generiert. Die Speichermodule unterscheiden sich in der Art des verwendeten Speichers. Das Modul für geringe Datenmengen implementiert den Speicher in Form von Registern, die Module für Verzögerungen über mehrere hundert Takte und große Datenwortbreiten bedienen sich des BlockRAMs der Virtex-II FPGAs. Die Länge der Verzögerung kann während der Laufzeit angepasst werden.

Der Resourcenverbrauch des MWD auf dem Virtex-II 3000 ist relativ gering. Die "direct"-Implementierung benötigt drei der 96 vorhandenen BlockRAMs sowie einen der 96 dedizierten 18x18 Bit Multiplikatoren. Die Anzahl der verwendeten Logikelemente liegt unterhalb von 3%. Der größte Resourcenverbrauch betrifft die I/O-Pins des Chips. 14 Eingangsbit und 44 Ausgangsbit der Daten sowie die Eingänge zum setzen der Parameter nehmen hier bis zu 20% der vorhandenen Pins in Anspruch.

SCC

Kernalgorithmus

Das SCC-Modul setzt sich aus zwei Blöcken zusammen, dem Vergleichsblock und einem Bitzähler. Der Vergleichsblock führt die M Vergleiche für das vordere Fenster innerhalb eines Taktes parallel durch. Daneben verzögert er diese Vergleichsergebnisse, da sie für das hintere Fenster noch einmal in

negierter Form verwendet werden können. Abbildung 4.3 zeigt diesen Aufbau.

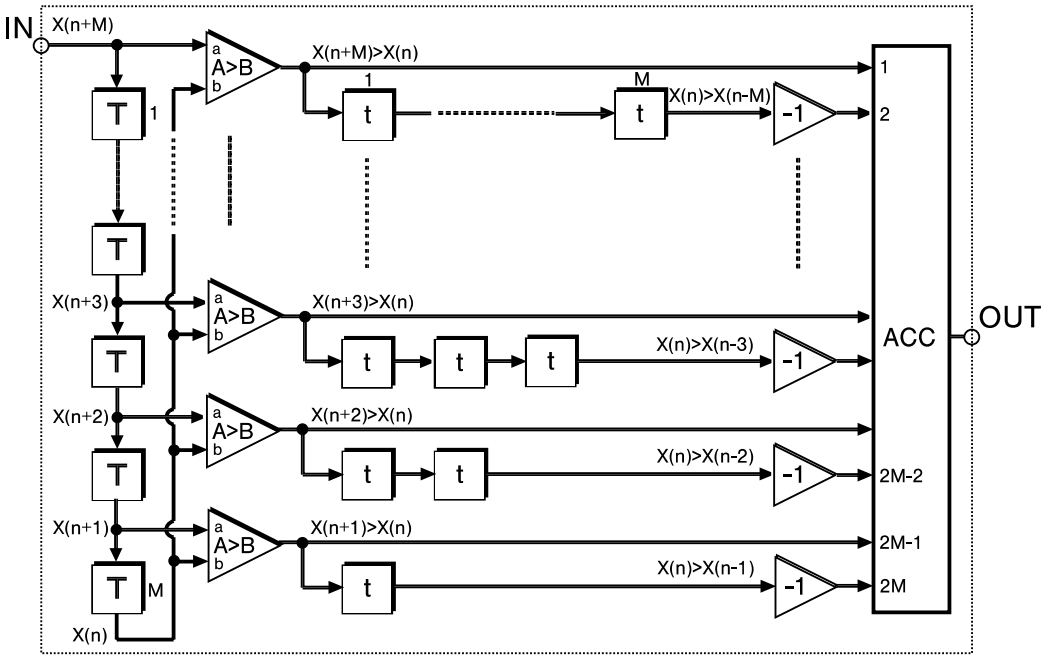


Abbildung 4.3: Darstellung des SCC-Kernalgorithmus. (Bildquelle: [11], W. Gast)

Das Problem der Hardwareumsetzung des SCC-Triggers besteht in der Addition der gewichteten Vergleichsergebnisse, in Abbildung 4.3 geschieht dies im Modul `acc`. Innerhalb jeden Taktes müssen 2M Werte addiert werden. Der Algorithmus soll in der Lage sein, auch Fenster mit Breiten $M \geq 50$ zu verarbeiten. Im Fall des SCC-Triggers lässt sich die Summierung der 2M Werte auf ein Abzählen von gesetzten Bits in zwei M-Bit-breiten Datenwörtern reduzieren. Da dies während eines Taktes nicht möglich ist, wird das Abzählen auf 4 Bit Abschnitte beschränkt, welche parallel ausgewertet werden. Die für diese Bereiche gleichzeitig gewonnenen Ergebnisse werden mittels einer Additionspyramide zusammengeführt. Abbildung 4.4 veranschaulicht diese Konstruktion. Das BitCounter-Modul `psa_bitcounter` erstellt diese Pyramide aufgrund der vorgegebenen Fensterbreite automatisch. Der Quelltext des Moduls findet sich in Anhang F.

Im Anschluss an das Abzählen der Bits für jeden Takt folgt noch eine Integration dieser Ergebnisse in einem externen Integrationsfilter.

Aufgrund der Zerlegung des Bitzählers in die Additionspyramide ist die maximale Taktfrequenz des SCC-Kernalgorithmus unabhängig von der

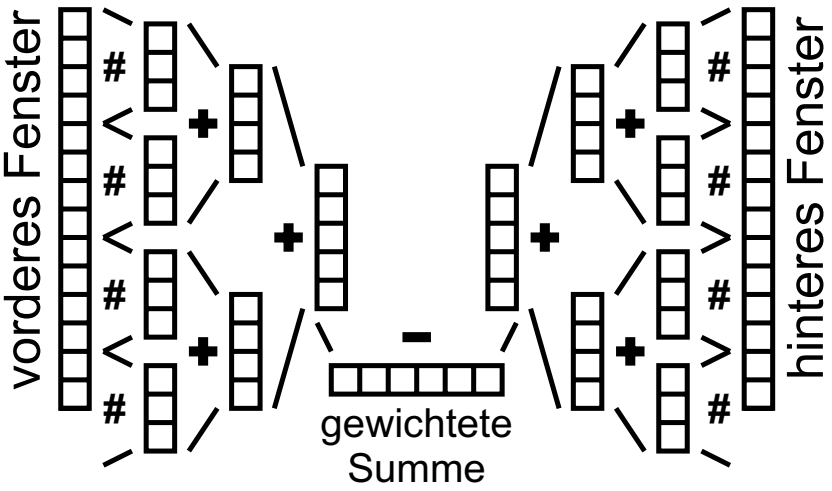


Abbildung 4.4: Additionspyramide des Bitcounter-Moduls. Links und rechts werden jeweils die Vergleichsergebnisse für das vordere und das hintere Fenster eingespeist, das Ergebnis liegt in diesem Fall vier Takte später vor.

gewählten Fensterbreite. Allerdings korreliert der Resourcenverbrauch auf dem FPGA stark mit der gesetzten Fensterbreite.

Mittelung

Als Mittelungsfilter kommt wieder das im MWD-Abschnitt vorgestellte Summationsmodul zum Einsatz.

Der VHDL-SCC-Kernalgorithmus verbraucht bei einer Fensterbreite von 50 Takten und einer Datenwortbreite von 14 Bit etwa 123000. Die durch den Integrator vereinnahmten Logikelemente sind dabei vernachlässigbar ($<$ 1%), gleiches gilt (bis auf das verwendete BlockRAM) auch für den Summationsfilter zur Mittelung.

Simulationen

Alle Tests der VHDL-Algorithmen wurden als Simulation durchgeführt, da die Programmierung der Testumgebung auf der VMEbus-FPGA-Karte noch nicht vollständig einsatzbereit ist. Dabei kamen zwei Arten von Simulation zum Einsatz, zum Einen die reine Funktionssimulation, die Signallaufzeiten unbeachtet lässt, zum Anderen eine Simulation der für den FPGA synthetisierten Schaltung unter Einbeziehnug der Timingparameter des Virtex-II FPGAs. Als problematisch erwies sich dabei ein Fehler in der Timingsimulation des Virtex-II BlockRAMs. Hier kommt es sporadisch zu einem Datenverlust. Dies konnte aber aufgrund der hohen Modularität der Programmierung

durch getrennte Simulation von Algorithmus und Datenmodul umgangen werden. Desweiteren machte sich die begrenzte Lizenz der Simulationssoftware in teilweise sehr langen Simulationszeiten bemerkbar (10 min, um 20 μs Laufzeit zu simulieren).

Synthese und Simulationen wurden für einen Virtex-II 3000 mit Speedgrade -4 durchgeführt, dem FPGA, der sich auf der VMEbus-FPGA-Karte befindet. Die Überprüfung der Simulationsergebnisse erfolgt durch einen Vergleich der Ausgangssignale von VHDL-Simulation und C++-Algorithmus. Die ebenfalls in VHDL programmierten Simulationsumgebung liest dabei Daten aus einer Datei und speichert die Ausgabewerte des simulierten Moduls. Dabei ist teilweise ein Abschneiden der niederwertigsten Bits unumgänglich, da ein Integerwert maximal 32 Bit umfasst.

MWD

Die Synthese spezifiziert das `psa_filter_mwdchain2`-Moduls mit einer maximalen Taktfrequenz von 94 MHz ("direct") bzw. 83 MHz ("modular") für einen FPGA mit Speedgrade -4. In der nächst schnelleren Stufe (-5) werden die 100 MHz für beide Ausführungen knapp überschritten, für Speedgrade -6 bei weitem (140 MHz "direct"). Eine Timingsimulation des Gesamtmoduls ist aufgrund des oben beschriebenen Fehlers der BlockRAM-Simulation nicht möglich, so dass nur die Einzelmodule zeitkritisch simuliert werden konnten. Als langsamstes Element erwies sich die die Mittelung ersetzende Summation mit einer Datenwortbreite von 34 Bit. Hier traten für Speedgrade -4 und 100 MHz Bitfehler für das Vorzeichenbit auf. Der MWD-Kernalgorithmus lieferte für Funktions- und Timingsimulation die exakt gleichen Ergebnisse.

SCC

Das SCC-Modul `psa_filter_scc_falling` wird von der Synthesesoftware mit 190 MHz spezifiziert und liefert für die Timingsimulation mit 100 MHz dasselbe Ergebnis wie für die reine Funktionssimulation.

Zusammenfassend kann gesagt werden, dass die beiden Algorithmen auf einem Virtex-II implementiert werden können. Der auf der VMEbus-FPGA-vorhandene Chip mit Speedgrade -4 ist für den MWD-Algorithmus mit 14 Bit wohl etwas zu langsam, eine schnellere Version (Speedgrade -5 oder -6) des FPGAs bewältigt den Algorithmus aber mit der angestrebten Datenbreite und Taktfrequenz.

4.1.3 Vergleich C++ $\Leftrightarrow$ VHDL

MWD

Die aus den VHDL-Simulationen erhaltenen Integer-Ergebnisse müssen zunächst auf die C++-Gleitkommawerte normiert werden. Der Normierungsfaktor setzt sich dabei aus der Mittelungs- bzw. Summenlänge und der Datenwortbreitenerweiterung des MWD-Kernalgorithmus zusammen. Außerdem muss das eventuell nötige Abschneiden der niederwertigsten Bits vor der Speicherung durch die Simulationsumgebung berücksichtigt werden. Um den MWD-Algorithmus in C++ und VHDL vergleichen zu können, wurden beide auf die selben Datensamples angewendet. Um Abweichungen aufgrund unterschiedlicher Vorverstärkerzeitkonstanten auszuschließen, wurde die dem C++-Programm übergebene Zeitkonstante der des VHDL-Moduls angepasst. Abbildung 4.5 zeigt das Ergebnis eines solchen Vergleichs.

Die Ergebnisse der beiden Programme unterscheiden sich praktisch nicht. Für Werte nahe Null ergeben sich größere Abweichungen, da hier das Entfernen der niederwertigsten Bits vor der Speicherung durch die VHDL-Testumgebung einen großen relativen Fehler erzeugt.

SCC

Die Implementierung in C++ wie auch in VHDL liefern exakt das gleiche Ergebnis, da der Algorithmus keinerlei Gleitkommaoperationen erfordert.

4.2 Algorithmen

4.2.1 Vorbereitende Analyse der MARS-Daten

Zur Anwendung des MWD-Algorithmus auf die MARS-Daten müssen zunächst die Integrationszeiten τ der 26 Vorverstärker bestimmt werden. Dies geschieht anhand der 20.000 Ereignisse aus Datensatz eins mittels einiger C-Routinen.

Die Integrationszeit wird aus dem exponentiellen Abfall der Nettoladungssignale gewonnen. In einem ersten Schritt werden für jeden Kanal diejenigen Datensamples herausgesucht, die ein solches enthalten. Dafür werden der Anfang und das Ende des Datensamples gemittelt. Die Differenz dieser beiden Mittelwerte dient als Auswahlkriterium. Anhand Abbildung 3.10 ist erkennbar, dass diese sehr einfache Methode für diesen Datensatz ausreichend ist, um Nettoladungssignale zu finden, die deutlich über dem Rauschen liegen.

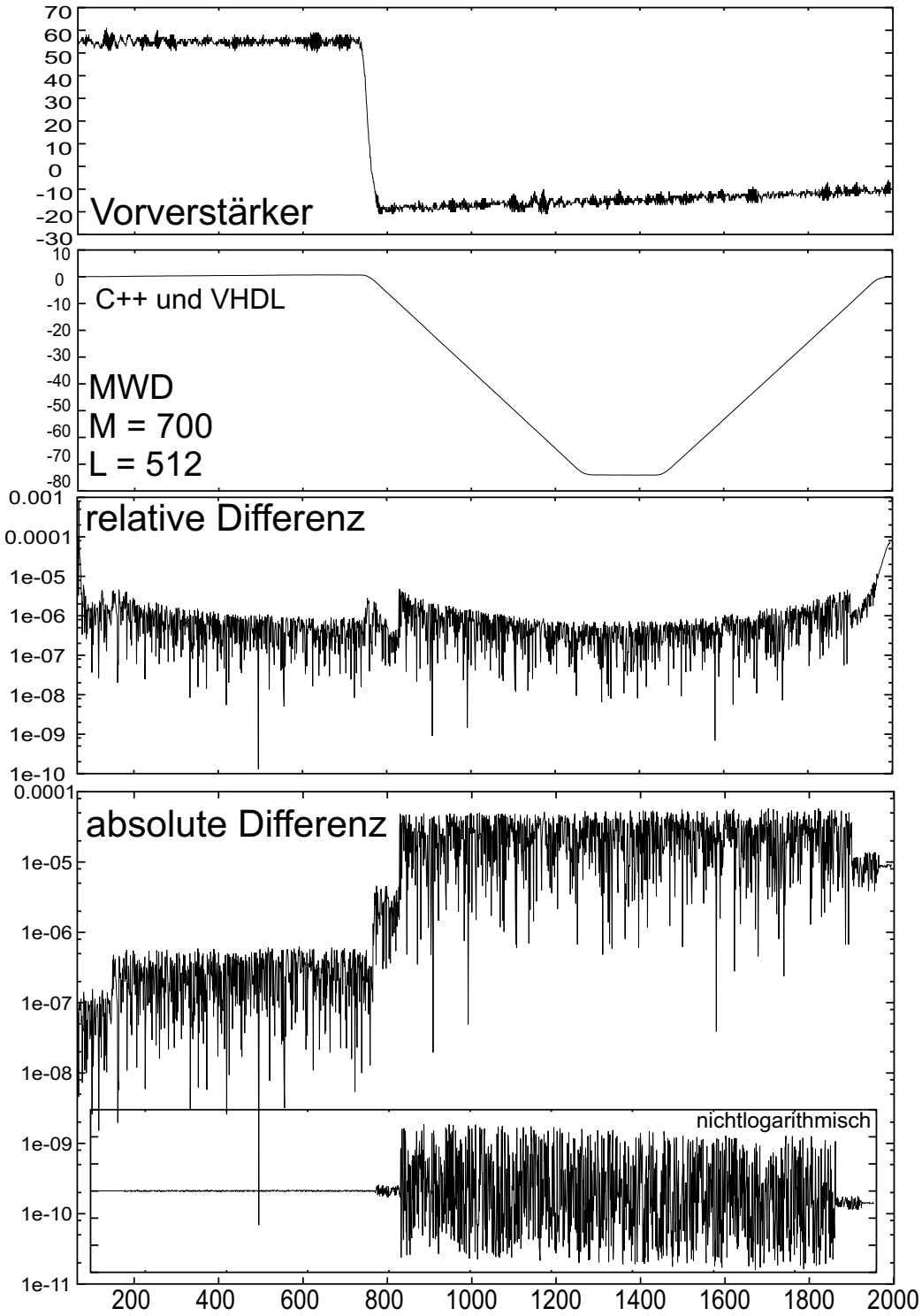


Abbildung 4.5: Beispiel für einen Vergleich der Ergebnisse des VHDL-MWD-Moduls mit denen des C++-MWD-Algorithmus. Die MWD-Kurven der beiden Implemetationen unterscheiden sich nur minimal.

Anschließend wird für jedes dieser ausgewählten Datensamples die Position des absoluten Minimums bestimmt und der hintere Teil mit dem exponentiellen Abfall zur weiteren Analyse herausgeschnitten. Mit Hilfe der in die Klasse PSADataSample integrierten ROOT-Fitting-Routinen wird dann eine Exponentialfunktion $A \cdot \exp(\alpha x)$ an die Daten gefittet und der angepasste Exponent α samt Fehler gespeichert. Als Startwerte für den Fit werden für A die Amplitude des absoluten Minimums und für α ein Wert zwischen -1 und 0 gesetzt.

Aus dieser Menge von gefitteten Exponenten werden Werte mit großem relativen oder absoluten Fehler entfernt, da hier davon ausgegangen werden kann, dass der automatische Fit nicht funktioniert hat. Über den Rest wird arithmetisch gemittelt und der Fehler dieser Mittelung berechnet[3].

In Abbildung 4.6 sind zwei der durch die Fits gewonnenen Verteilungen dargestellt. Tabelle 4.1 zeigt die aus Datensatz 1 auf obige Weise gewonnenen Integrationszeiten. Es gilt $\tau = -(f_{sampling} \cdot \alpha)^{-1}$ und $f_{sampling} = 200$ MHz.

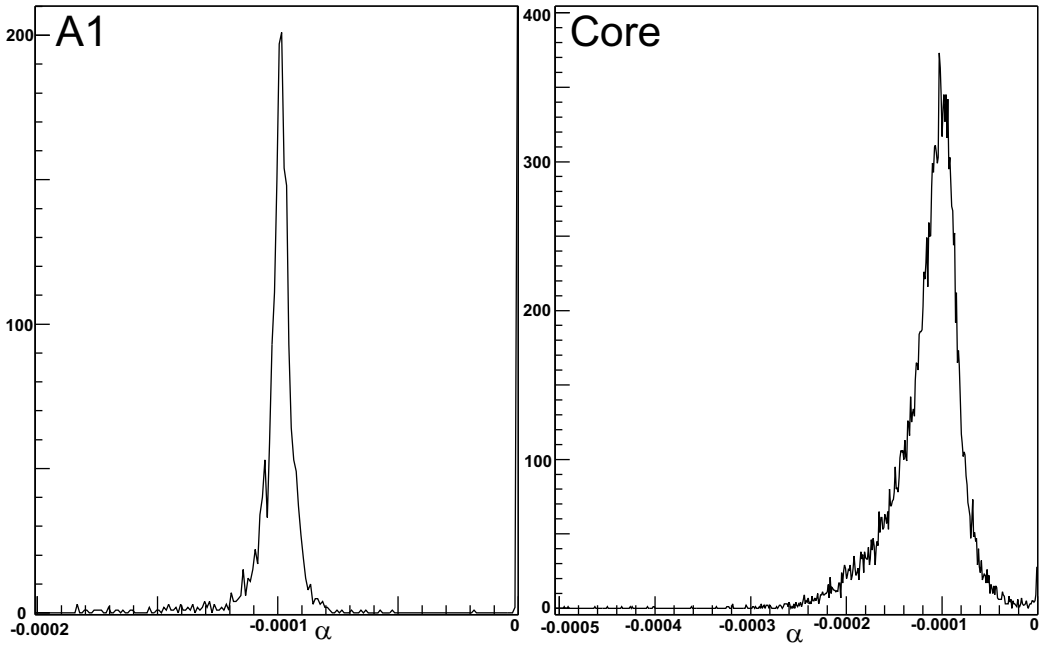


Abbildung 4.6: Verteilung der gefitteten Exponenten für Segment A1 und den Zentralkontakt.

Die Exponentenverteilung des Zentralkontakts weist einen Schwanz zu kleineren Integrationszeiten hin auf. Die vorgenommene arithmetische Mittelung ergibt also tendenziell einen zu kleinen Wert, was man in Tabelle 4.1

[3]Die Fehler der einzelnen Fits (zwischen 1 und 5%) werden hierbei nicht beachtet, sie dienen nur zur Vorsortierung der Daten.

Segment	α	$\pm$	rel. Unsicherheit	Anzahl Signale	τ [μs]
A1	-0,000102	1,71E-05	17%	1907	49,2
A2	-0,000101	1,89E-05	19%	2463	49,7
A3	-0,000100	2,61E-05	26%	1957	49,8
A4	-0,000101	2,94E-05	29%	2318	49,3
A5	-0,000104	1,49E-05	14%	2080	48,2
A6	-0,000103	2,57E-05	25%	1900	48,8
B1	-0,000101	1,90E-05	19%	1516	49,7
B2	-0,000103	2,26E-05	22%	1778	48,7
B3	-0,000104	3,08E-05	30%	1905	48,3
B4	-0,000105	2,38E-05	23%	1866	47,7
B5	-0,000097	1,79E-05	18%	1329	51,5
B6	-0,000102	1,91E-05	19%	1297	49,0
C1	-0,000103	2,48E-05	24%	1159	48,6
C2	-0,000101	1,79E-05	18%	1321	49,4
C3	-0,000103	2,13E-05	21%	1451	48,4
C4	-0,000104	3,86E-05	37%	1456	48,3
C5	-0,000103	1,96E-05	19%	948	48,5
C6	-0,000097	1,49E-05	15%	974	51,6
D1	-0,000102	3,44E-05	34%	628	49,2
D2	-0,000101	2,85E-05	28%	727	49,4
D3	-0,000102	2,66E-05	26%	852	49,0
D4	-0,000101	4,50E-05	45%	785	49,5
D5	-0,000105	1,17E-05	11%	623	47,8
D6	-0,000102	2,40E-05	23%	514	48,8
front	-0,000091	1,29E-05	14%	192	54,9
core	-0,000115	3,84E-05	33%	19564	43,4

Tabelle 4.1: Integrationszeiten der Vorverstärker des MARS-Detektors. Die Daten wurden anhand der exponentiellen Schwänze der Nettoladungssignale aus Datensatz 1 mittels Fits gewonnen. Die Spalte "Anzahl Signale" gibt die Anzahl der durch den Vergleich von Anfangs- zu Endstück ausgewählten Nettoladungssignale, die zur Mittelung herangezogen wurden (siehe Text). $\tau = -(f_{sampling} \cdot \alpha)^{-1}$

bestätigt findet. Der Zentralkontakt des MARS-Detektors besitzt einen anderen Vorverstärker als die Segmente. Er hat eine längere Anstiegszeit, die Integrationszeit ist jedoch die gleiche. Das Rauschniveau auf dem Zentralkontakt ist wesentlich höher als auf den Segmenten, was die, betrachtet man die Anzahl der ausgewerteten exponentielle Abfälle, relativ große Breite der Verteilung erklärt.

Allgemein weisen die gefitteten Exponenten für jeden Kanal eine breite Verteilung auf. Dies erklärt sich aus der Verwendung realer Detektorsignale. Zum Einen beeinflusst die Dauer der Signale die Form des Vorverstärkerpulses, zum Anderen tritt teilweise auch in Datensatz eins pile-up (durch Nettoladungssignale wie auch transiente Signale) auf, so dass der exponentielle Abfall gestört wird. Idealerweise müsste der Vorverstärker mit reinen Deltapulsen gefüttert werden, um so eine saubere Bestimmung der Integrationszeit zu ermöglichen. Hinzu kommt, dass in Datensatz 1 jeweils nur die ersten 5 μs des Abfalls zur Verfügung stehen und somit nur ein kleiner Ausschnitt der gesamten Flanke ($\tau \approx 50\mu$s).

Aufgrund dieser ungenauen Bestimmung der Integrationszeit aus den vorhandenen Daten wurde für einen Teil der folgenden Analysen die Integrationszeit für alle Kanäle auf 50 μs ($\alpha = -0.0001$) gesetzt[4]. Die Auswirkungen einer falsch bestimmten Integrationszeit werden in Abschnitt 4.2.2 besprochen.

4.2.2 MWD

Energieeichung

Die Ergebnisse des MWD-Energiealgorithmus müssen zunächst geeicht werden. Wie in Abschnitt 2.1.3 beschrieben, bezieht sich diese Eichung nicht auf die Parameter des Algorithmus, was später auch anhand der MARS-Daten gezeigt werden wird. Alle Tests des MWD-Algorithmus wurden mit Datensatz 1 der MARS-Daten durchgeführt.

Zur Energieeichung wird mittels des MWD-Algorithmus für alle 26 Kanäle eine Spektrum aus Datensatz 1 erstellt. Für die Segmente verteilen sich die darin jeweils enthaltenen 20.000 Werte auf drei Strukturen:

- Der Photopeak enthält diejenigen Gammaquanten, die ihre volle Energie (von in diesem Fall 662 keV) innerhalb eines Segmentes abgeben. Dies kann durch Photoeffekt geschehen, aber prinzipiell auch durch auf das Segment beschränkte Comptonstreunungen.

[4]Dem direkt auf den btrc-Datendateien aufsetzenden C++-MWD-Filter *BTRC_MWD* wird eine Kalibrationsdatei übergeben, die die Integrationszeiten aus Tabelle 4.1 enthält. Sinnvoll ist dies allerdings erst, wenn diese genauer bestimmt werden können.

- Der Comptonuntergrund wird durch Gammaquanten gebildet, die in das oder aus dem Segment comptongestreut wurden.

- Der alles überragende Nullpunktpeak wird durch die Ereignisse gebildet, bei denen keine Wechselwirkung im betrachteten Segment stattgefunden hat.

Der Zentralkontakt weist diesen Nullpunktpeak nicht auf. Im Gegensatz dazu ist der Photopeak nun die dominierende Struktur des Spektrums. Da der Zentralkontakt bei Gewinnung von Datensatz 1 zur Triggerung der Datenaufnahme diente, ist dies nicht verwunderlich. Abbildungen 4.7 und 4.8 zeigen zwei aus Datensatz 1 gewonnene Spektren für Segment A1 und den Zentralkontakt.

Die Form des Photopeaks setzt sich aus mehreren Komponenten zusammen. Wie man insbesondere am Spektrum des Zentralkontaktes sehen kann, weist der Peak neben einer Gaussverbreiterung einen Schwanz zu geringeren Energien hin auf.

Zunächst kommt es durch den statistischen Prozess der Elektron-Loch-Generierung im Detektor zu einer Verbreiterung der Photoeffektlinie, d.h. die durch die Wechselwirkung des Gammaquants mit dem Detektormaterial freiwerdende Energie verteilt sich statistisch auf die Bildung von Elektron-Loch-Paaren und der Anregung von Gitterschwingungen. Eine weitere Verbreiterung entsteht durch das elektronische Rauschen der verwendeten Bauelemente.

Der niederenergetische Schwanz entsteht zum Einen durch unvollständige Ladungssammlungen der Elektrode. Dazu führen können:

- Trapping: Ein Loch oder Elektron wird im Halbleiter von einer Störstelle eingefangen und gelangt dadurch gar nicht oder verspätet zur Elektrode. Dieser Effekt sollte für den MARS-Detektor vernachlässigbar sein, da der Kristall keine Strukturschäden aufweist.

- Rekombination: Loch und Elektron annihilieren sich gegenseitig und gehen nicht mehr in die Ladungssammlung ein.

- Zonen mit geringem elektrischen Feld: Ladungsträger werden nur sehr langsam oder gar nicht zur Elektrode transportiert.

Zum Anderen können elektrische Effekte in der Ausleseelektronik dazu beitragen. Warum der Zentralkontakt des MARS-Detektors diese schlechte Auflösung produziert, ist nicht verstanden und wird noch untersucht, da andere segmentierte Detektoren wie z.B. die Miniball-Kristalle dieses Problem nicht zeigen.

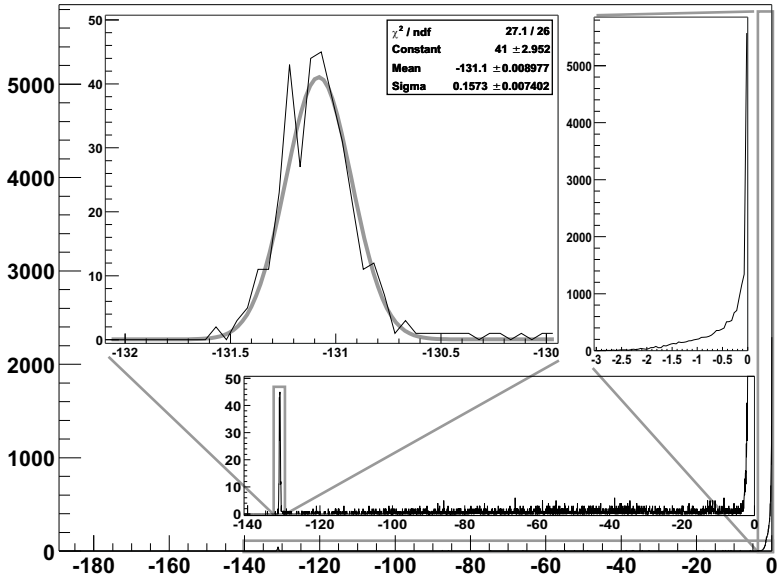


Abbildung 4.7: Ungeeichtes Energiespektrum für Segment A1 bestehend aus 20.000 Einträgen. Binbreite 0,05. Der Photopeak enthält knapp 370 Einträge. Ausgeprägteste Struktur ist der Nullpunktpeak.

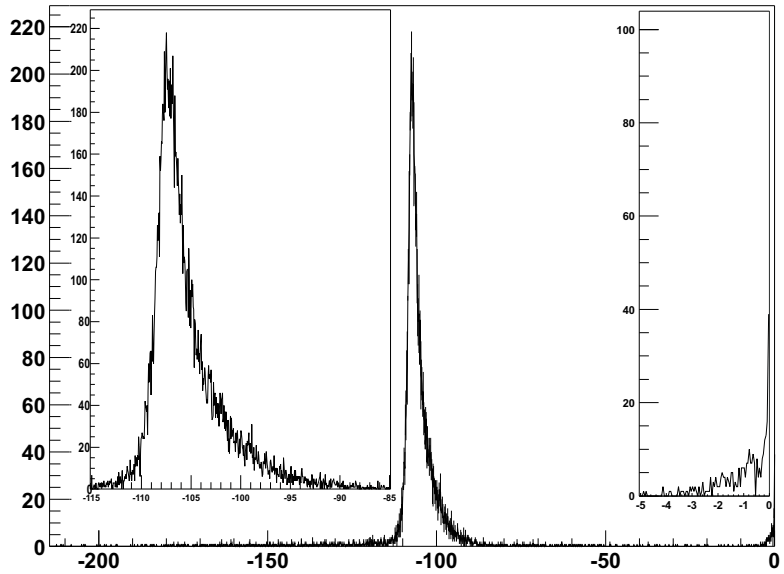


Abbildung 4.8: Ungeeichtes Energiespektrum für den Zentralkontakt bestehend aus 20.000 Einträgen. Binbreite 0,05. Der Photopeak ist die dominierende Struktur. Der Nullpunktpeak hat knapp über 400 Einträge, die durch die Leersamples in Datensatz 1 erzeugt werden.

An den Photopeak schließt sich der Comptonuntergrund zu niedrigeren Energien an. Paarbildungseffekte treten bei einer ^{137}Cs-Quelle augrund der zu geringen Energie nicht auf. [27], [28] und [16] geben einen detailliertere Überblick über die Zusammensetzung des Spektrums sowie über verschiedene Ansätze die Linienform der Photopeaks zu fitten.

Aufgrund der geringen Statistik in den Photolinien der Segmentspektren (maximal zwischen 300 und 400 Einträge im Peak) werden die Fits dieser Linien mit einer reinen Gaussfunktion durchgeführt. Der Mittelwert dieser gefitteten Verteilung dient zur Eichung des Spektrums, wird also mit 661,66 keV gleichgesetzt, die Standardabweichung ergibt, normiert auf den Mittelwert und multipliziert mit dem Faktor 2,35 , die Energieauflösung (FWHM).

Auswirkungen der Parameter

Die Darstellung der Auswirkungen der Parameter des MWD-Algorithmus in Abschnitt 2.1.2 beruhen auf theoretischen Überlegungen. Die folgenden Analysen dienen der experimentellen Untermauerung dieser Aussagen. In die erstellten Energiespektren flossen jeweils alle 20.000 Ereignisse aus Datensatz 1 ein. Der Gaussfit des Photopeaks erfolgte im Fall der Untersuchung der Integrationszeit von Hand, d.h. der Fitroutine wurde manuell der zu fittende Bereich um die Linie vorgegeben, da diese Analyse nur für ein Segment vorgenommen wurde. Zur Untersuchung der Auswirkungen der Parameter WS und AL wurde der Fit automatisiert. Nach Entfernung des Nullpunktpeaks aus dem Spektrum wurde die Position des dann verbliebenen absoluten Maximums als Ausgangspunkt für den Gaussfit in einem Bereich um dieses Maximum genommen. Wie man anhand der folgenden Abschitte sehen kann, liefert diese Methode etwas schlechtere Werte als die manuelle Auswahl des zu fittenden Bereiches.

Integrationskonstante des Vorverstärkers τ

Die Auswirkungen einer falsch bestimmten Integrationszeit lassen sich durch die Differenzengleichung

$$\begin{aligned} y'[n] &= y[n] + \left(1 - \frac{k'}{k}\right) \cdot \sum_{m=1}^{\infty} k^m \cdot y[n-m] \\ &= y[n] - k' \cdot y[n-1] + k \cdot y'[n-1] \end{aligned} \tag{4.3}$$

beschreiben. Hierbei gilt $k = \exp(\alpha) = \exp(-(f_{sampling} \cdot \tau)^{-1})$, ebenso für k'. $y[n]$ ist das Ergebnis des MWD-Algorithmus mit richtiger Intergrationszeit τ, $y'[n]$ stellt das Ergebnis der Anwendung mit falscher Integrationszeit τ' dar.

Man erkennt, dass sowohl das Verhältnis wie auch die absoluten Werte der Integrationszeiten die Größe des Fehlers beeinflussen. Schematisch ist die Veränderung der Pulsform in Abbildung 4.9 dargestellt.

Eine im MWD-Algorithmus zu klein gewählte Vorverstärkerintegrationszeit lässt den Algorithmus mehr Ladungsträger sehen als tatsächlich vorhanden sind. Der MWD-Peak wird dadurch erhöht. Eine zu lang bestimmte Integrationszeit hat den umgekehrten Effekt, die ermittelte Energie verkleinert sich. Hinzu kommt, dass die maximale Amplitude des MWD-Peaks abhängig von der Pulsdauer wird (siehe Abbildung 4.10), was zu einer Verbreiterung der Linie führt.

Abbildung 4.11 zeigt die Ergebnisse der Untersuchung unterschiedlich gewählter Vorverstärkerintegrationszeiten, Tabelle 4.2 die dazugehörigen Werte. Die verwendeten Daten stammen aus Segment A1, Fensterbreite und Mittelungslänge sind für alle Spektren konstant ($WS = 4\mu s, AL = 3,5\mu s$).

$\frac{\tau'}{\tau}$	$\bar{\mu}$	FWHM [keV]
0,53	136,1	2,32 ± 0,127
0,59	135,0	2,26 ± 0,117
0,63	134,5	2,25 ± 0,127
0,71	133,4	1,95 ± 0,099
0,77	132,8	2,00 ± 0,098
0,83	132,3	1,94 ± 0,091
0,91	131,7	1,85 ± 0,093
1,00	131,2	1,87 ± 0,083
1,11	130,6	1,96 ± 0,088
1,25	130,1	1,94 ± 0,088
1,43	129,6	1,93 ± 0,083
1,67	129,1	1,91 ± 0,089
3,33	127,6	1,88 ± 0,085
10,00	126,6	1,83 ± 0,080

Tabelle 4.2: Werte zu Abbildung 4.11.

Sehr schön zu sehen ist die oben beschriebene Veränderung der Position des Mittelwertes der Gaussverteilung sowie das Minimum in der Breite des Gaussfits an der Position der wahren Vorverstärkerintegrationszeit.

Wie sich eine falsch gewählte Integrationszeit auf die Linearität des Spektrums auswirkt, kann anhand der zur Verfügung stehenden MARS-Daten nicht untersucht werden, da Datensatz 1 monoenergetisch ist und für Datensatz 2 keine Energieeichung möglich ist.

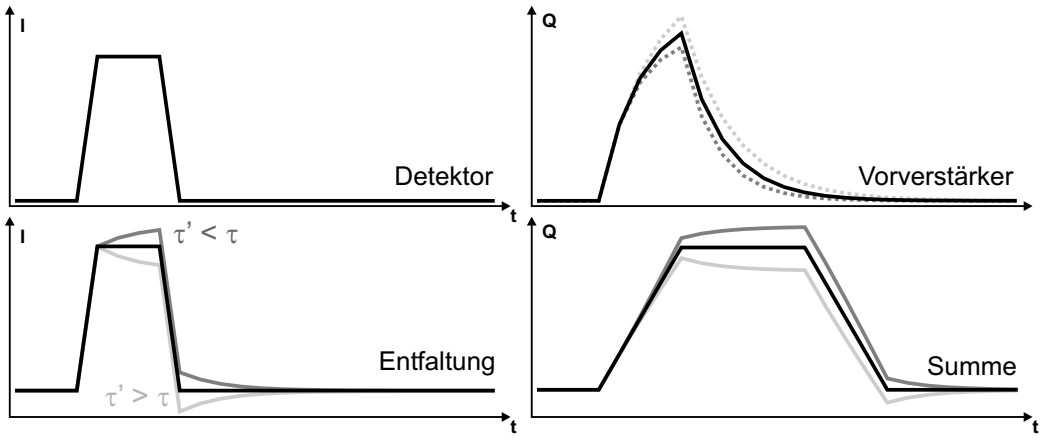


Abbildung 4.9: Schematische Darstellung der Auswirkung einer falsch bestimmten Integrationszeit auf den MWD-Puls.

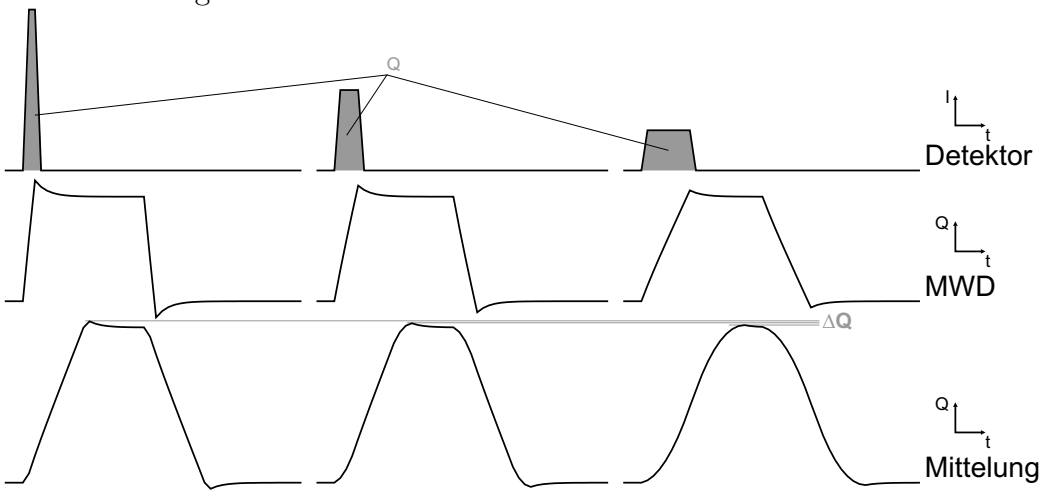


Abbildung 4.10: Schematische Darstellung der Auswirkung einer falsch bestimmten Integrationszeit auf die Auflösung des Photopeaks. Die Fläche unter den Detektorpulsen ist bei allen dreien die gleiche, sie unterscheiden sich nur in der Pulsdauer.

Mittelungslänge AL

Der Parameter AL beeinflusst die Rauschunterdrückung. Dabei ist AL den in Abschnitt 2.1.2 formulierten Bedingungen unterworfen. Wie man anhand Abbildung 4.12 sehen kann, stellt

$$AL = WS - \Delta t_{max}$$

den besten Wert für die Mittelungslänge dar. Wird AL zu groß gewählt, so gehen Teile der Flanken des MWD-Pulses in die Mittelung über das Plateau

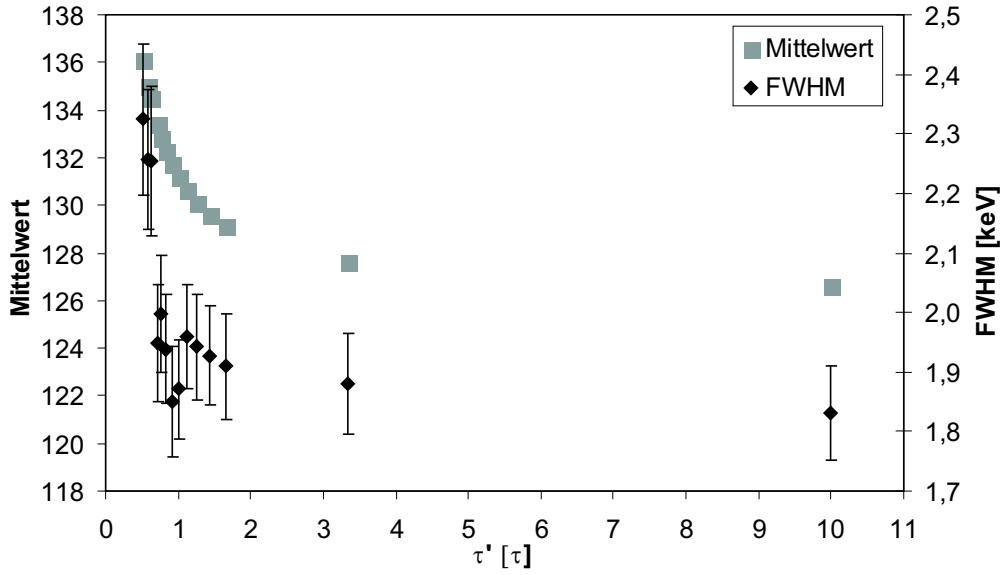


Abbildung 4.11: Normierungswert und Energieauflösung in Abhängigkeit von der Vorverstärkerintegrationskonstanten. Das σ der gefitteten Gaussverteilungen ist die Energieauflösung FWHM umgerechnet und auf den gefitteten Mittelwert normiert. Der exponentielle Verlauf von Normierungswert und Auflösung ergibt sich aus $k = \exp\left(-(f_{sampling}\tau)^{-1}\right)$.

WS-AL [μs]	AL [μs]	$\Delta\bar{\mu}$ [keV]	FWHM [keV]
3,5	0	18,16	5,71 $\pm$ 0,309
3	0,5	2,52	2,78 $\pm$ 0,133
2,5	1	1,51	2,43 $\pm$ 0,106
1,75	1,75	0,50	2,12 $\pm$ 0,092
1	2,5	0,50	1,82 $\pm$ 0,081
0,5	3	0,00	1,76 $\pm$ 0,077
0,25	3,25	-1,01	2,26 $\pm$ 0,109
0,1	3,4	-6,56	4,22 $\pm$ 0,204

Tabelle 4.3: Werte zu Abbildung 4.12.

mit ein. Dadurch wird der gemessene Wert zu niedig. Außerdem wird die Höhe des Maximums abhängig von der Steigung der Flanken, letztendlich also von der Pulsdauer des Detektorsignales. Dies verschlechtert die Energieauflösung.

Eine kurze Mittelungslänge belässt mehr Rauschen auf dem Signal, was

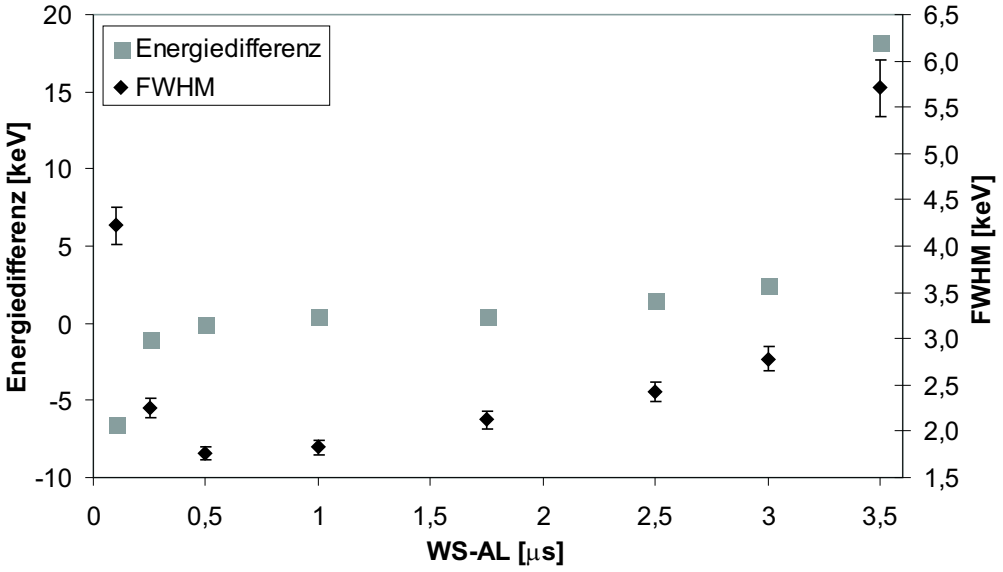


Abbildung 4.12: Der MWD-Algorithmus angewendet auf Segment A1 mit konstanter Fensterbreite $WS = 3,5\mu s$. Die Mittelungslänge AL variiert zwischen 0 (keine Mittelung) und 3,4 μs. Die Energiedifferenzen und Halbwertsbreiten sind auf die Messung mit dem Parametersatz $WS = 3,5\mu s$ und $AL = 3,0\mu s$ geeicht.

das Maximum des gemittelten Peaks tendenziell erhöht und die Verteilung dieser Maximalwerte verbreitert, die Energieauflösung also ebenfalls verschlechtert.

Fensterlänge WS

Die Messungen für unterschiedliche Fensterlängen wurden jeweils mit der optimalen Mittelungslänge von $AL = WS - 500ns$ durchgeführt. Zu erwarten ist, dass sich die Breite der Photolinie bei größer werdender Fensterlänge verringert und schließlich einen Minimalwert erreicht. Der Mittelwert der Verteilung sollte unabhängig von der gewählten Fensterbreite sein. Abbildung 4.13 zeigt jedoch ein etwas anderes Bild.

Die Auflösung scheint sich für große Fensterbreiten wieder zu verschlechtern. Dies lässt sich auf allen 25 Segmenten beobachten. Auswirken dürfte sich hier die geringe Länge der Samples in Datensatz 1 von 10 μs. Die Vorverstärkerpulse beginnen etwa bei 4 μs, so dass sehr lange Fenster den Puls nicht vollständig überstrichen haben, bevor das Datensample zu Ende ist. Zusätzlich treten vermehrt pile-up-Effekt auf, da das Fenster bei entspre-

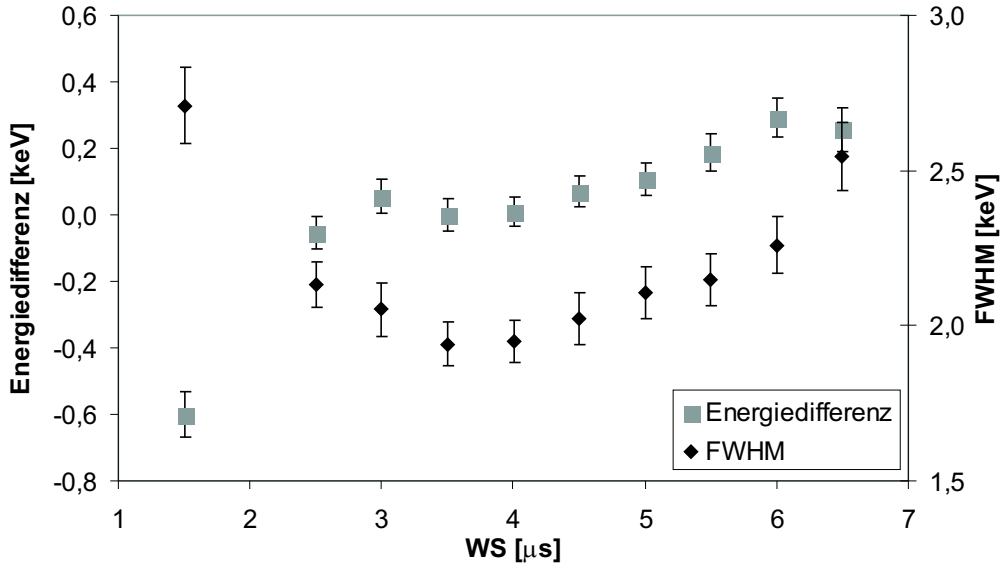


Abbildung 4.13: Energieauflösung des Segmentes A1 für verschiedene Fensterbreiten. Die Mittelungslänge war jeweils 500 ns kürzer. Entgegen den Erwartungen scheint die Energieauflösung für große Fensterbreiten schlechter zu werden. Dies sind Auswirkungen der Beschaffenheit von Datensatz 1. Die maximale Auflösung ist schlechter als in den vorangegangenen Darstellungen, da diese Messungen automatisch gefittet wurden.

WS [μs]	$\Delta\bar{\mu}$ [keV]	FWHM [keV]
1,5	-0,60 $\pm$ 0,066	2,71 $\pm$ 0,123
2,5	-0,06 $\pm$ 0,049	2,13 $\pm$ 0,074
3	0,06 $\pm$ 0,050	2,05 $\pm$ 0,086
3,5	0,00 $\pm$ 0,048	1,94 $\pm$ 0,070
4	0,01 $\pm$ 0,045	1,95 $\pm$ 0,066
4,5	0,07 $\pm$ 0,047	2,02 $\pm$ 0,085
5	0,11 $\pm$ 0,050	2,11 $\pm$ 0,085
5,5	0,19 $\pm$ 0,055	2,15 $\pm$ 0,083
6	0,29 $\pm$ 0,058	2,26 $\pm$ 0,091
6,5	0,26 $\pm$ 0,065	2,54 $\pm$ 0,110

Tabelle 4.4: Werte zu Abbildung 4.13.

chender Breite mehrere Ereignisse umfassen kann. Die Samples in Datensatz 1 enthalten zwar hauptsächlich Einzelereignisse, dennoch treten teilweise auch

zwei Ereignisse innerhalb der 10 μs auf.

Energieauflösung

Segmente

Die Auswertung der 25 Segmentkanäle aus Datensatz 1 legt nahe, dass mit dem MWD-Algorithmus für ein einzelnes Segment eine Energieauflösung von 1,9 keV bei 662 keV, d.h. von etwa 3‰, erreicht werden kann, bei einer maximalen Ereignisrate von bis zu 120 kHz auf dem Segment [5].

Zur genaueren Bestimmung der maximal möglichen Energieauflösung mangelt es dem MARS-Datensatz 1 an Statistik. Selbst die ausgeprägtesten Photopeaks enthalten nur zwischen 300 und 400 Werte. Der Photopeak des Zentralkontakts hat zwar eine wesentlich bessere Statistik, allerdings aufgrund des höheren Rauschniveaus seines Vorverstärkers dennoch eine schlechtere Energieauflösung als die Segmente.

Tabelle 4.5 zeigt die aus MARS-Datensatz 1 mittels des MWD-Algorithmus errichbare Energieauflösung für die 26 Kanäle des MARS-Detektors.

Gesamtdetektor

Neben der Energieauflösung der einzelnen Segmente interessiert die daraus resultierende Energieauflösung des Gesamtdetektors. Zur Bestimmung dieser wird zunächst die Breite der Nullpunktpeaks der geeichten Segmentspektren bestimmt. Anschließend werden für jedes Ereignis die in den 25 Segmenten ermittelten Energien aufsummiert, wenn sie oberhalb des Nullpunktpeaks liegen. Ein so gewonnenes Spektrum zeigt Abbildung 4.14. Die Energien des Frontsegments wurde in diesem Fall nicht in die Summe miteinbezogen, da für dieses Segment keine Energieeichung anhand des MARS-Datensatz 1 möglich ist. Es fand keine explizite Diskriminierung transienter Signale statt. Es wurde einfach das absolute Minimum des 10 μs langen gemittelten MWD-Signales als Wert für das Spektrum verwendet. Der negative Teil der bipolaren transienten MWD-Pulse wird daher als Energie in das Spektrum eingetragen, wenn er über das Rauschniveau hinausragt. Verhindern ließe sich das auf die einfachste Weise mittels eines Vergleichs der Amplituden des absolutem Minimums und Maximums des betrachteten Bereichs. Befindet sich hier ein einzelnes transientes Signal, so ist dieses Verhältnis ≈ 1. (siehe unten)

Zum Vergleich ist in Abbildung 4.15 das Zentralkontaktspektrum, gewonnen aus denselben Messungen, dargestellt.

[5] Wählt man WS = 3,5 μs und AL = 3 μs, so ist die maximale Pulslänge des gemittelten MWD-Pulses 7,5 μs.

WS [μs]	3,5	4
Segment	FWHM [keV]	
A1	1,94 ± 0,070	1,95 ± 0,066
A2	2,03 ± 0,058	2,04 ± 0,053
A3	2,62 ± 0,078	2,78 ± 0,087
A4	2,26 ± 0,082	2,32 ± 0,081
A5	2,08 ± 0,069	2,14 ± 0,065
A6	2,12 ± 0,077	2,08 ± 0,068
B1	2,19 ± 0,136	2,31 ± 0,153
B2	2,20 ± 0,111	2,21 ± 0,076
B3	2,31 ± 0,126	2,18 ± 0,108
B4	2,52 ± 0,182	2,41 ± 0,162
B5	2,27 ± 0,123	2,56 ± 0,237
B6	2,49 ± 0,157	2,49 ± 0,107
C1	1,85 ± 0,123	1,75 ± 0,100
C2	2,43 ± 0,110	2,48 ± 0,104
C3	2,07 ± 0,094	2,32 ± 0,139
C4	1,94 ± 0,141	2,17 ± 0,126
C5	4,84 ± 1,188	2,76 ± 0,151
C6	2,12 ± 0,188	2,18 ± 0,118
D1	2,59 ± 0,211	3,20 ± 0,249
D2	2,46 ± 0,147	3,08 ± 0,167
D3	2,25 ± 0,162	2,21 ± 0,137
D4	2,23 ± 0,174	2,33 ± 0,176
D5	2,39 ± 0,192	2,32 ± 0,162
D6	4,11 ± 0,433	5,69 ± 1,375
front	-	-
core	(19,97)	(25,35)

Tabelle 4.5: Energieauflösung des MWD-Algorithmus für die 26 Kanäle des MARS-Detektors. Die Gaussfits der Photopeaks erfolgten auf die oben beschriebene automatische Art.

Wie man schon in Abbildung 4.8 anhand des ungeeichten Zentralkontaktspektrums sehen kann, besitzt der Photopeak auf dem Zentralkontakt (core) aufgrund des hohen Rauschniveaus einen sehr ausgedehnten Schwanz zu niedrigen Energien hin. Daher muss ein Gaussfit für diesen Peak zu einem nur sehr eingeschränkt nutzbaren Ergebnis führen.

Für das Frontsegement (front) kann keine Energieauflösung angegeben werden, da dafür die Anzahl an Ereignissen mit der vollen Gammaenergie in diesem Segment zu gering ist. Das aus Datensatz 1 erstellte Spektrum weist für dieses Segment keinen deutlichen Photopeak auf.

Segment D6 ist leicht defekt und zieht einen Leckstrom wodurch sich die Energieauflösung verschlechtert.

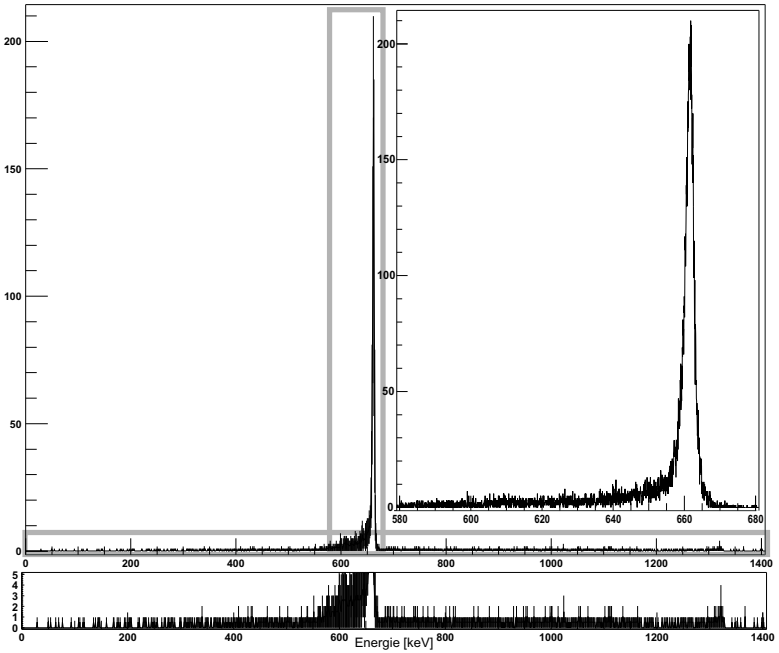


Abbildung 4.14: MWD-Energiespektrum des Gesamtdetektors. Die Fensterbreite beträgt 4 μs, die Mittelungslänge 3,5 μs. Die Binbreite beträgt 0,05 keV. Das Frontsegment wurde nicht mit einbezogen, da sich dieses mangels Statistik nicht eichen lässt (kein Photopeak). Bei etwa 1325 keV, der doppelten Gammaenergie, erkennt man eine kleine Kante. Die Ereignisse mit Energien zwischen dem Photopeak bei 662 keV und dieser Kante stellen pile-ups dar, also zwei Wechselwirkungen innerhalb von 4 μs.

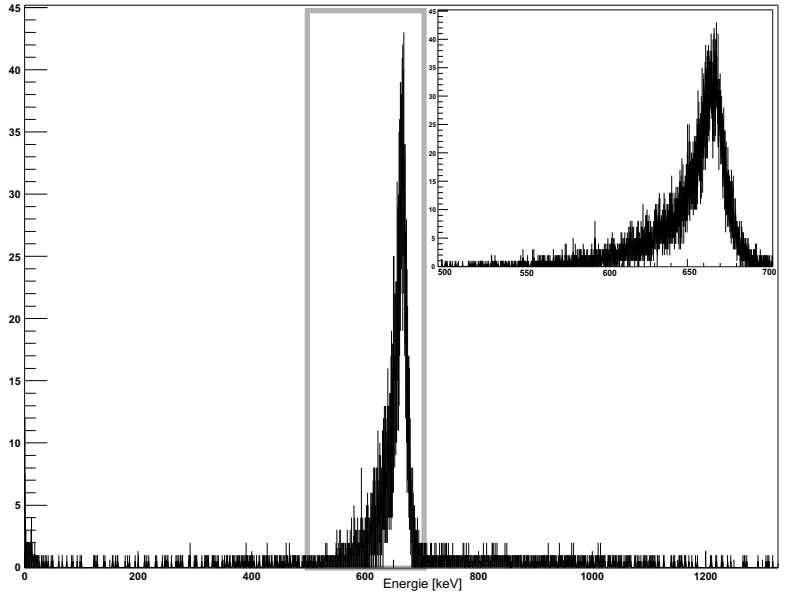


Abbildung 4.15: Zum Vergleich das Energiespektrum des Zentralkontakts für den selben MWD-Lauf wie in Abbildung 4.14. Die Binbreite ist auch hier 0,05 keV. In Abbildung 4.8, dem noch ungeeichten Spektrum, ist die Binbreite etwa um den Faktor 6 größer, so dass das Spektrum dort wesentlich glatter aussieht.

Die Breite (FWHM) des aus den 24 Segmenten berechneten Photopeaks beträgt etwas unter 4 keV (Gaussfit), sein Mittelwert liegt bei etwa 661,5 keV. Hierbei wirkt sich das n-fach aufsummierte Rauschen der Einzelkanäle aus.

Eine zusätzliche Diskriminierung transienter Signale mittels eines Vergleichs von absolutem Maximum und Minimum eines MWD-Samples brachte keine Verbesserung. Dieser Effekt durch fälschlicherweise in die Summe mit einbezogene transiente Signale ist also wohl sehr klein.

Deutlich zu erkennen ist zudem die sehr schlechte Energieauflösung des Zentralkontakts aufgrund seines höheren Rauschniveaus.

4.2.3 Trigger

Abbildungen 4.16 bis 4.21 geben einen ersten Eindruck von der Leistungsfähigkeit der beiden Triggermethoden. Die Daten stammen aus Datensatz zwei des MARS-Detektors und wurden mit 8 Bit bei 100 MHz abgetastet. Die mittlere Ereignisrate im Detektor (gemessen auf dem Zentralkontakt) ist 38 kHz, die Quelle ist ^{60}Co. Eine Energieeichung für die Daten ist mangels Statistik (ca. 2.000 Ereignisse völlig unterschiedlicher Energie) nicht möglich. Abbildung 4.16 zeigt die Trigger-Algorithmen, angewendet auf den Zentralkontakt des Detektors, Abbildung 4.19 dieselbe Zeitspanne für Segment A1, eines der sechs Segmente im vordersten Ring. Abbildungen 4.17 und 4.18 sowie 4.20 und 4.21 sind Ausschnittsvergrößerungen zur besseren Veranschaulichung der im Text dargestellten Sachverhalte.

Ausschnitt 1 Zentralkontakt (Abb. 4.17):

Dieser Ausschnitt zeigt die Sensitivität der beiden Trigger-Algorithmen:

Zeitliche Pulstrennung: Signale A und B liegen etwas weniger als 2μs auseinander.

Pulshöhensensitivität: Signal C verschwindet fast im Rauschen. Es handelt sich dabei tatsächlich um ein Ereignis mit sehr geringer Energiedeposition in Segment A3 des Detektors (siehe Abb. 3.11).

Ausschnitt 2 Zentralkontakt (Abb. 4.18):

Ausschnitt 2 zeigt zwei weitere eng beieinander liegende Ereignisse. Signale D und E liegen etwa 1,2 μs auseinander, Signale F und G ungefähr 1μs. Bei F handelt es sich zusätzlich um ein sehr langes Signal von knapp 400 ns.

Ausschnitt 3 Segment A1 (Abb. 4.20):

Ausschnitt 3 zeigt einige transiente Signale[6].

Ausschnitt 4 Segment A1 (Abb. 4.21):

Ausschnitt 4 zeigt ein Nettoladungssignal. überlagert durch ein transientes Signal.

Anhand der zur Verfügung stehenden Daten lassen sich folgende qualitative Aussagen zu den beiden Triggermethoden treffen:

MWDTrigger

Aus prinzipiellen Überlegungen lässt sich ableiten, dass der MWDTrigger-Algorithmus, kombiniert man den Schwellendiskriminator noch zusätzlich mit einem Maximumdetektor, Signale ab einem Abstand von $\Delta T = 2 \cdot WS = 1\mu s$ (bei $\Delta t_{max} = 500ns$) trennen können sollte. Ohne Maximumdetektor wird ein Schwellendiskriminator Pulse spätestens ab einem Abstand von $\Delta T = 2 \cdot AL = 4 \cdot WS = 2\mu s$ trennen können, da $2AL$ in dieser Parameterkonstellation die maximale Länge eines gemittelten MWD-Pulses ist.

Die Wahl der Schwellenenergie hängt vom Rauschlevel auf dem Signal ab und lässt sich ohne eingehende Analyse der Rauschquellen nur schwer abschätzen.

Setzt man sowohl oberhalb als auch unterhalb des Nullniveaus eine Schwelle, so lassen sich mit diesem Algorithmus auch transiente Signale aus dem Datenstrom herausfiltern. Sie erzeugen einen bipolaren MWD-Puls, der mit den zwei Schwellendiskriminatoren erkannt werden kann. Die Schwellenwerte können in diesem Fall eventuell sogar niedriger angesetzt werden, da die Bipolarität als zusätzliche Auslösebedingung verwenden werden kann.

Schlussfolgerungen aus den Abbildungen:

Ausschnitt 1 Zentralkontakt (Abb. 4.17):

Die beiden etwa 2 μs auseinander liegenden Signale A und B werden durch den MWDTrigger sauber getrennt. Signal C wird bei entsprechend niedrig gesetzter Schwelle ebenfalls ein Triggersignal auslösen.

[6]Transiente Signale können nur in den einzelnen Segmenten auftreten, nicht auf dem Zentralkontakt.

Ausschnitt 2 Zentralkontakt (Abb. 4.18):

Hier erkennt man die Grenzen der zeitlichen Auflösung des MWDTriggers. Signale D und E (1,2 μs) ließen sich unter Einsatz eines Maximumdetektors wohl gerade noch trennen, während Signale F und G (1 μs) nicht mehr sauber unterscheidbar sind.

Ausschnitt 3 Segment A1 (Abb. 4.20):

Dieser Ausschnitt verdeutlicht die Bipolarität der MWDTrigger-Pulse.

Ausschnitt 4 Segment A1 (Abb. 4.21):

Das überlagernde transiente Signal lässt sich nur unter erheblichem Aufwand automatisch erkennen. Das Ladungssignal wird dennoch klar erkannt.

SCC

Die Pulsbreite des SCC-Ausgangspulses bewegt sich bei Flankenanstiegszeiten von 100 bis 500 ns zwischen 1,2 und 1,6 μs. Aufgrund des parabelförmigen Anstiegs und Abfalls der SCC-Pulse sollte die zeitliche Trennbarkeit zweier Pulse unterhalb dieses Bereichs liegen.

Der Algorithmus ist nicht in der Lage, transiente Signale als solche zu erkennen. Da der von ihnen erzeugte SCC-Puls zumeist oberhalb der Triggerschwelle liegt, können sie zu einem ungewollten Auslösen des Triggers führen. Bei einem auf den Zentralkontakt beschränkten Einsatz tritt diese Problematik natürlich nicht auf.

Ausschnitt 1 Zentralkontakt (Abb. 4.17):

Die Flanke C ist sehr schwach ausgeprägt und erzeugt daher im SCC-Trigger ein schwächeres Signal. Die Triggerschwelle muss dementsprechend angepasst werden.

Ausschnitt 2 Zentralkontakt (Abb. 4.18):

Alle Pulse erzeugen getrennt voneinander saubere Triggersignale.

Ausschnitt 3 Segment A1 (Abb. 4.20):

Alle drei transienten Signale erzeugen Peaks, die zwar nicht die maximale Höhe erreichen (siehe auch Abb. 4.19), aber bei zu niedriger Schwelle der anschließenden Maximumdetektion zu einem ungewollten Auslösen führen können.

Ausschnitt 4 Segment A1 (Abb. 4.21):

> Das dem Nettoladungspuls überlagerte transiente Signal führt zu keiner merklichen Änderung der SCC-Pulsform.

Ein quantitativer Vergleich des SCC-Triggers mit analogen Triggermodulen findet sich in [12].

Vergleich MWDTrigger ⇔ SCC

Ebenso wie der MWDTrigger-Algorithmus erkennt auch der SCC-Trigger alle Ereignisse in den zwei Abschnitten in Abbildung 4.16 und 4.19.

Der SCC-Trigger erlaubt aufgrund der schmaleren Pulsform eine bessere zeitliche Trennung zweier Pulse als der MWDTrigger. Desweiteren erzeugt der SCC-Algorithmus auch für schwächere Signale einen deutlicheren Triggerpuls (siehe Abbildung 4.18, Signale E und F).

Der MWDTrigger bietet die Möglichkeit, unabhängig von der Pulsform auf die Energie eines Pulses zu triggern. Desweiteren kann er zur Erkennung transienter Signale herangezogen werden. Die benötigte Logik bzw. Rechenkapazität ist für den MWDTrigger unabhängig von der tatsächlich gewählten Fensterbreite, während der SCC-Trigger für jeden Vergleichspunkt innerhalb des Fensters weitere Logikgatter benötigt.

Um einen quantitativen Vergleich der beiden Triggermethoden durchzuführen, benötigt man Datensätze mit einem herkömmlichen Referenztrigger.

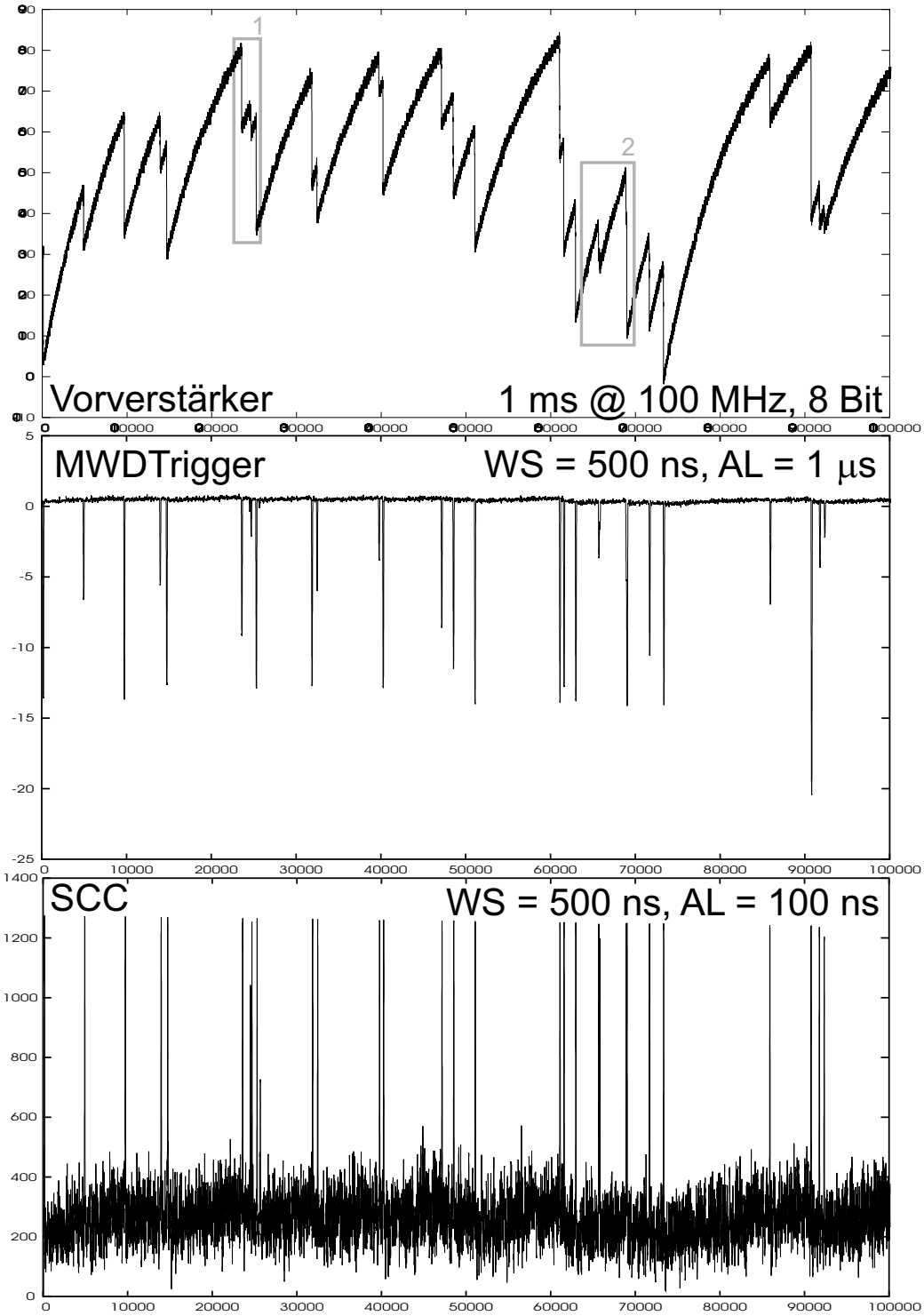


Abbildung 4.16: Zentralkontakt, 1 ms

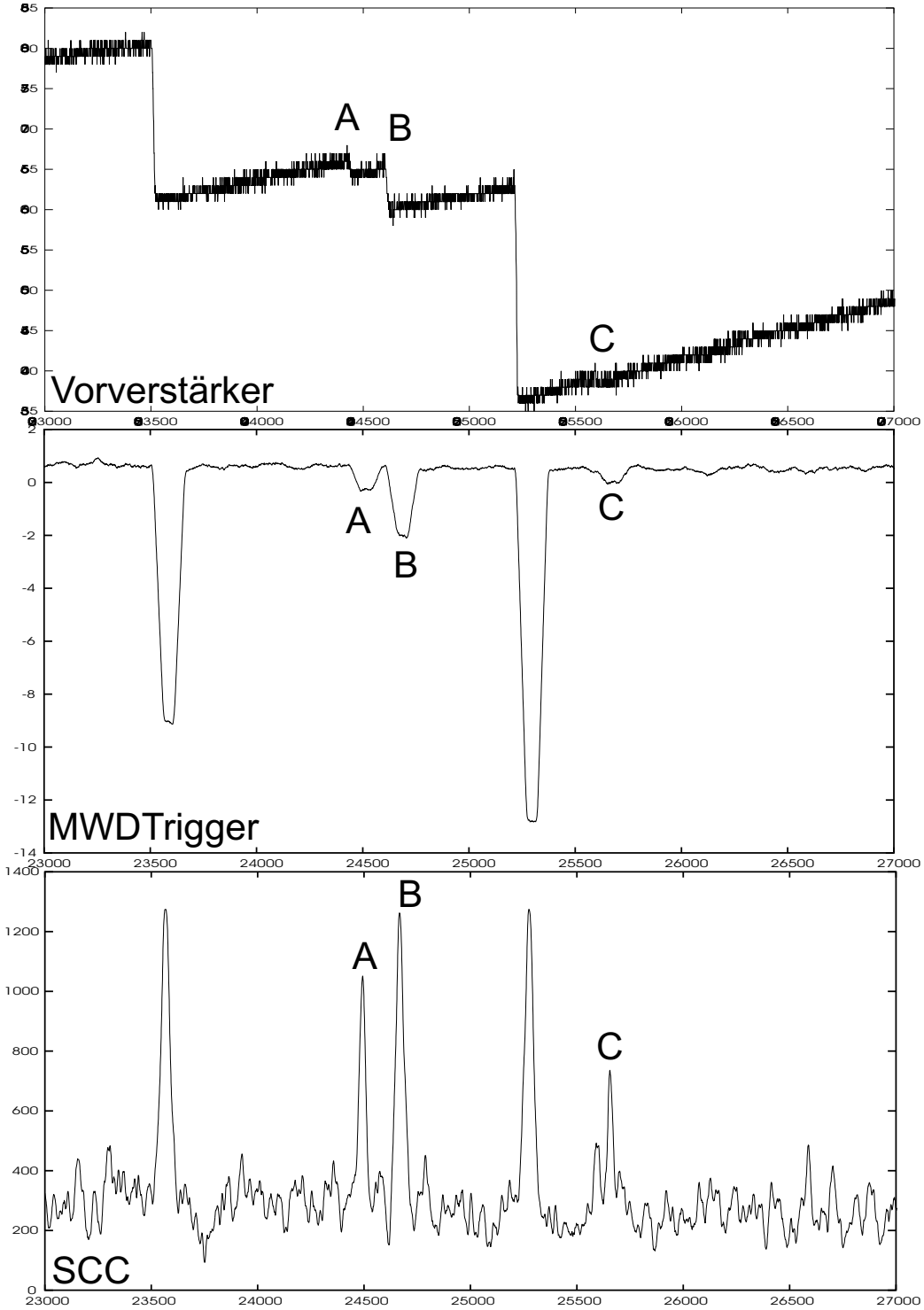


Abbildung 4.17: Zentralkontakt, 40 μs, Ausschnitt 1

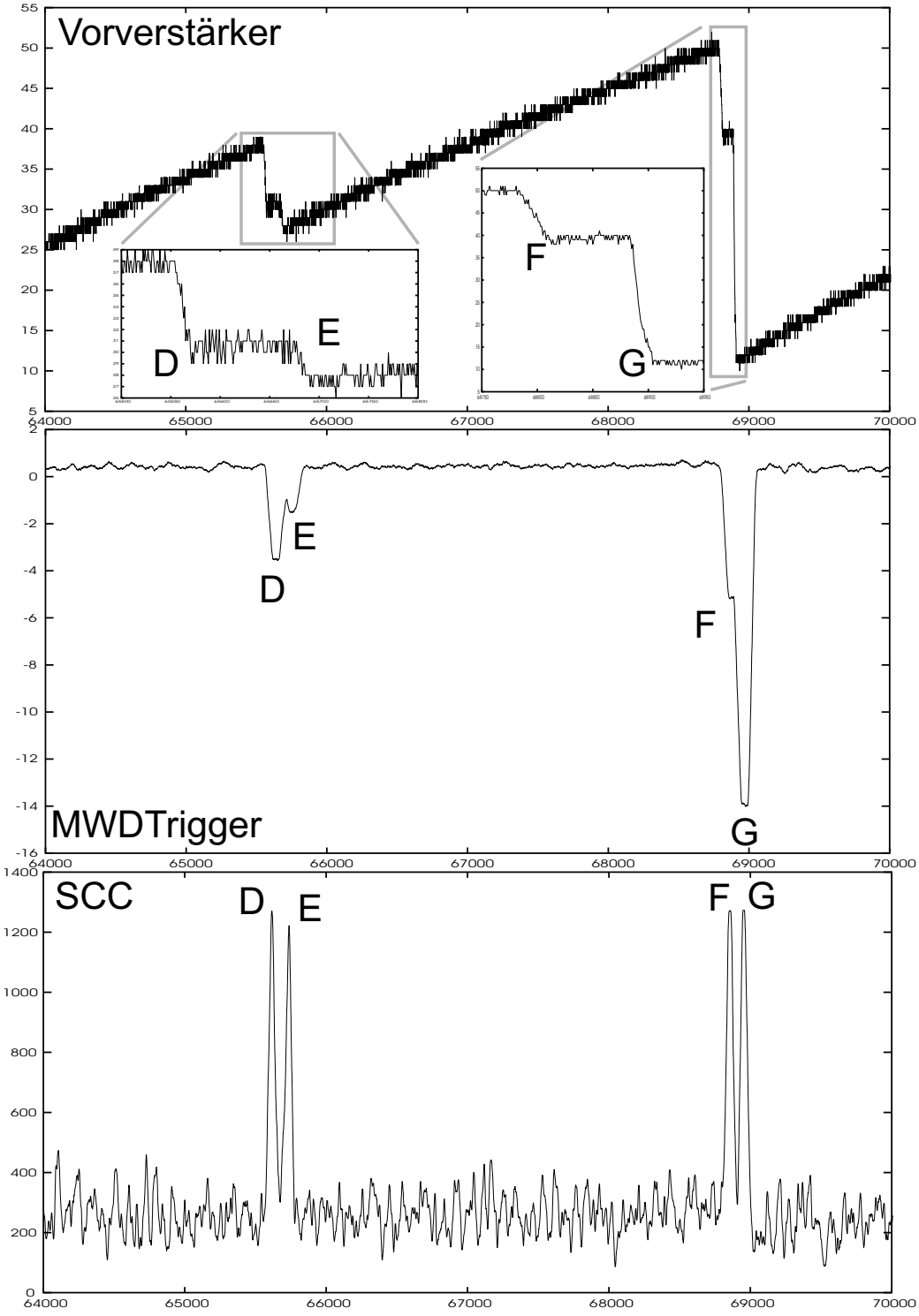


Abbildung 4.18: Zentralkontakt, 60 μs, Ausschnitt 2

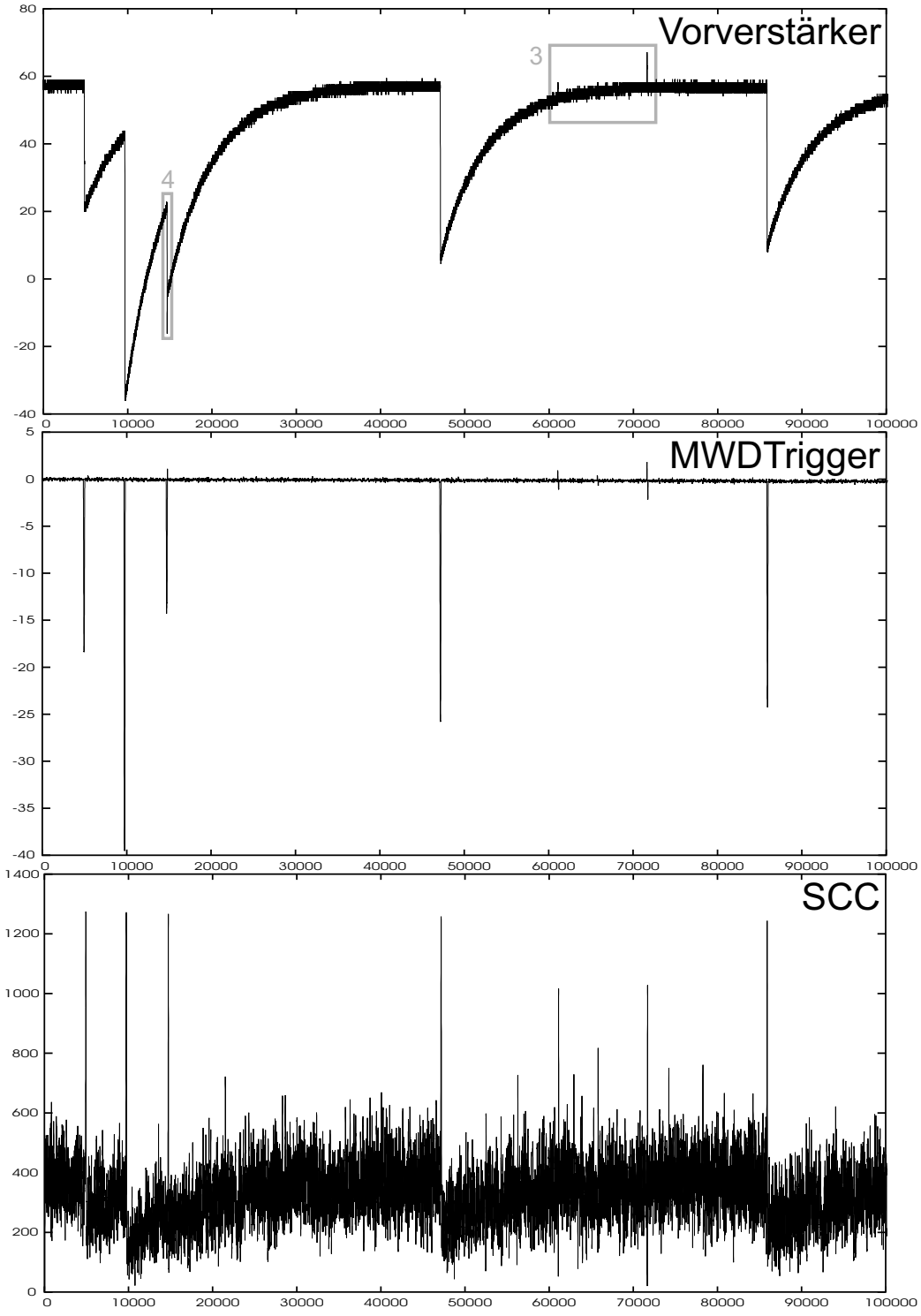


Abbildung 4.19: Segment A1, 1 ms

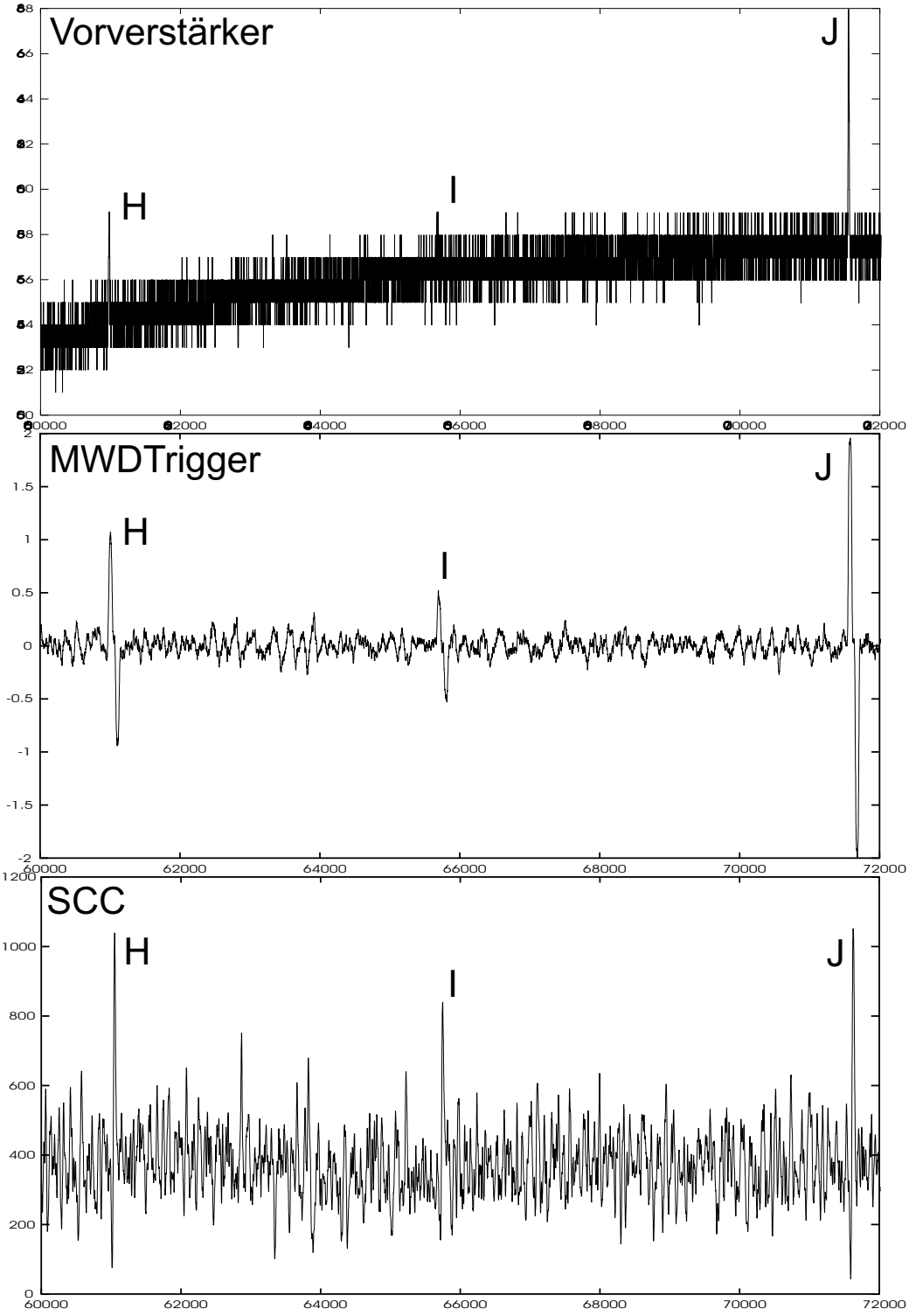


Abbildung 4.20: Segment A1, 120 μs, Ausschnitt 3

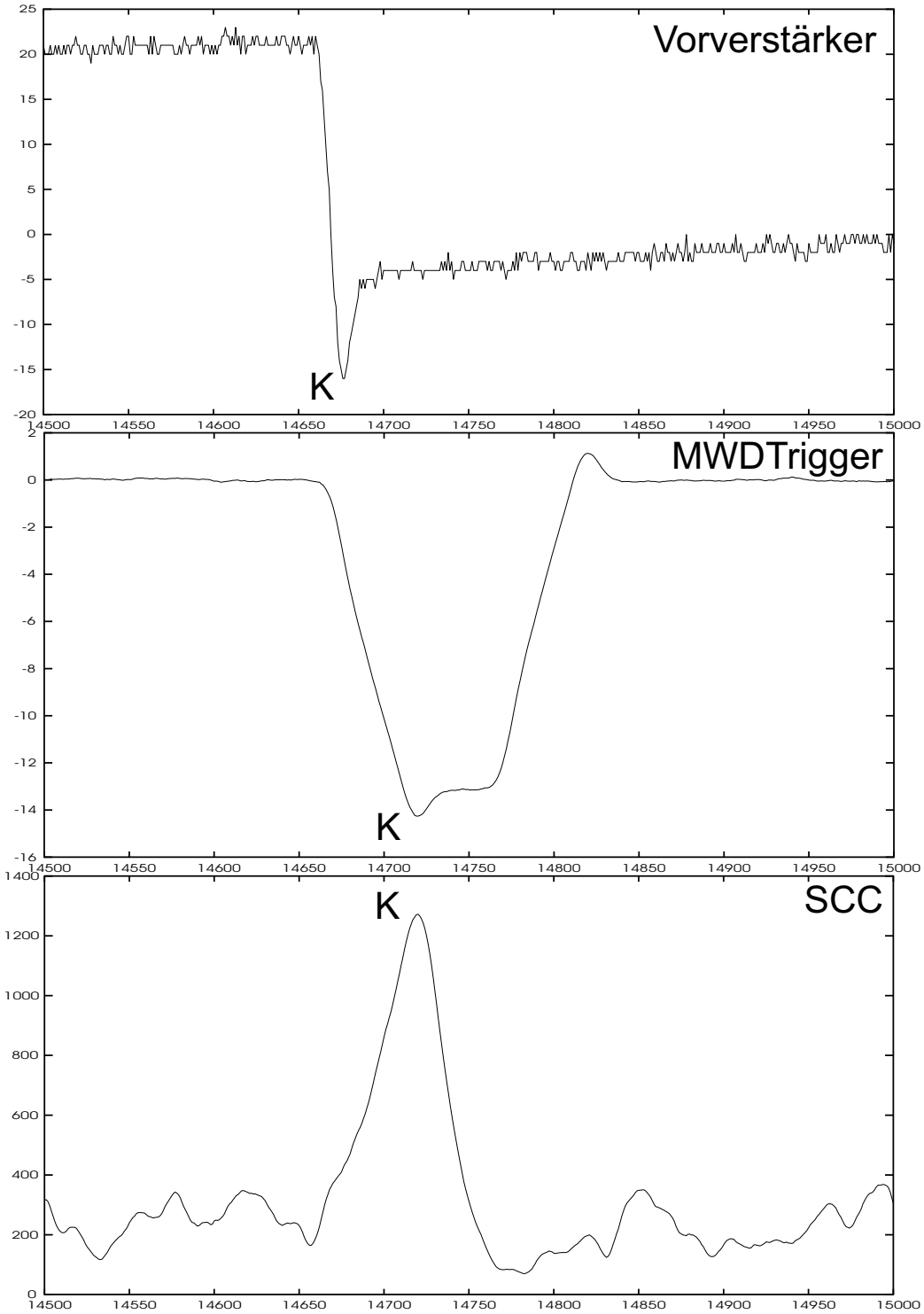


Abbildung 4.21: Segment A1, 5 μs, Ausschnitt 4

Kapitel 5

Zusammenfassung und Ausblick

Das Ziel, den MWD- und SCC-Algorithmus in C++ und VHDL zu implementieren und zu testen, konnte erreicht werden. Simulationen zeigen, dass die für das AGATA-Projekt formulierten Anforderungen an die Hardwareimplementierung (14 Bit Datenbreite, 100 MHz Taktfrequenz, 50 kHz Ereignisrate) von beiden Algorithmen erfüllt werden können. Durch den Einsatz eines schnellen FPGAs kann die Energiebestimmung sowie die Triggergenerierung digital und ohne Auflösungsverlust in Echtzeit durchgeführt werden. Nachfolgende Pulsformanalysen beruhen auf der ansteigenden Flanke des Vorverstärkersignales, so dass es mittels des digitalen Triggers möglich wird, die vom Experiment zur weiteren Auswertung zu übertragenden Daten auf diese Flanken zu reduzieren. Der relativ geringe Ressourcenbedarf der Algorithmen erlaubt es, auch mehrere Kanäle parallel mit einem heutigen FPGA zu verarbeiten.

Aus der intensiven Beschäftigung mit dem theoretischen Hintergrund digitaler Filter und Systeme, insbesondere des MWD-Agorithmus, ist das Konzept des MWDTriggers entstanden. Dieser Trigger sollte es erlauben, die Triggergenerierung von der Form und Länge der betrachteten Pulse vollständig zu entkoppeln. Ausschlaggebendes Triggerkriterium ist einzig und allein die Energie der Pulse. Der MWDTrigger ermöglicht außerdem das Aufspüren transienter Signale, womit sich die Energieauflösung des Gesamtdetektors durch Diskriminierung derselben verbessern lässt.

Neben der Programmierung der Algorithmen sind sowohl in C++ als auch in VHDL Codebibliotheken und Entwicklungsumgebungen entstanden, die sich zum Testen und Entwickeln einer Vielzahl digitaler Filter und Algorithmen eignen. Die hohe Modularität und Flexibilität des entwickelten Codes erlaubt die schnelle Umsetzung neuer Ideen oder Anpassung an andere Systeme und Anforderungen.

Der Test des MWD-Algorithmus anhand der MARS-Daten hat gezeigt,

dass mit diesem Algorithmus eine Energieauflösung ($< 3‰$ bei 662 keV) ähnlich der analoger System erreicht werden kann. Dabei können theoretisch Ereignisraten von bis zu 120 kHz ohne Auflösungsverlust verarbeitet werden. Im Hinblick auf die adaptive Ausführung des MWD-Algorithmus erlaubt es das entwickelte VHDL-MWD-Modul, während der Laufzeit Fenstergröße und Mittelungslänge zu ändern.

Der SCC-Trigger ist in der Lage, auch kleine An- oder Abstiege auf dem Signal zu detektieren. Probleme bereitet es diesem Triggeralgorithmus, Nettoladungssignale von transienten Signalen zu unterscheiden. Der Ressourcenverbrauch der Hardwareimplementation ist stark abhängig von der gewünschten Fenstergröße.

Ausblick

Eine Erweiterung der C++-Codebibliothek um eine Standard-I/O-Schnittstelle für die Datenein- und -ausgabe ermöglicht es, die Filter als kleine eigenständige Programme zu compilieren, die anschließend mittels Betriebssystempipes auf einfache Art und Weise zusammengeschlossen werden können. Dadurch wird es möglich, auch ohne großen Programmieraufwand Filter in verschiedenen Kombinationen miteinander zu testen.

Die VHDL-Module zur Programmierung des FPGAs müssen endgültig zusammengefügt und ausführlich getestet werden. Damit wird es möglich sein, die Simulationsergebnisse der VHDL-Algorithmen zu verifizieren. Der synchrone FIFO-Adressgenerator sollte so modifiziert werden, dass er den Füllstand des SDRAM-Moduls ohne zeitliche Verzögerung anzeigt.

Die adaptive Ausführung des MWD-Algorithmus wird es erlauben, die Verarbeitungszeit des MWD-Algorithmus der jeweiligen Ereignisrate anzupassen und so die Effizienz des Systems ohne einen generellen Verlust an Energieauflösung zu steigern.

Die beiden Triggeralgorithmen müssen quantitativen Tests in Kombination mit einem herkömmlichen Referenztrigger unterzogen werden, um ihre Leistungsfähigkeit besser einschätzen zu können.

Die verwendete VMEbus-FPGA-Karte kann nur eine Übergangslösung für erste Tests der Algorithmen darstellen. Eine auf den Test digitaler Algorithmen ausgelegte Karte für die Detektorauslese sollte mehrere Kriterien erfüllen:

- Zusammenfassung von ADC und FPGA auf einer Karte, um die erforderlichen Datenraten zu gewährleisten.

- Ein moderner FPGA sollte in der Lage sein, bis zu vier Kanäle parallel zu verarbeiten.
- Je größer der verwendete FPGA ist, desto flexibler kann auf neue Situationen und Probleme reagiert werden. Die Simulationen haben aber auch gezeigt, dass die Geschwindigkeitsspezifikation der verwendeten Komponenten entscheidend sein kann.
- Um die Nichtlinearitäten der ADCs ausgleichen zu können, bieten sich externe Dual-Port-Speicher zur Implementierung als Look-Up-Table an.
- Die Karte sollte mit ausreichend Datenspeicher bestückt sein, um Daten vor der Auslese zwischenspeichern oder in das Modul hochladen zu können.

Anhang A

Digitale Filter

Die folgenden Abschnitte sollen nur einen kleinen Überblick über die verwendeten Begriffe geben. Für weiterführende Informationen zur digitalen Signalverarbeitung siehe z.B. [29] oder [9].

A.1 Digitale Signalverarbeitung

Zur rechnergestützen Analyse von Daten müssen diese zunächst digitalisiert werden. Dabei findet eine Diskretisierung sowohl der Zeit als auch der Amplitude statt. Die Umwandlung wird mittels eines ADCs[1] bewerkstelligt, der das kontinuierliche Eingangssignal mit der Abtastfrequenz $f_{sampling}$ in einen ganzzahligen Wert umwandelt:

$$x[n] = [x(nT)]$$

Dabei ist $T = f_{sampling}^{-1}$ die Abtastperiode des ADCs. Im Folgenden soll die Diskretisierung der Amplitude zur Vereinfachung außer Acht gelassen werden, so dass gilt:

$$x[n] = x(nT) = \int_{-\infty}^{\infty} dt\, x(t)\, \delta(t - nT) \tag{A.1}$$

Kontinuierliche und zeitdiskrete Fouriertransformation

Effekte, die diese Diskretisierung mit sich bringt, lassen sich am Besten im Frequenzraum betrachten. Dafür verwenden wir die kontinuierliche sowie

[1] analog-digital converter

zeitdiskrete Fouriertransformation. Diese sind definiert als kontinuierliche Fouriertransformation

$$X_k(\omega) := \int_{-\infty}^{\infty} dt\, x(t)\, \exp(-i\omega t) \tag{A.2}$$

mit der Rücktransformation

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} d\omega\, X_k(\omega)\, \exp(i\omega t)$$

und zeitdiskrete Fouriertransformation

$$X_d(\Omega) := \sum_{n=-\infty}^{\infty} x[n]\, \exp(-i\Omega n) \tag{A.3}$$

mit der zeitdiskreten Rücktransformation

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} d\Omega\, X_d(\Omega)\, \exp(i\Omega n)\ ,$$

wobei $x(t), x[n] \in C$. Sind $x(t)$ bzw. $x[n]$ reellwertig, so ist $X_k(\omega) = X_k^*(-\omega)$ bzw. $X_d(\Omega) = X_d^*(-\Omega)$. Der positive Frequenzraum, $\omega, \Omega \geq 0$, enthält in diesem Fall alle Informationen über das Signal. Dabei ist der Betrag der Fouriertransformierten eine gerade Funktion, die Phase eine ungerade.

Abtast-Theorem

Aus Glg. A.3 folgt $\Omega \in [-\pi, \pi]$, da $\exp(-i\Omega n)$ 2π-periodisch ist. Die Diskretisierung der Zeitachse führt also zu einer Einschränkung des Signals im Frequenzraum. Dabei ist $\Omega = \frac{\omega}{f_{sampling}} = \frac{2\pi f}{f_{sampling}}$, womit wir $f \in [-\frac{f_{sampling}}{2}, \frac{f_{sampling}}{2}]$ erhalten. Diese Grenzfrequenz $\frac{f_{sampling}}{2}$ wird auch als Nyquistfrequenz $f_{Nyquist}$ bezeichnet. Ist das kontinuierliche Signal schon vor der Diskretisierung auf diesen Bereich bandbreitenbeschränkt, so sind die kontinuierliche und die zeitdiskrete Fouriertransformierte gleich, also $X_k(\omega) = X_d(\Omega = \frac{\omega}{f_{sampling}})$. In diesem Fall lässt sich das kontinuierliche Signal vollständig aus dem diskretisierten zurückgewinnen (Nyquist- / Abtast-Theorem).

Erfüllt das kontinuierliche Signal nicht diese Bandbreitenbeschränkung, so werden Signalanteile, deren Frequenzen oberhalb der Nyquistfrequenz liegen, durch die Diskretisierung in diesen Bereich abgebildet (Aliasing). Man kann

sich dies sowohl grafisch (Abb. A.1) wie auch mathematisch veranschaulichen. Dazu betrachte man zwei Komponenten des Signals mit den Frequenzen f_1 und $f_2 = f_1 + af_{sampling}$ mit $a \in Z$.

$$x_1(t) = \sin(2\pi f_1 t) \qquad x_2(t) = \sin(2\pi(f_1 + af_{sampling})t)$$

Diskretisiert man $x_2(t)$ zu $x_2[n]$ mit der Abtastfrequenz $f_{sampling}$ so gilt für alle n:

$$\begin{aligned}
x_2[n] &= \int_{-\infty}^{\infty} dt\, \sin(2\pi(f_1 + af_{sampling})t)\, \delta\left(t - \frac{n}{f_{sampling}}\right) \\
&= \sin\left(2\pi(f_1 + af_{sampling})\frac{n}{f_{sampling}}\right) \\
&= \sin\left(2\pi f_1 \frac{n}{f_{sampling}} - 2\pi an\right) \\
&= \sin\left(2\pi f_1 \frac{n}{f_{sampling}}\right) \\
&= \int_{-\infty}^{\infty} dt\, \sin(2\pi f_1 t)\, \delta\left(t - \frac{n}{f_{sampling}}\right) = \quad x_1[n]
\end{aligned}$$

D.h., der Signalanteil mit der Frequenz f_1 wird zur Frequenz f_2 hinzuaddiert (Abb. A.2). Kontinuierliche und zeitdiskrete Fouriertransformierte unterscheiden sich jetzt, was zu einem unterschiedlichen zeitlichen Signal führt.

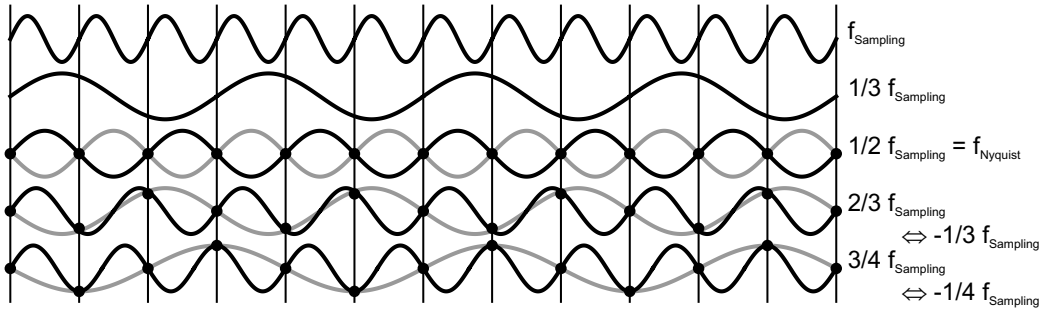


Abbildung A.1: Eine Frequenz f oberhalb der Nyquistfrequenz $f_{Nyquist}$ hinterlässt dasselbe diskretisierte Signal wie die Frequenz $f - f_{Sampling}$.

A.2 Digitale Filter

- **Signale** x[n]

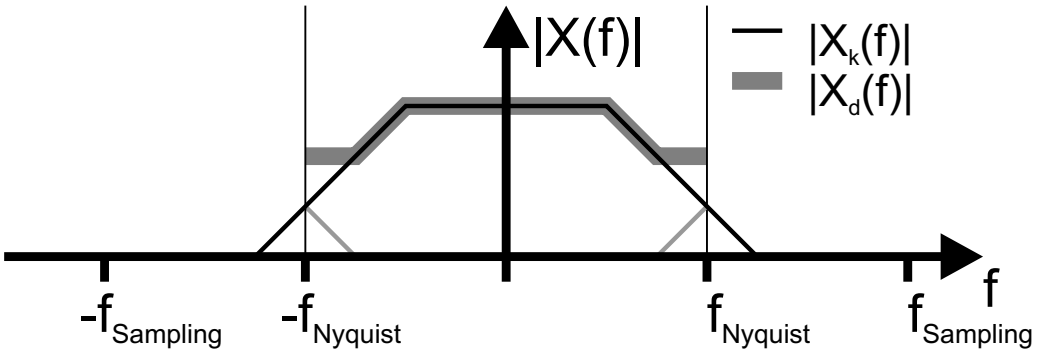


Abbildung A.2: Fouriertransformierte eines kontinuierlichen Signals sowie dessen zeitdiskrete Version. Frequenzanteile höher der Nyquistfrequenz werden in den beschränkten Bandbreitenbereich abgebildet (Aliasing).

Es seien:

- $x[n]$ ... das zeitdiskrete Eingangssignal und
- $y[n]$ ... das zeitdiskrete Ausgangssignal des Filters.

Ein zeitdiskretes Signal lässt sich als Summe gewichteter Einheitsimpulse $\delta[n] = \begin{cases} 1 & \text{für } n = 0 \\ 0 & \text{für } n \neq 0 \end{cases}$ darstellen:

$$x[n] = \sum_{m=-\infty}^{\infty} x[m]\delta[n-m] \tag{A.4}$$

- **Filter $\mathcal{F}_n$**
  $y[n] = \mathcal{F}_n(x[\cdot])$
  $x[\cdot] = \{x[i] : -\infty < i < +\infty\}$

  Ein Filter kann durch folgende Eigenschaften charakterisiert werden:

  - **Kausalität**
    $y[n] = \mathcal{F}_n(x'[\cdot])$
    $x'[\cdot] = \{x[i] : -\infty < i \leq n\}$
    Das Ausgangssignal des Filters wird nur durch Eingangswerte aus Vergangenheit und Gegenwart bestimmt. Reale (Hardware-) Filter haben diese Eigenschaft immer.
  - **Linearität**
    Proportionalität: $\mathcal{F}_n(c \cdot x[\cdot]) = c \cdot \mathcal{F}_n(x[\cdot])$

Superposition: $\mathcal{F}_n(x_1[\cdot] + x_2[\cdot]) = \mathcal{F}_n(x_1[\cdot]) + \mathcal{F}(x_2[\cdot])$
$\forall\, n \geq 0$, $c = konst.$

- **Zeitinvarianz**
  $\mathcal{F}_{n-N}(x[\cdot]) = y[n-N]$
  Eine zeitliche Verschiebung der Eingangssignale führt zu einer gleichartigen Verschiebung der Ausgangswerte.

- **Speicherfreiheit**
  $y[n] = \mathcal{F}_n(x[n])$
  Nur der gegenwärtige Eingangswert bestimmt das Ausgangssignal.

Eine wichtige Unterklasse sind lineare, zeitinvariante Filter:

- **LTI-Filter** (linear-time-invariant)
  Für diese lassen sich weitere Eigenschaften definieren:

  - **Differenzengleichung**
    $y[n] = \sum_{i=-\infty}^{+\infty} b_i\, x[n-i] - \sum_{j=1}^{+\infty} a_j\, y[n-j]$
    b_i ... feedforward Koeffizienten
    a_j ... feedback Koeffizienten
    kausal für $b_i = 0\ \forall\, i < 0$
    Die Differenzengleichung ist das zeitdiskrete Analogon zur Differentialgleichung.

  - **Impulsantwort** $h[n]$
    Antwort des Filters auf einen Delta-Puls $\delta[n] = \begin{cases} 1 & \text{für } n = 0 \\ 0 & \text{für } n \neq 0 \end{cases}$
    $h[n] = \mathcal{F}_n(\delta[\cdot])$

  - **Ordnung**
    $y[n] = \sum_{i=0}^{M} b_i\, x[n-i] - \sum_{j=1}^{N} a_j\, y[n-j]$
    Ordnung endlich, wenn N, M endlich.
    Die Ordnung wird durch den größeren der beiden Werte N oder M bestimmt.

  - **Rekursivität**
    $a_j \neq 0$ für ein oder mehrere j
    dies wird *feedback* genannt
    nicht-rekursive LTI-Filter: *FIR*-Filter (finite-impulse-response) - besitzen eine endliche Impulsantwort
    rekursive LTI-Filter: *IIR*-Filter (infinite-impulse-response)

– **Stabilität**
$\lim_{n \to +\infty} h[n] \to 0$
ein beschränktes Eingangssignal liefert ein beschränktes Ausgangssignal.

– **Darstellung durch Faltungssumme**
LTI-Systeme lassen sich durch eine Faltungssumme aus Eingangswerten und Impulsantwort beschreiben:

$$\begin{aligned} y[n] &= \mathcal{F}_n(x[n]) \\ &= \mathcal{F}_n\left(\sum_{m=-\infty}^{\infty} x[m]\delta[n-m]\right) \\ &= \sum_{m=-\infty}^{\infty} \mathcal{F}_n\left(x[m]\delta[n-m]\right) \\ &= \sum_{m=-\infty}^{\infty} x[m]\mathcal{F}_n(\delta[n-m]) \\ &= \sum_{m=-\infty}^{\infty} x[m]h[n-m] \\ &= x[m] * h[n] \end{aligned}$$

Dabei wurde im ersten Schritt Glg. A.4 verwendet. Schritt zwei und drei beruhen auf der Linearität des Systems, Umformung vier auf dessen Zeitinvarianz.
Die Form von Impulsantwort und Differenzengleichung ist gleich für FIR-Filter.

– **Seriell- und Parallelschaltung**
Werden zwei LTI-Filter hintereinander geschaltet, so ist die Impulsantwort des Gesamtsystems die Faltungssumme der beiden Einzelimpulsantworten. Für parallel geschaltete Filter addieren sich die beiden Impulsantworten.

A.3 z-Transformation

Definition

Die z-Transformation ist die diskrete Version der Laplace-Transformation und wird zur Lösung von Differenzengleichungen, der diskreten Form der Differentialgleichung, genutzt. Sie ist für eine Folge $f = \{f_0, f_1, ...\}$ definiert

als

$$F(z) := \sum_{n=0}^{\infty} f_n z^{-n} \ , \tag{A.5}$$

falls diese (Potenz-)Reihe absolut konvergiert, d.h. $\sum_{n=0}^{\infty} |f_n z^{-n}| < \infty$. Dies ist der Fall für alle z mit $|z| > r$ und $|f_n| \leq \text{const.} \cdot r^n$. r ist der Konvergenzradius der Reihe. $F(z) = \mathcal{Z}\{f_n\}$ wird die z-Transformierte von $\{f_n\}$ genannt und ist komplexwertig. Obige Form bezeichnet man als unilateral, daneben gibt es noch die bilaterale z-Transformation

$$F(z) := \sum_{n=-\infty}^{\infty} f_n z^{-n} \ .$$

Die inverse z-Transformation wird beschrieben durch

$$f_n = \frac{1}{2\pi i} \oint_C dz \, F(z) \, z^{n-1} \ . \tag{A.6}$$

Eigenschaften

Wichtige Eigenschaften der z-Transformation sind unter anderem:

- *Linearität*:

$$\mathcal{Z}\{af_n + bg_n\} = a\mathcal{Z}\{f_n\} + b\mathcal{Z}\{g_n\}$$

- *Faltungsregel*:

$$\mathcal{Z}\{f_n * g_n\} = \mathcal{Z}\{f_n\} \cdot \mathcal{Z}\{g_n\}$$

Signal	z-Transformierte	ROC[2]
$\delta[n-k]$	z^{-k}	$z \in C$
$u[n]$	$\frac{z}{z-1}$	$\|z\| > 1$
$\alpha^n \cdot u[n]$	$\frac{z}{z-\alpha}$	$\|z\| > \|\alpha\|$

Tabelle A.1: Einige Funktionen und ihre z-Transformierten (aus [30])

Weitere Eigenschaften der z-Transformation sowie Tabellen mit bekannten Transformationen finden sich zum Beispiel unter [30] oder in [5].

z-Transformation und diskrete Signaltheorie

Transferfunktion

Die z-Transformation ist ein wichtiges Hilfsmittel für die Konstruktion und die Analyse digitaler LTI-Systeme. Aufgrund der Faltungsregel lässt sich aus z-transformierter Impulsantwort und Eingangssignal durch eine einfache Multiplikation das Ausgangssignal eines solchen Systems gewinnen. Die z-Transformierte der Impulsantwort wird als *Transferfunktion* bezeichnet.

Pol- und Nullstellendiagramm

Anhand eines Pol- und Nullstellendiagramms (pole-zero-plots), einer Auftragung der Pol- und Nullstellen der Transferfunktion in der komplexen Ebene (z-Ebene, Abb. A.3), lassen sich Kausalität und Stabilität sowie die Frequenzantwort $H(\omega)$ eines Systems schnell abschätzen.

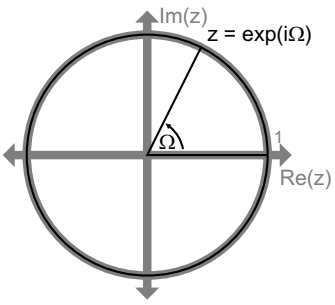


Abbildung A.3: z-Ebene mit Einheitskreis

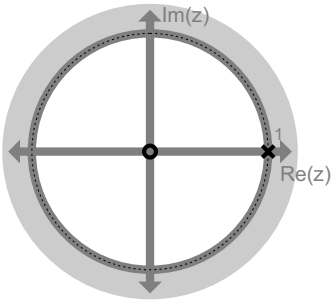


Abbildung A.4: idealer Integrator

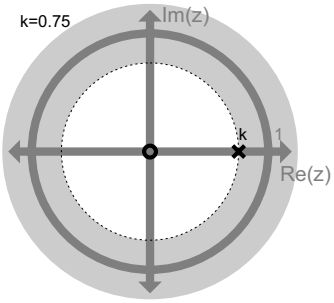


Abbildung A.5: ladungssensitiver Vorverstärker

Die grau hinterlegten Flächen der pole-zero-plots zeigen die Konvergenzgebiete der z-Transformierten auf. Nullstellen sind durch Kreise gekennzeichnet, Pole durch ein Kreuz. Das Konvergenzgebiet wird durch die Polstellen begrenzt. Ein System ist

kausal, wenn das Konvergenzgebiet außerhalb des Konvergenzradius liegt (im Fall obiger unilateraler Definition A.5 der z-Transformation immer gegeben),

stabil, wenn das Konvergenzgebiet den Einheitskreis mit einschließt.

Als Beispiel sei die Impulsantwort eines idealen Integrators $i[n] = u[n]$ aufgeführt. Die zugehörige Transferfunktion hat die Form

$$I(z) = \sum_{0}^{\infty} \left(\frac{1}{z}\right)^n = \frac{z}{z-1} \text{ für } |z| > 1 \ ,$$

d.h. die Reihe ist nur konvergent für alle Punkte z außerhalb des Einheitskreises (Abb. A.4). Dieser Filter ist zwar kausal, aber instabil.

Im Gegensatz dazu ist der ladungssensitive Vorverstärker $h[n] = k^n \cdot u[n]$ sowohl kausal als auch stabil. Seine Transferfunktion lautet

$$H(z) = \sum_{0}^{\infty} \left(\frac{k}{z}\right)^n = \frac{z}{z-k} \text{ für } |z| > k \text{ mit } 0 < k < 1 \ .$$

Der Einheitskreis befindet sich also im Konvergenzgebiet (Abb. A.5). Dies erlaubt außerdem die Berechnung der Frequenzantwort $H(\Omega)$ des Vorverstärkers (Abb. A.6 für verschiedene k).

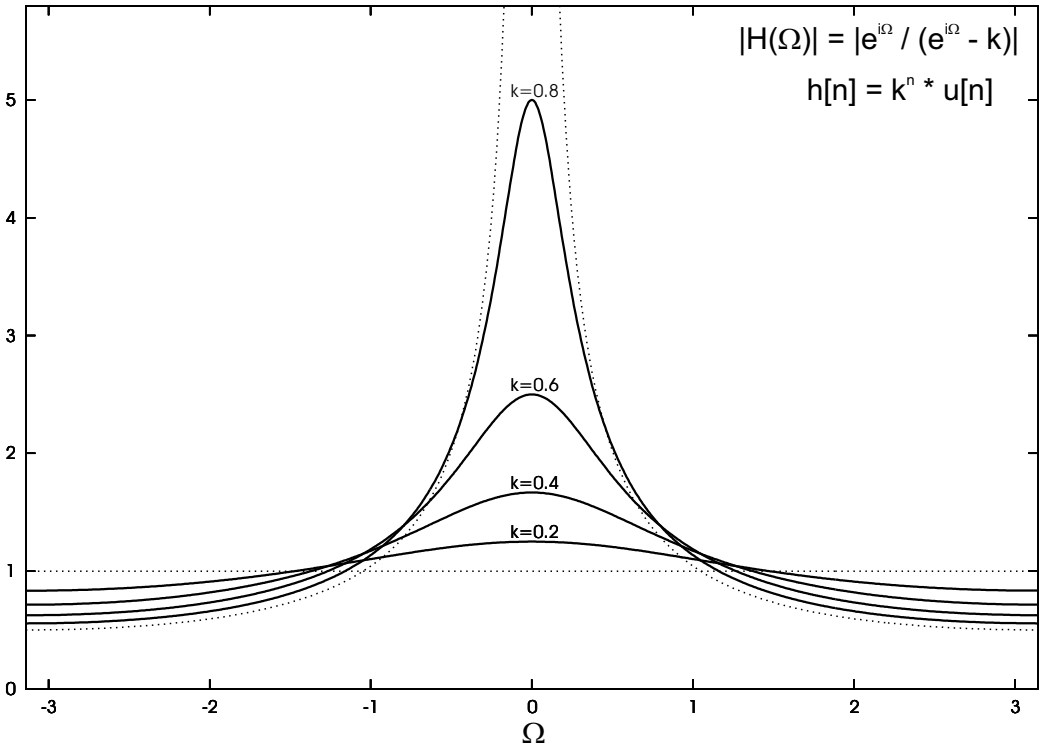


Abbildung A.6: Frequenzantworten eines ladungssensitiven Vorverstärkers für verschiedene Zeitkonstanten τ. $k = \exp(-\tau f_{sampling})$. Die gepunkteten Kurven ergeben sich für $k = 1$ und $k = 0$.

Frequenzantwort

Die Frequenzanwort $H(\Omega)$ erhält man aus $H(z)|_{z=\exp(i\Omega)}$, also der Beschränkung der Transferfunktion auf den Einheitskreis. Sie ist die zeitdiskrete Fouriertransformierte der Impulsantwort und beschreibt die Frequenzabhängigkeit des Filters.

Anhang B

Ladungssensitive Vorverstärker

Ein Vorverstärker dient dazu, ein schwaches Detektorsignal so weit aufzubereiten bzw. zu verstärken, dass eine Übertragung des Signals über größere Distanzen (mehrere Meter) möglich wird, ohne dass dieses im Rauschen verschwindet. Daher ist der Vorverstärker meist ganz in der Nähe des Detektors montiert, im Fall der AGATA-Detektoren werden sie sich teilweise (die Eingangs-FETs) innerhalb des kalten Bereichs des Kryostaten befinden, was neben kurzen Signalwegen auch zu einer Minderung des thermischen Eigenrauschens führt.

Man unterscheidet drei Arten von Vorverstärkern (entnommen [23]):

- Stromverstärker,
- Spannungsverstärker und
- ladungssensitive Verstärker.

Der Stromverstärker benötigt ein niederohmiges Eingangssignal, so dass er mit einem Halbleiterdetektor nicht verwendet werden kann. Der Einsatz eines Spannungsverstärkers bereitet insofern Probleme, als dass die Ausgangsspannung des Detektors V_{det} von seiner Kapazität abhängt, die aufgrund der Leckströme im Detektor sehr temperaturabhängig ist. Es gilt $V_{det} = \frac{Q_{WW}}{C_{det}(T)}$.

Ladungssensitiver Vorverstärker
Ein ladungssensitiver Vorverstärker zeigt diese Kapazitäts- und damit Temperaturabhängigkeit nicht. Er sammelt die von der Wechselwirkung im Detektor generierten freien Ladungsträger und erzeugt daraus ein Spannungssignal. Der Nachteil dieses Verstärkertyps ist, dass der Sammelkondensator entleert werden muss, um einer Sättigung des Ausgangssignales vorzubeugen.

Dies geschieht meist kontinuierlich durch einen zur Kapazität parallelen ohmschen Widerstand. Dadurch entsteht der für diesen Verstärkertyp charakteristische exponentiell abfallende Schwanz der Pulse (siehe Abb. B.1). Dies kann bei zu hoher Zählrate zu sogenannten pile-ups führen, d.h. das folgende Signal fällt noch in die abfallende Flanke des vorherigen Pulses, wodurch die Pulshöhe verändert wird (siehe Abb. B.2). Daneben tritt der Effekt des sogenannten ballistic deficit (siehe Anhang B.2) auf.

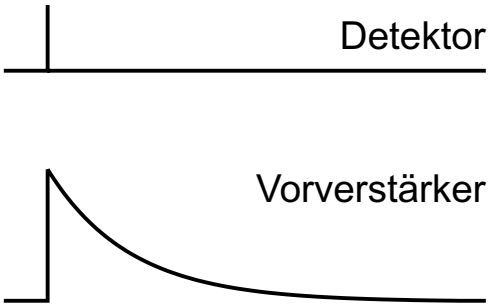


Abbildung B.1: Vereinfachte Darstellung der Antwort eines ladungssensitiven Vorverstärkers auf einen δ-Puls.

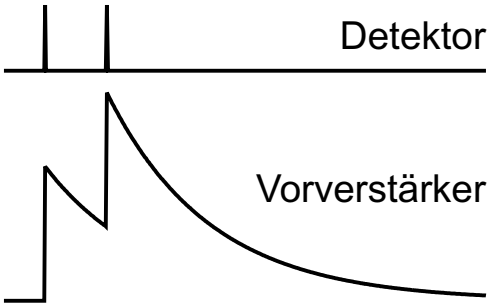


Abbildung B.2: Pile-Up zweier δ-Pulse.

Eine weitere Möglichkeit Sättigung zu vermeiden, ist, den Kondensator von Zeit zu Zeit (wenn seine Ladung einen bestimmten Grenzwert überschreitet oder nach einer festgelegten Zeitspanne) z.B. über einen Transistor komplett zu entladen. Dies kann bis zu 10 μs dauern, was die Totzeit des Detektors erhöht[1], und wird daher für Experimente mit hoher Ereignisrate nicht verwendet.

B.1 Impulsantwort eines ladungssensitiven Vorverstärkers

Differentialgleichung des Vorverstärkers

Abbildung B.3 stellt die einfachste Form eines Halbleiterdetektors und eines ladungssensitiven Vorverstärkers dar. Der Detektor selbst kann vereinfacht als Stromquelle betrachtet werden. Der Vorverstärker besteht aus einem Operationsverstärker (OpAmp), einer Kapazität und einem dazu parallelen ohm-

[1] Angaben entnommen privater Korrespondenz mit Dr. George Pascovici, IKP, Universität Köln.

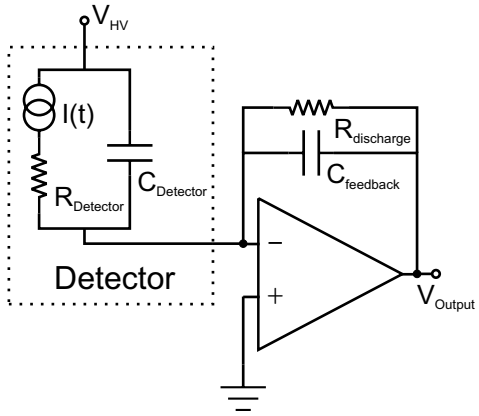


Abbildung B.3: Schematische Darstellung eines Detektors mit ladungssensitiven Vorverstärker an einer Segmentelektrode. Der Vorverstärker ist DC-gekoppelt. Für den Zentralkontakt benötigt man aufgrund der Hochspannung eine AC-Kopplung über einen Kondensator.

schen Widerstand. Ein Operationsverstärker ist ein Differenzenverstärker, der im Idealfall Spannungsunterschiede zwischen invertierendem [-] und nichtinvertierendem [+] Eingang unendlich verstärkt, dabei aber keinen Strom aus den Eingängen bezieht [18].

Mittels des ersten Kirchhoffschen Gesetzes (Knotenregel) für den invertierenden Eingang des Operationsverstärkers erhält man folgende mathematische Beschreibung der Vorverstärkerschaltung:

$$\begin{aligned} 0 &= I_{det}(t) + I_{C_{fb}}(t) + I_{R_{dis}}(t) \\ &= I_{det}(t) + C_{fb}\,\frac{dV_{out}(t)}{dt} + \frac{1}{R_{dis}}\,V_{out}(t)\ , \end{aligned}$$

woraus sich eine inhomogene Differentialgleichung erster Ordnung für das Ausgangssignal $V_{out}(t)$ ergibt:

$$\frac{dV_{out}(t)}{dt} + \frac{1}{R_{dis}\,C_{fb}}\,V_{out}(t) = -\frac{1}{C_{fb}}\,I_{det}(t) \tag{B.1}$$

Die allgemeine Lösung einer solchen Gleichung ist eine Superposition einer beliebigen partikulären Lösung der inhomogenen Gleichung und Lösungen der homogenen Gleichung.

Zur Verbesserung der Übersichtlichkeit sind im Folgenden die beschreibenden Indizes weggelassen worden.

Lösung der homogenen Gleichung

Die homogene Gleichung lautet

$$\frac{dV_{hom}(t)}{dt} + \frac{1}{R\,C}\,V_{hom}(t) = 0\ , \tag{B.2}$$

woraus folgt

$$\begin{aligned}
\frac{dV_{hom}(t)}{dt} &= -\frac{1}{R\,C}\,V_{hom}(t) \\
\frac{dV_{hom}(t)}{V_{hom}(t)} &= -\frac{1}{R\,C}\,dt \\
\int dV_{hom}(t)\,\frac{1}{V_{hom}(t)} &= -\frac{1}{R\,C}\int dt \\
\ln\left(V_{hom}(t)\right) &= -\frac{1}{R\,C}\,t + konst.\ ,
\end{aligned}$$

und damit die Lösung

$$V_{hom}(t) = K \cdot \exp\left(-\frac{1}{R\,C}\,t\right)\ , \tag{B.3}$$

wobei $K \in R$ die Einheit einer Spannung [V] hat.

Partikuläre Lösung der inhomogenen Gleichung

Ein Ansatz zur Lösung der inhomogenen Gleichung B.1 lautet

$$V_{part}(t) = v(t) \cdot V_{hom}(t) \tag{B.4}$$

und dessen Ableitung

$$V'_{part}(t) = v'(t) \cdot V_{hom}(t) + v(t) \cdot V'_{hom}(t)\ ,$$

wobei $v(t)$ und $v'(t)$ $[\frac{1}{s}]$ zunächst unbekannt sind. Eingesetzt in Gleichung B.1 ergibt sich

$$\begin{aligned}
-\frac{1}{C}\,I(t) &= v'(t) \cdot V_{hom}(t) + v(t) \cdot V'_{hom}(t) + v(t) \cdot \frac{1}{R\,C}\,V_{hom}(t) \\
&= v'(t) \cdot V_{hom}(t) + v(t) \cdot \underbrace{\left(V'_{hom}(t) + \frac{1}{R\,C}\,V_{hom}(t)\right)}_{=0 \text{ (Glg. B.2)}}
\end{aligned}$$

$$= \quad v'(t) \cdot V_{hom}(t)$$

und daraus

$$v'(t) \quad = \quad -\frac{1}{C}\,\frac{I(t)}{V_{hom}(t)}$$

bzw.

$$\begin{aligned} v(t) &= -\frac{1}{C}\int_0^t dt'\,\frac{I(t')}{V_{hom}(t')} \\ &= -\frac{1}{C\,K}\int_0^t dt'\,I(t')\,\exp\left(\frac{1}{R\,C}\,t'\right)\,. \end{aligned}$$

Dies wiederum eingesetzt in den Ansatz B.4 führt zu

$$\begin{aligned} V_{part}(t) &= -\frac{1}{C}\int_0^t dt'\,I(t')\cdot\exp\left(\frac{1}{R\,C}\,t'\right)\cdot\exp\left(-\frac{1}{R\,C}\,t\right) \\ &= -\frac{1}{C}\int_0^t dt'\,I(t')\cdot\exp\left(-\frac{1}{R\,C}\,(t-t')\right)\,, \end{aligned} \tag{B.5}$$

einer partikulären Lösung der inhomogenen Differentialgleichung B.1.

Da $I(t) = 0$ für $t < 0$ und unter Verwendung der Stufenfunktion $u(t) = \begin{cases} 0 & \text{für } t < 0 \\ 1 & \text{für } t \geq 0 \end{cases}$, können wir die Lösung B.5 umschreiben zu

$$\begin{aligned} V(t) &= -\frac{1}{C}\int_{-\infty}^t dt'\,I(t')\cdot\exp\left(-\frac{1}{R\,C}\,(t-t')\right) \\ &= -\frac{1}{C}\int_{-\infty}^{\infty} dt'\,I(t')\cdot u(t-t')\cdot\exp\left(-\frac{1}{R\,C}\,(t-t')\right)\,, \end{aligned}$$

was nichts anderes darstellt als eine Faltung

$$V(t) = I(t) * \underbrace{\left(-\frac{1}{C}\cdot u(t)\cdot\exp\left(-\frac{1}{R\,C}\,t\right)\right)}_{h(t)} \tag{B.6}$$

des Eingangsignales $I_{Detector}(t)$ mit der Funktion $h(t)$.

Impulsantwort

Diese Funktion $h(t)$ wird die *Impulsantwort* des Vorverstärkers genannt. Sie beschreibt das System vollständig. Das Ausgangssignal des Vorverstärkers lässt sich auf einfache Weise als Faltung des Eingangssignales mit dieser Impulsantwort berechnen.

Benutzt man die Zeitkonstante $\tau = R\,C$ [s] so erhält man für die Impulsantwort

$$h_\tau(t) = -\frac{1}{C} \cdot u(t) \cdot \exp\left(-\frac{1}{\tau}\,t\right) \tag{B.7}$$

und damit

$$V(t) = I(t) * h_\tau(t)\ . \tag{B.8}$$

Diskrete Zeitdomäne

Im zeitlich diskreten Fall wird $h_\tau(t)$ zu

$$h_\tau[n] = h_\tau(nT) = -\frac{1}{C} \cdot u[n] \cdot \exp\left(-\alpha\, n\right) \tag{B.9}$$

mit der diskreten Stufenfunktion $u[n] = \left\{ \begin{array}{ll} 0 & \text{für } n < 0 \\ 1 & \text{für } n \geq 0 \end{array} \right.$, $n \in Z$, $T = \frac{1}{f_{sampling}}$ [s] der Abtastperiode und entsprechend $f_{sampling}$ [Hz] der Abtastfrequenz des ADCs und $\alpha = (\tau \cdot f_{sampling})^{-1}$.

Idealer Vorverstärker = Idealer Integrator

Zum Vergleich noch die Impulsantwort eines idealen Vorverstärkers, der einem reinen Integrator entsprechen würde.

Um dies zu erreichen wird der ohmsche Widerstand aus der Schaltung entfernt (d.h. $R \to +\infty$), was eine Entladung des Kondensators verhindert. Aus

$$R \to +\infty \Longrightarrow \left\{ \begin{array}{c} \tau \to +\infty \\ \alpha \to 0 \end{array} \right\} \Longrightarrow \left\{ \begin{array}{c} \exp\left(-\frac{1}{\tau}\right) \\ \exp\left(-\alpha\right) \end{array} \right\} \to 1$$

folgt somit für die Impulsantwort

$$\begin{aligned} i(t) &= -\frac{1}{C} \cdot u(t) \\ i[n] &= -\frac{1}{C} \cdot u[n]\ . \end{aligned}$$

Frequenzbetrachtung

Die oben hergeleitete Impulsantwort $h(t)$ kann nicht nur als Integrator, sondern auch als Tiefpass 1.Ordnung aufgefasst werden. Hohe Frequenzen werden stärker gedämpft bzw. schwächer verstärkt als niedrige (siehe Abb. A.6). Dies führt dazu, dass ein stufenförmiges Eingangssignal $A \cdot u(t)$ geglättet wird und zu einem Anstieg mit zeitlicher Dauer führt. Anhand von Abbildung B.4 kann man erkennen, dass diese Vorverstärkerschaltung nur für Pulse mit einer Dauer wesentlich kürzer als die Zeitkonstante τ annähernd integrative Eigenschaften hat.

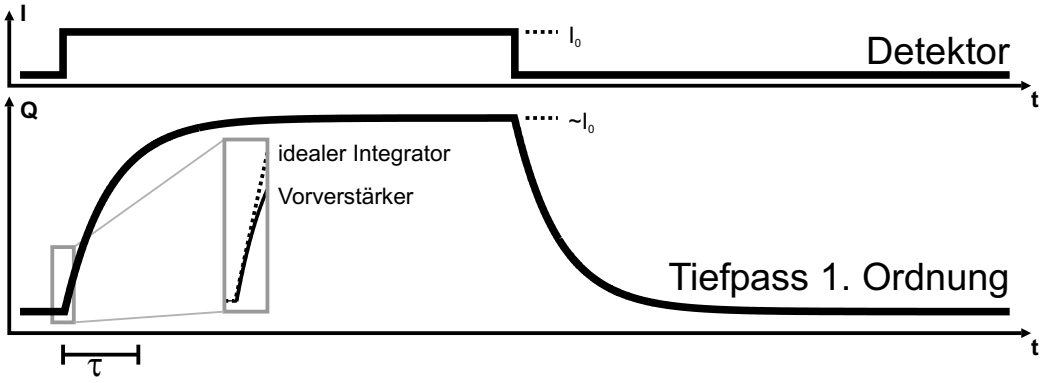


Abbildung B.4: Antwort eines Tiefpass 1.Ordnung auf ein Stufensignal bzw. einen sehr langen Rechteckpuls ($\Delta t >> \tau$). Der Wert, dem sich das Vorverstärkersignal asymptotisch annähert, ist proportional zur Eingangspulshöhe, während der Wert für kurze Pulse $\Delta t << \tau$ ungefähr proportional zur Fläche unter dem Puls ist (Integration, siehe vergrößerten Ausschnitt).

Obige Ableitung der Impulsantwort für den ladungssensitiven Vorverstärker beruht auf einem Idealbild des Operationsverstärkers. Völlig außer Acht gelassen wurde unter anderem die Frequenzabhängigkeit der Verstärkung durch den Operationsverstärker. Sie nimmt mit steigender Frequenz ab, so dass dadurch weitere Tiefpasseigenschaften in der Gesamtschaltung hinzukommen.

Die durch all diese Effekte hervorgerufene endliche Anstiegszeit des Ausgangssignals auf einen Stufenpuls ist eine Auswirkung der Bandbreitenbeschränkung des Vorverstärkers.

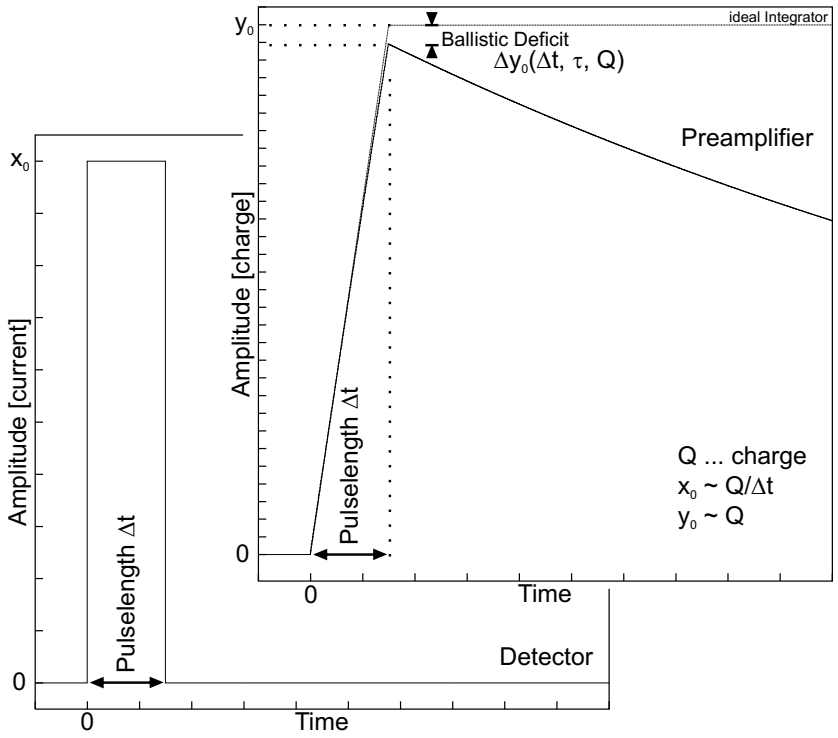


Abbildung B.5: Ausgangssignal eines ladungssensitiven Vorverstärkers für einen Rechteckpuls.

B.2 Ballistic Deficit

Das so genannte "Ballistic Deficit" (siehe auch [17]) folgt zwangsläufig aus dem prinzipiellen Aufbau eines ladungssensitiven Vorverstärkers. Bezeichnet wird damit die Eigenheit, dass die Amplitude des Vorverstärker-Ausgangssignales einer einzelnen Wechselwirkung im Detektor bei gleicher deponierter Energie von der Dauer der Ladungssammlung und damit letztendlich vom Ort der Wechselwirkung im Detektor abhängt. Je mehr Zeit die vollständige Ladungssammlung benötigt, desto niedriger fällt der Ausgangspuls des Vorverstärkers und damit die gemessene Energie aus.

Allen ladungssensitiven Vorverstärkern gemein ist mindestens ein Kondensator zur Sammlung der freien Ladungsträger aus dem Detektor, um ein zu dieser Ladung proportionales Spannungssignal zu erzeugen. Die Notwendigkeit der Entleerung dieses Kondensators von Zeit zu Zeit führt im einfachsten Fall zu einem zur Kapazität parallel geschalteten, ohmschen Widerstand, der für eine langsame, kontinuierliche Entladung des Kondensators sorgt (siehe Abb. B.3). Dies führt dazu, dass noch während der Sammlung der freien Ladungsträger aus dem Detektor diese parallel schon wieder über den Wider-

stand abfließen und somit nicht mehr in die Messung eingehen. Mathematisch veranschaulichen lässt sich dieser Effekt anhand eines Rechteckpulses

$$x(t) = \begin{cases} 0 & \text{für } t < 0 \\ \frac{Q}{\Delta t} & \text{für } 0 \leq t \leq \Delta t \\ 0 & \text{für } t > \Delta t \end{cases} \tag{B.10}$$

mit der Gesamtladung Q und der Länge Δt, was zu einer Amplitude (welche dem Strom durch den Detektor entspricht) des Pulses von $\frac{Q}{\Delta t}$ führt.

Wie in Abschnitt B.1 gezeigt, ergibt sich das Ausgangssignal $y(t)$ des Vorverstärkers aus der Faltung des Detektorsignales $x(t)$ mit der Impulsantwort Glg. B.7

$$h_\tau(t) = u(t) \cdot \exp\left(-\frac{t}{\tau}\right) .$$

Der Vorverstärker liefert also

$$\begin{aligned} y(t) &= x(t) * h(t) \\ &= \int_{-\infty}^{+\infty} dt'\, x(t') \cdot h(t - t') \\ &= \int_{0}^{\Delta t} dt'\, \frac{Q}{\Delta t} \cdot u(t - t') \cdot \exp\left(-\frac{t - t'}{\tau}\right) \\ &= \frac{Q}{\Delta t} \cdot \exp\left(-\frac{t}{\tau}\right) \cdot \int_{0}^{\Delta t} dt'\, u(t - t') \cdot \exp\left(\frac{t'}{\tau}\right) . \end{aligned}$$

Damit ergibt sich für das Ausgangssignal während der Ladungssammlung, also für $0 \leq t \leq \Delta t$:

$$\begin{aligned} y(t) &= \frac{Q}{\Delta t} \cdot \exp\left(-\frac{t}{\tau}\right) \cdot \int_{0}^{t} dt'\, \exp\left(\frac{t'}{\tau}\right) \\ &= \frac{Q}{\Delta t} \cdot \exp\left(-\frac{t}{\tau}\right) \cdot \left[\tau \cdot \exp\left(\frac{t'}{\tau}\right)\right]_0^t \\ &= Q \cdot \frac{\tau}{\Delta t} \cdot \exp\left(-\frac{t}{\tau}\right) \cdot \left[\exp\left(\frac{t}{\tau}\right) - 1\right] \\ &= Q \cdot \frac{\tau}{\Delta t} \cdot \left[1 - \exp\left(-\frac{t}{\tau}\right)\right] . \end{aligned}$$

Für $t < 0$ ist $y(t) = 0$. Sind alle freien Ladungsträger vom Detektor abgeflossen, d.h. ist $t > \Delta t$, erhält man

$$y(t) = \frac{Q}{\Delta t} \cdot \exp\left(-\frac{t}{\tau}\right) \cdot \int_{0}^{\Delta t} dt'\, \exp\left(\frac{t'}{\tau}\right)$$

$$= \quad Q \cdot \exp\left(-\frac{t}{\tau}\right) \cdot \underbrace{\frac{\tau}{\Delta t} \cdot \left[\exp\left(\frac{\Delta t}{\tau}\right) - 1\right]}_{konstant},$$

den charakteristischen exponentiellen Abfall, den das Ausgangssignal eines ladungssensitiven Vorverstärkers aufweist. Der gesamte Puls hat also die Form

$$y(t) = \begin{cases} 0 & \text{für } t < 0 \\ Q \cdot \frac{\tau}{\Delta t} \cdot \left[1 - \exp\left(-\frac{t}{\tau}\right)\right] & \text{für } 0 \leq t \leq \Delta t \\ Q \cdot \exp\left(-\frac{t}{\tau}\right) \cdot \frac{\tau}{\Delta t} \cdot \left[\exp\left(\frac{\Delta t}{\tau}\right) - 1\right] & \text{für } t > \Delta t \ . \end{cases} \tag{B.11}$$

Die maximale Amplitude, welche zu der zu messenden Energie proportional sein sollte, wird für einen Rechteckpuls bei $t = \Delta t$ erreicht und beträgt

$$y_{preamp}^{max} = Q \cdot \underbrace{\frac{\tau}{\Delta t} \cdot \left(1 - \exp\left(-\frac{\Delta t}{\tau}\right)\right)}_{\leq 1} \leq Q \ . \tag{B.12}$$

Dabei wurde die Ungleichung

$$\exp(x) \geq 1 + x, \, \forall \, x \in R$$

mit $x = -\frac{\Delta t}{\tau}$, $\frac{\Delta t}{\tau} > 0$ benutzt, woraus folgt

$$\begin{aligned} & \exp\left(-\tfrac{\Delta t}{\tau}\right) && \geq 1 - \frac{\Delta t}{\tau} \\ \Leftrightarrow \quad & \tfrac{\Delta t}{\tau} && \geq 1 - \exp\left(-\frac{\Delta t}{\tau}\right) \\ \Leftrightarrow \quad & 1 && \geq \frac{\tau}{\Delta t} \cdot \left(1 - \exp\left(-\frac{\Delta t}{\tau}\right)\right) \ . \end{aligned}$$

Der relative Fehler, den das Ballistic Deficit erzeugt, ist also

$$\begin{aligned} 1 - \frac{y_{preamp}^{max}}{Q} &= 1 - \frac{\tau}{\Delta t} \cdot \left(1 - \exp\left(-\frac{\Delta t}{\tau}\right)\right) \\ &\approx \frac{1}{2}\frac{\Delta t}{\tau} - \frac{1}{6}\frac{\Delta t^2}{\tau^2} + \ldots \ . \end{aligned}$$

Typischerweise liegt die Zeitkonstante τ des Vorverstärkers im Bereich von 10...100 μs, die Pulslänge Δt zwischen 100...500 ns. Somit ist $\frac{\Delta t}{\tau} < 0,05$ und die maximale Abweichung der Amplitude bewegt sich im unteren einstelligen Prozentbereich. Damit ist auch die Reihenentwicklung der Exponentialfunktion in obiger Umformung gerechtfertigt.

Mindern lässt sich der Effekt durch die Wahl einer möglichst großen Zeitkonstante τ des Vorverstärkers. Dies bringt allerdings ein vermehrtes pile-up von Ereignissen mit sich, d.h. das exponentiell abklingende Signal des vorhergehenden Pulses ist noch nicht vollständig verschwunden und verfälscht somit die Messung des momentanen Pulses (siehe Abb. B.2).

Anhang C

Herleitungen MWD

C.1 Herleitung der Differenzengleichung

Der MWD-Kernalgorithmus wird durch die Differenzengleichung 2.1 repräsentiert:

$$y[n] = x[n] + (1-k) \cdot \sum_{m=1}^{M} x[n-m] - x[n-M]$$

$x[n]$ ist hierbei das digitalisierte Vorverstärkersignal, $y[n]$ das Ausgangssignal des MWD-Algorithmus. Die Form dieser Differenzengleichung kann mittels der digitalen Signaltheorie hergeleitet werden:

Ein idealer Energiealgorithmus sollte jeden Detektorpuls (Stromsignal) einzeln integrieren, um die durch eine Wechselwirkung erzeugte freie Ladung korrekt zu bestimmen. Ein solcher Filter hätte die Form

$$y_{ideal}[n] = \sum_{m=0}^{M-1} v[n-m] \ ,$$

wobei $M \cdot T_{sampling}$ ($T_{sampling} = f_{sampling}^{-1}$, Abtastperiode) hierbei größer, mindestens aber gleich der Länge des Strompulses Δt ist und $v[n]$ das Ausgangssignal des Detektors ohne Vorverstärker darstellt. Diese Gleichung repräsentiert dabei einen digitalen Filter mit der Impulsantwort

$$g[n] = u[n] \cdot u[M-1-n] \qquad \text{mit } u[n] = \begin{cases} 0 & \text{für } n < 0 \\ 1 & \text{für } n \geq 0 \end{cases} \ ,$$

welcher die letzten M Werte aufsummiert. Grafisch veranschaulicht ist dies in Abbildung 2.2.

Digitalisiert wird jedoch das Vorverstärkersignal $x[n]$, welches sich aus $v[n]$ und der Impulsantwort des Vorverstärkers $h[n]$ zusammensetzt, so dass

ein direkter Einsatz obigen Filters nicht möglich ist. Ziel muss es also sein, einen Filter zu konstruieren, der zunächst die unvollständige Integration des Vorverstärkers rückgängig macht, d.h. das Ladungssignal des Vorverstärkers entfaltet und dadurch das Stromsignal des Detektors zurück gewinnt, um anschließend obige Summation über die Länge des Pulses auszuführen. Die Impulsantwort dieses Filters sei $f[n]$ mit der Eigenschaft

$$y[n] = f[n] * x[n] = y_{ideal}[n] \ ,$$

wobei gilt

$$y_{ideal}[n] = g[n] * v[n] \qquad \text{und} \qquad x[n] = h[n] * v[n] \ .$$

Daraus ergibt sich die Gleichung

$$\begin{aligned} f[n] * h[n] * v[n] &= g[n] * v[n] \\ f[n] * h[n] &= g[n] \ . \end{aligned}$$

Das Symbol $*$ repräsentiert die Faltungssumme. Es gilt

$$a[n] * b[n] = b[n] * a[n] = \sum_{m=-\infty}^{\infty} a[m] \cdot b[n-m] \ .$$

z-Transformation
An diesem Punkt wird es wesentlich einfacher, wenn man die Berechnung der gewünschten Impulsantwort $f[n]$ durch eine z-Transformation (siehe Anhang A.3) in die komplexe Ebene verlegt. Hierbei wird aus der Faltung eine Multiplikation und man erhält

$$F(z) \cdot H(z) = G(z)$$

bzw.

$$F(z) = \frac{G(z)}{H(z)} \ . \tag{C.1}$$

Die z-Transformierten der bekannten Impulsantworten $g[n]$ und $h[n]$ sind

$$\begin{aligned} g[n] = u[n] \cdot u[M-1-n] \quad &\Longleftrightarrow \quad G(z) = \sum_{m=0}^{M-1} z^{-m} = \begin{cases} \frac{1-z^{-M}}{1-z^{-1}} & \text{für } z \neq 1 \\ M & \text{für } z = 1 \end{cases} \\ h[n] = k^n \cdot u[n] \quad &\Longleftrightarrow \quad H(z) = \sum_{m=0}^{\infty} \left(\frac{k}{z}\right)^m = \frac{1}{1-\frac{k}{z}} \ \text{ für } |z| > k \end{aligned}$$

und werden Transferfunktionen genannt. Dabei wurde die in Anhang B.1 hergeleitete Impulsantwort eines einfachen ladungssensitiven Vorverstärkers für $h[n]$ benutzt (Glg. B.9) und $\exp(-\alpha n)$ zu $k^n, 0 < k < 1$ zusammengefasst. Der konstante Vorfaktor $\frac{1}{C}$ ist für die folgende Betrachtung nicht relevant.

Wir erhalten also

$$F(z) = \frac{G(z)}{H(z)} = \frac{\sum_{m=0}^{M-1} z^{-m}}{\frac{1}{1-\frac{k}{z}}}$$

für $z \neq 1$

$$\begin{aligned}
&= (1-\frac{k}{z}) \cdot \sum_{m=0}^{M-1} z^{-m} \\
&= (1\underbrace{-k+k}_{=0}-\frac{k}{z}) \cdot \sum_{m=0}^{M-1} z^{-m} \\
&= (1-k) \cdot \sum_{m=0}^{M-1} z^{-m} + k \cdot (1-\frac{1}{z}) \cdot \sum_{m=0}^{M-1} z^{-m} \\
&= (1-k) \cdot \sum_{m=0}^{M-1} z^{-m} + k \cdot (1-z^{-1}) \cdot \frac{1-z^{-M}}{1-z^{-1}} \\
&= (1-k) \cdot \sum_{m=0}^{M-1} z^{-m} + k \cdot (1-z^{-M}) \\
&= (1-k) \cdot \sum_{m=0}^{M-1} z^{-m} + \underbrace{(1-1)}_{=0} + \underbrace{(z^{-M}-z^{-M})}_{=0} + k \cdot (1-z^{-M}) \\
&= (1-k) \cdot \sum_{m=0}^{M-1} z^{-m} - (1-k) + (1-k) \cdot z^{-M} + 1 - z^{-M} \\
&= 1 + (1-k) \cdot \sum_{m=1}^{M} z^{-m} - z^{-M}
\end{aligned}$$

für $z = 1$

$$= M \cdot (1-k) \ .$$

Obige Fallunterscheidung wurde getroffen, um die Umformungen zu erleichtern, das Ergebnis für $z \neq 1$ ist aber offensichtlich auch für $z = 1$ gültig. Somit lässt sich der Filter vollständig beschreiben durch

$$F(z) = 1 + (1-k) \cdot \sum_{m=1}^{M} z^{-m} - z^{-M} \ , \tag{C.2}$$

unter der Voraussetzung $|z| > k$.

Pol- und Nullstellenbetrachtung
Anhand der pole-zero-plots (Abb. C.1(a), C.2(a) und C.3(a)) der Transferfunktionen $G(z)$, $H(z)$ und $F(z)$ und der zugehörigen Frequenzantworten (Abb. C.1(b), C.2(b) und C.3(b)) lässt sich die Funktionsweise des MWD-Filters gut veranschaulichen. Generelle Erläuterungen zum pole-zero-plot und zur Frequenzantwort finden sich in Abschnitt A.3.

Die Polstelle des Vorverstärkers bei $z = k$ wird durch eine Nullstelle des MWD-Filters an derselben Stelle neutralisiert. Die $M - 1$ Nullstellen von $F(z)$ sowie $G(z)$ entstehen durch die Summation über die letzten M Eingangswerte. Signalanteile, deren Frequenz f ein ganzzahliges Vielfaches von $\frac{f_{sampling}}{M}$ ist, löschen sich innerhalb dieser M Werte selbst aus (es gilt $\Omega = \frac{\omega}{f_{sampling}}$ und $\omega = 2\pi f$).

Während der Vorverstärker ein zeitlich konstantes Signal um das $\frac{1}{1-k}$-fache verstärkt (Abb. C.1(b)), ist die Frequenzantwort des MWD-Filters (Abb. C.2(b)) so beschaffen ($F(\omega = 0) = M \cdot (1 - k)$), dass ein solches Signal insgesamt um das M-fache verstärkt wird (Abb. C.3(b)).

Impulsantwort
Eine Rücktransformation dieses Ergebnisses in den diskreten Zeitraum ergibt die gesuchte Impulsantwort

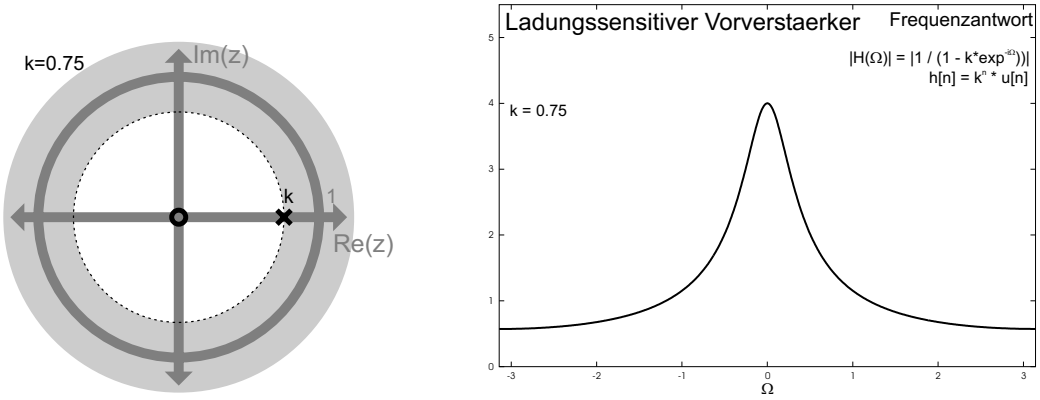
$$\begin{aligned} f[n] &= \delta[n] + (1-k)\sum_{m=1}^{M}\delta[n-m] - \delta[n-M] \\ &= \delta[n] + (1-k)u[n-1]u[M-n] - \delta[n-M] \end{aligned} \tag{C.3}$$

und daraus die Differenzengleichung (Glg. 2.1) des MWD-Filters

$$\begin{aligned} y[n] &= f[n] * x[n] \\ &= x[n] + (1-k)\cdot\sum_{m=1}^{M} x[n-m] - x[n-M] \ . \end{aligned}$$

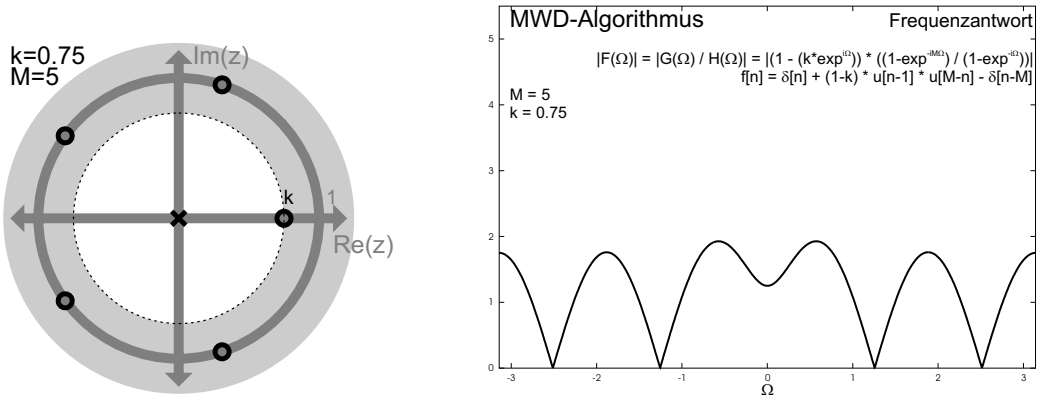
pole-zero-plot

Frequenzantwort von $-\pi$ bis π



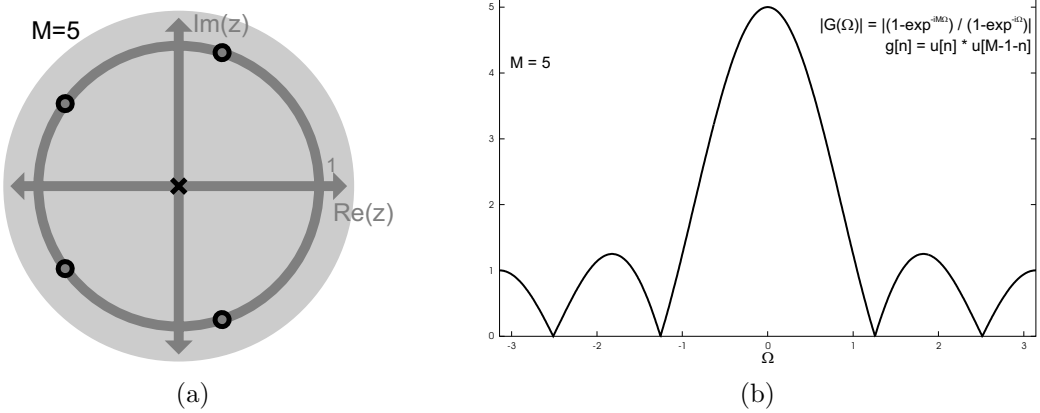
(a) (b)

Abbildung C.1: Ladungssensitiver Vorverstärker



(a) (b)

Abbildung C.2: MWD-Filter



(a) (b)

Abbildung C.3: Summierungs-Filter

C.2 Vertauschbarkeit von M und L

Zeigen lässt sich die Vertauschbarkeit der Parameter M und L am elegantesten unter Betrachtung der Gesamttransferfunktion $\bar{F}(z) = F(z) \cdot W(z)$, wobei $W(z) = \begin{cases} \frac{1}{L} \cdot \frac{1-z^{-L}}{1-z^{-1}} & \text{für } z \neq 1 \\ 1 & \text{für } z = 1 \end{cases}$ die Transferfunktion des arithmetischen Mittelungsfilters ist.

$$\bar{F}(z) = \left(1 - \frac{k}{z}\right) \cdot \begin{cases} \left(\frac{1-z^{-M}}{1-z^{-1}}\right) \cdot \left(\frac{1}{L} \cdot \frac{1-z^{-L}}{1-z^{-1}}\right) & \text{für } z \neq 1 \\ M & \text{für } z = 1 \end{cases}$$

Ein Austauch von M und L verändert $\bar{F}(z)$ also um den Faktor $\frac{L}{M}$.

Anhang D

Begriffserklärung Hardware

D.1 VMEbus

... VERSAModule Eurocard bus Dieser offene Industriestandard beschreibt die physikalischen Maße, die mechanische Befestigung und elektrischen Anschlüsse sowie das Verbindungsprotokoll eines Steckkartensystems. Er wurde 1981 von Motorola, Mostek und Signetics verabschiedet und basiert im elektrischen Teil auf dem älteren VERSAbus-Standard von Motorola sowie dem Eurocard-Standard für den mechanischen Teil. Er ist seither mehrmals überarbeitet worden und mittlerweile von verschiedenen Normierungsorganisationen ratifiziert worden: IEEE[1] 1014-1987, ANSI[2]/VITA[3] 1-1994 (VME64) und IEC[4] 821.

Der VMEbus ist eine asynchrone Master-Slave-Architektur mit einer dynamisch konfigurierbaren Daten- und Adressbreite von 8 bis 64 Bit. Dadurch werden Datentransferraten von bis zu 80MB/s (VME64) möglich. Es können maximal 21 Karten an einem Bus betrieben werden. Für weitere Informationen zum VMEbus siehe [25] oder [26].

D.2 Programmierbare Logik-Bausteine

Programmierbare Logik besteht grundsätzlich aus Flip-Flops und Logikgattern, welche flexibel miteinander verbunden werden können. Informationen über diese Verbindungen werden in EPROM, EEPROM, FLASH oder SRAM-Speicherzellen gespeichert. Dadurch wird es teilweise möglich

[1] Institute of Electrical and Electronics Engineers, http://www.ieee.org
[2] American National Standards Institute, http://www.ansi.org
[3] VMEbus International Trade Association, http://www.vita.com
[4] International Electrotechnical Commission, http://www.iec.ch

(bzw. im SRAM-Fall sogar nötig), die Programmierung der Verbindungen in-circuit, d.h. in eingebautem Zustand, vorzunehmen. Dies wird als ISP, in-system programmable, bezeichnet. Programmiert wird ein solcher Baustein mittels schaltplanbasierter Designtools oder einer Hardware Description Language (HDL) wie Verilog, VHDL oder ABEL. Diese Entwürfe werden dann durch herstellerspezifische Programme in Informationen über die jeweiligen Verbindungen auf dem Baustein umgesetzt. Mit Hilfe von Simulationsprogrammen lassen sich dabei Fehler in der Programmierung oder zu lange Signallaufzeiten auf dem Chip schon frühzeitig ohne Hardwareaufwand erkennen und beseitigen.

D.2.1 CPLD

... Complex Programmable Logic Devices Ein CPLD besteht aus mehreren *Function Blocks*, die untereinander bzw. mit der Außenwelt über eine globale *Switch Matrix* verbunden werden können. Jeder Block besteht aus mehreren *Macrocells*, die jeweils kombinatorische Elemente (UND-, ODER-Gatter) und Flip-Flops zusammenfassen. CPLDs sind auf schnelle Signallaufzeiten innerhalb des Bausteines optimiert und meistens als nichtflüchtige (EPROM, EEPROM oder FLASH) Bausteine ausgelegt.

D.2.2 FPGA

... Field Programmable Gate Array Ein FPGA ist aus vielen, in einer Matrix angeordneten *Logic Cells* aufgebaut, die flexibel untereinander und mit den I/O-Pins des Chips verbunden werden können. Die Dichte von Logikgattern, Flip-Flops und Verbindungsmöglichkeiten ist wesentlich höher als bei CPLDs. FPGAs sind meist in SRAM-Bauweise ausgeführt. Dabei werden neben den Verbindungen auch die Logikelemente als Look-Up-Tables (LUT) im SRAM abgebildet. Moderne FPGAs bieten neben den reinen Logikfunktionen noch eine Fülle weiterer Elemente auf dem Baustein, wie RAM, Clockbuffer oder Multipliziereinheiten. Diese ermöglichen es, auch komplizierte und aufwändige Schaltungen flexibel auf einem FPGA zu realisieren, ohne dabei Rücksicht auf die Umgebung nehmen zu müssen. Da SRAM eine flüchtige Speicherart ist, d.h. der Inhalt ohne externe Energieversorgung verloren geht, muss ein FPGA nach jedem Einschaltvorgang erneut programmiert werden. Dies kann entweder von außen über einen externen Prozessor oder Baustein geschehen oder mittels eines mit dem FPGA verbundenen PROMs, welches dieser selbsttätig beim Start ausliest.

D.3 VHDL

... VHSIC Hardware Description Language Wie der Name schon sagt, handelt es sich bei VHDL um eine Hardware-Beschreibungssprache, wobei VHSIC für "Very High Speed Integrated Circuit" steht. VHDL *beschreibt* die Funktionalität bzw. das Verhalten digitaler elektrischer Schaltungen. Die Sprache dient dabei als Basis sowohl für die Simulation als auch die Synthese von Schaltungen, d.h. die Umsetzung in Hardwareelemente.

VHDL wurde im Auftrag des US Department of Defence von IBM, Texas Instruments und Intermetrics in den 80er Jahren entwickelt. Nach der Erstveröffentlichung des LRM 7.2 (Language Reference Manual) im Jahre 1985 kam es 1987 zur Standardisierung durch das IEEE (IEEE 1076-1987) und 1993 zu einer Überarbeitung (IEEE 1076-1993). Um Inkompatibilitäten zwischen Simulationsprogrammen verschiedener Hersteller und damit einhergehender proprietärer Erweiterungen der Sprache zu vermeiden bzw. zu verringern, folgte der Standard IEEE 1164 (Standard Logic package), der den 9-wertigen Datentyp std_logic beschreibt. Ebenfalls hinzu kam 1995 IEEE 1076.3 (Numeric Standard), der herstellerspezifische Bibliotheken zur Synthese von Schaltungen ersetzen soll. Daneben gibt es mittlerweile auch eine Erweiterung zur Simulation analoger Schaltungen, VHDL-AMS (Analogue and Mixed Signals).

Aufgrund des rein beschreibenden Charakters von VHDL kann die Sprache vielfältig eingesetzt werden - vom Entwurf bzw. Simulation konventioneller Schaltungen bis hin zur Programmierung programmierbarer Logikbausteine. Dabei kann der Code völlig unabhängig vom späteren Einsatzzweck bzw. der konkreten Hardwareimplementierung gehalten werden. VHDL unterstützt dabei eine modulare, hierarchische Programmierung auf unterschiedlichen Abstraktionsebenen - vom einzelnen Logikgatter bis hin zur reinen Verhaltensbeschreibung der Gesamtschaltung - je nach Fähigkeiten bzw. Anforderungen der verwendeten Simulations- und/oder Syntheseprogramme.

D.4 SDRAM

... Synchronous Dynamic Random Access Memory

SDR ... Single Data Rate Ein SDR-SDRAM-Modul arbeitet auf der ansteigenden Flanke des Clock-Signales, im Gegensatz zum DDR-SDRAM (Double Data Rate), welches sowohl ansteigende als auch abfallende Flanke nutzt.

ECC ... Error Correction Code ECC-Module besitzen zu jedem Byte ein weiteres Bit, in dem Informationen zur Überprüfung der Datenkonsistenz gespeichert werden. Dies führt effektiv zu einer Erhöhung des Speichers um ein Achtel.

DIMM ... Dual In-line Memory Module

Registered Alle Eingangssignale werden mit dem Speichertakt in einem Register für einen Takt zwischengespeichert, um gültige Signale bei Taktflanken zu gewährleisten. Zu beachten ist, dass sich dadurch die Reaktion des Speicherbausteines um einen Takt verzögert.

D.5 PROM

... Programmable Read-Only Memory Unter einem ROM versteht man einen nichtflüchtigen Speicher, auf den während des normalen Betriebes nur lesend zugegriffen werden kann. Unter anderem unterscheidet man zwischen EPROM (Eraseable PROM, kann mittels UV-Licht gelöscht werden) und EEPROM oder E^2PROM (Electrically EPROM, kann über elektrische Signale gelöscht bzw. neu beschrieben werden).

Anhang E

C++ Klassenbaum

Die Klassen dieses Projektes machen starken Gebrauch von der Standard C++ library, insbesondere des STL-Teils (Standard Template Library). Die Struktur der Klassen nimmt Anleihen beim ROOT-Framework.

E.1 Basisklassen

Grundlage für alle Klassen sind die Basisklassen `PSAClass` bzw. `PSAObject`:

PSAClass definiert einige grundlegende Funktionen bzw. Schnittstellen, die allen Klassen des Projektes gemein sind. Dies sind

- die Vergleichsoperatoren `==` und `!=` sowie der Zuweisungsoperator `=` ,
- die Funktion `runTest()`, die eine Testroutine für die jeweilige Klasse bereitstellen soll sowie

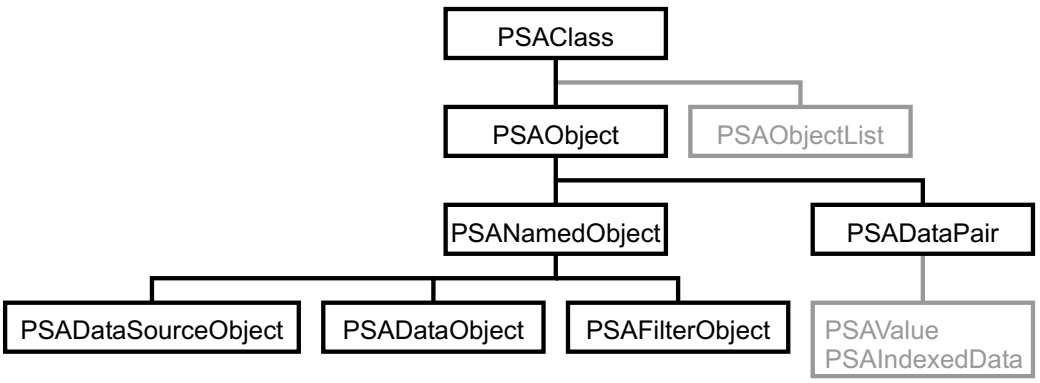


Abbildung E.1: C++ Basisklassen

- die Funktion `ToString()`, die es jeder Klasse ermöglichen soll, Informationen in einem lesbaren Format über sich auszugeben.

Diese Funktionen bzw. Operatoren müssen von jeder abgeleiteten Klasse überschrieben werden. Daneben gibt es die Funktionen

- `GetClassName()` und `GetClassDescription()`, welche zwei über den Konstruktor zu initialisierende Stringvariablen zurückgeben,
- `TestClass()`, welche die zur jeweiligen Klasse gehörende Funktion `runTest()` aufruft, und
- den globalen Operator `<<`, welcher die Ausgabe der Funktion `ToString()` der jeweiligen in ein `ostream`-Objekt umleitet.

PSAObject dient im Zusammenspiel mit der Klasse `PSAObjectList` zur Verwaltung aller vom Programm erzeugten Objekte. Sie speichert eine von einem `PSAObjectList`-Objekt eindeutig vergebene Objekt-ID und stellt diese über die Funktion `GetObjectID()` zur Verfügung.

Zu diesem Zweck muss die Klasse `PSAObjectList` global instanziert werden, damit dieser Mechanismus greift. Eine eindeutige ID für jedes Objekt erleichert den Debugprozess und wird auch von den ROOT-Schnittstellenfunktionen verwendet, um eindeutige ROOT-Objektnamen zu generieren. Desweiteren überprüft `PSAObjectList` bei seiner Beendigung, ob alle während des Programmablaufs erstellten `PSAObject`s auch wieder entfernt wurden. Ist dies nicht der Fall, wird eine Fehlermeldung generiert und das betreffende `PSAObject` automatisch aus dem Speicher entfernt.

Unterhalb dieser beiden Klassen findet eine erste Aufspaltung der Struktur statt:

PSADataPair ist als Template ausgeführt und definiert die Funktionalität, zwei Datenwerte beliebigen unterschiedlichen Formats zu setzten und zu lesen sowie diese in ein stream-Objekt zu schreiben oder daraus auszulesen. Voraussetzung dafür ist, dass die verwendeten Datentypen die Operatoren `<<` und `>>` zur Verfügung stellen. Dies trifft zu für alle Standarddatentypen von C++ sowie für alle `PSADataObject`- und die `PSADataPair`-Klassen selbst.

PSANamedObject stellt eine weitere Möglichkeit zur Namensvergabe sowie Beschreibung zur Verfügung, gedacht für eine Vergabe auf Objektebene. Name und Beschreibung können in einen stream geschrieben und daraus gelesen werden.

Unterhalb von `PSANamedObject` folgt die Gabelung des Klassenbaums in drei Stränge.

PSAFilterObject ist die Basisklasse für alle Filterklassen und stellt allen nachfolgenden Klassen einen Zähler zur Verfügung, der die Anzahl der durchgeführten Schritte mitverfolgt, sowie eine `Reset()`-Funktion.

PSADataObject bildet den Ausgangspunkt für alle datenverwaltenden und -speichernden Klassen. Es werden Funktionen zum Laden von und Schreiben auf der Festplatte bereitgestellt.

PSADataSourceObject ist ein Interface für Zugriffe auf zweidimensional organisierte Daten, z.B. eine Datei mit mehreren Messungen auf mehreren Kanälen. Die abstrakte[1] Klasse ist als Template realisiert, so dass das Format der Daten unabhängig von der Funktionalität ist.

E.2 Filterklassen

Ausgehend von `PSAFilterObject` sind die Filterklassen zunächst nach Anzahl ihrer Eingänge weiter unterteilt:

PSAFilterObject_1t1 bildet die Basis für Filter mit einem Eingang sowie einem Ausgang. Die abstrakte Klasse ist als Template implementiert, trifft also keine Festlegung über die Art des Ein- und des Ausgangs. `PSAFilterObject_1t1` definiert einige Funktionen und Operatoren, die den Umgang mit den Filterklassen stark vereinfachen:

- Der Operator `<<=` ist dreifach überladen. Übergeben werden kann entweder ein Einzelwert, ein `PSADataSample`-Objekt oder ein `PSADataSampleArray`-Objekt. Dieser Operator führt die Funktion `run()` des Filters aus, entweder einmalig für den Einzelwert oder als Schleife für alle Daten im `PSADateSample`-Objekt bzw. als Doppelschleife mit zwischenzeitlichem Reset für ein `PSADataSampleArray`-Objekt. Dieser Operator macht es möglich, Filter auf einfache Art direkt hintereinander zu schalten. Es müssen allerdings Ausgang und Eingang der miteinander verknüpften Filter das gleiche Datenformat aufweisen.

[1] Eine abstrakte Klasse beinhaltet eine oder mehrere rein virtuelle Funktionen. Sie kann nicht direkt instanziert werden. Abgeleitete Klassen müssen diese Funktionen implementieren. Für weitere Informationen über die Konzepte der objektorientierten und generischen Programmierung siehe z.B. [6].

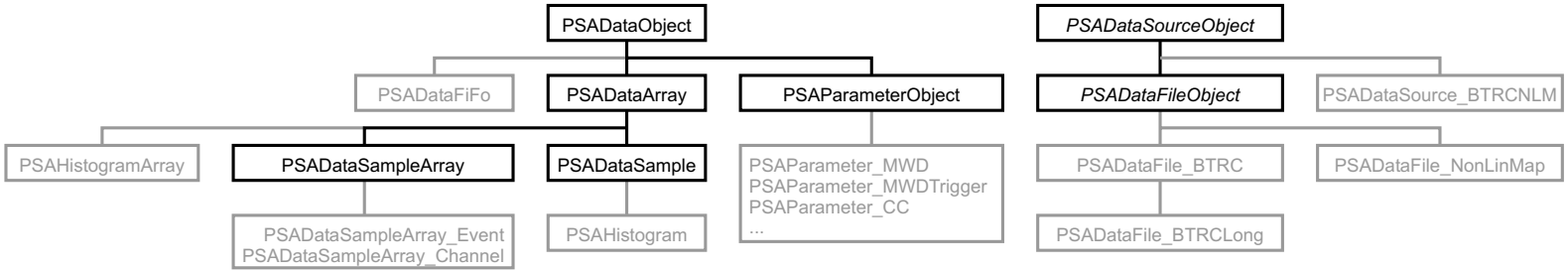


Abbildung E.2: Struktur der von PSADataObject sowie PSADataSourceObject abgeleiteten Klassen. Kursiv gesetzt sind abstrakte Klassen.

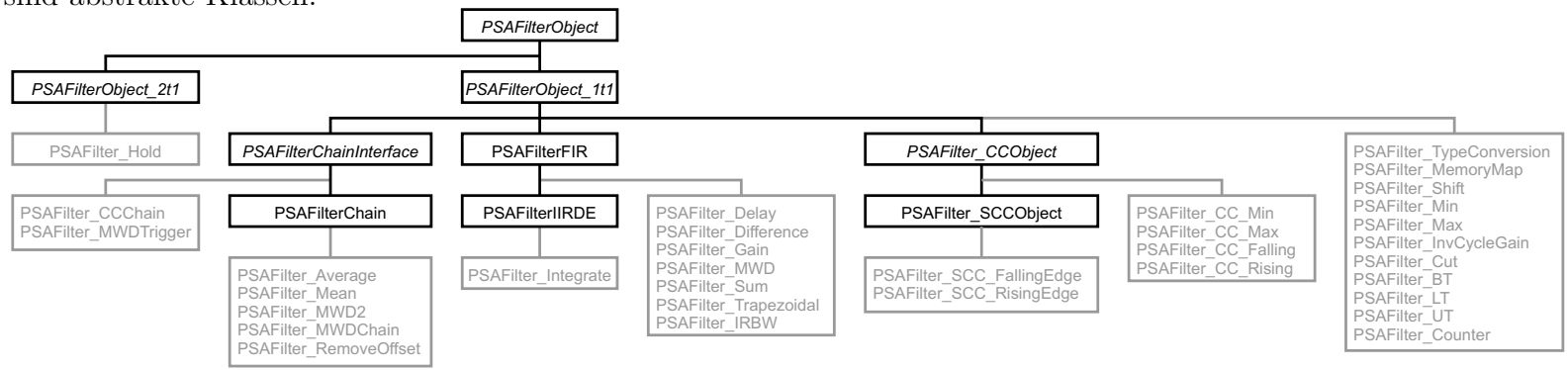


Abbildung E.3: Struktur der von PSAFilterObject abgeleiteten Klassen. Kursiv gesetzt sind abstrakte Klassen.

- Die eben schon erwähnte Funktion `run()` ist das Herzstück der Filterklasse. Diese Funktion muss von jedem Filter überschrieben werden und implementiert die jeweilige Filtereigenschaft.
- `GetCopy()` muss ebenfalls von jeder abgeleiteten Klasse überschrieben werden. Sie erzeugt eine Kopie des Filters und gibt einen Zeiger auf diese zurück.
- `GetImpulsResponse()` berechnet die Impulsantwort des Filters, indem sie einen Einheitspuls durch den Filter laufen lässt. Rückgabewert ist ein `PSADataSample`-Objekt mit der Impulsantwort.

PSAFilterObject_2t1 ist abstrakte Basisklasse für Filter mit zwei Eingängen und einem Ausgang. Auch diese Klasse definiert zu überschreibende Funktionen `run()` und `GetCopy()` und ist ebenfalls datentypunabhängig.

`PSAFilterObject_1t1` differenziert sich weiter in:

PSAFilterChainInterface ist eine abstrakte Klasse, welche Basisfunktionalitäten zur Zusammenfassung mehrerer Filter zu einer Einheit bereitstellt. Im Wesentlichen sind dies eine Liste mit `PSAFilterObject`-Zeigern auf alle innerhalb des Interfaces erzeugten Filter, sowie eine `Reset()`- und `ToString()`-Funktion, die auf alle diese Filter wirken. Der Destruktor der Klasse entfernt automatisch alle Filterobjekte in dieser Liste.

Eine Stufe weiter geht die davon abgeleitete Klasse `PSAFilterChain`. Sie reduziert die Auswahl der in der Kette einsetzbaren Filter auf `PSAFilterObject_1t1`-Klassen mit dem gleichen Aus- und Eingabedatentyp, bietet dafür aber eine automatische Verknüpfung der Filterein- und -ausgänge zu einer Kette. Im Gegensatz zur Verknüpfung verschiedener Filter mit Hilfe des Operator `<<=` berechnet ein `PSAFilterChainObject` in jedem Schritt die gesamte Filterkette für den momentanen Eingangswert, so dass dadurch Feedback-Schleifen zwischen verschiedenen Filtern möglich werden. Der Operator `<<=` in Verbindung mit einem `PSADataSample`- oder `PSADataSampleArray`-Objekt wendet die Filter einzeln hintereinander auf den gesamten Datensatz an.

PSAFilterFIR ist eine Klasse für LTI-Filter ohne Feedback[2]. `PSAFilterIIRDE` erweitert sie um Feedback-Koeffizienten. Ein- und Ausgang haben denselben Datentyp.

[2]siehe Anhang A.

PSAFilter_CCObject (CC..condition counter) ist eine abstrakte Filterklasse, die darauf spezialisiert ist, Daten über einen bestimmten Bereich zu vergleichen (<, >, =, ...) und daraus einen Gewichtungswert zu erstellen. `PSAFilter_SCCObject` integriert diese Gewichtungswerte zusätzlich.

E.3 Datenklassen

Unterhalb von `PSADataObject` verzweigt der Klassenbaum erneut:

PSAParameterObject dient dazu, Parametersätzen für die Algorithmen eine gemeinsame Basis zu geben.

PSADataArray stellt einen eindimensionalen Speicher für beliebige Datenformate zur Verfügung. Funktionen der Klasse erlauben es, Daten zu verschieben, auszuschneiden und aneinander zu hängen sowie Einzelwerte zu verändern. Die Daten können auf der Festplatte gespeichert und auch wieder eingelesen werden. Die Daten können jederzeit zwischen einem `PSADataArray`-Objekt und einem C-Array oder einem STL-`vector` umgewandelt werden.

Die nächste Stufe stellt `PSADataSample` dar. Für die von dieser Klasse unterstützten Datentypen müssen arithmetische Operationen definiert sein. Die Klasse besitzt weitreichende Funktionen zur Manipulation der Daten. Auf zwei `PSADataSample`-Objekte können die Grundrechenarten $+, -, \cdot$ und / angewendet werden, sie können miteinander gefaltet sowie integriert oder differenziert werden. `PSADataSample` ist das Ausgangsdatenformat für die Filterklassen. Unter Verwendung einiger ROOT-Klassen bietet sie die Möglichkeit, die Daten als eps- oder ROOT-Datei zu exportieren oder am Bildschirm anzuzeigen sowie einen Fit an die Daten zu erzeugen.

Daneben existiert die Klasse `PSADataSampleArray`, ein `PSADataArray`-Objekt, welches `PSADataSample`-Objekte beinhaltet. Sie dient dazu, mehrere Ereignisse oder Kanäle innerhalb einer Struktur zu speichern und zu verarbeiten. Auch dieses Format kann direkt mit den Filtern gekoppelt werden.

E.4 Datenquellen

`PSADataSourceObject` beinhaltet die Schnittstellendefinition zu externen Datenquellen. Diese können zweidimensional strukturiert sein, die beiden Di-

mensionen sind als Event und Channel deklariert. Es werden Funktionen zum Öffnen und Schließen der Quelle sowie zum Abrufen der Daten definiert. Die abgeleitete Klasse `PSADataFileObject` ist ebenfalls abstrakt und implementiert Funktionen für Datenquellen im Dateiformat.

E.5 Headerdateien

Neben den Klassen-Headerdateien umfasst das Projekt noch weitere globale Headerdateien.

debug.h enthält die Präprozessoranweisungen, um den für die jeweilige Klasse ausgewählten Debugmodus umzusetzen. Jede Klasse muss diese Headerdatei einbinden.

global.h muss ebenso wie debug.h in jede Klasse eingebunden werden. Hier werden die globalen Debugeinstellungen getroffen sowie der gloable Dumpstream für nicht benötigte Debuginformationen deklariert. Einige für das Projekt wichtige Headerdateien werden ebenfalls eingebunden, unter anderem globalfunctions.h und os.h .

globalfunctions.h enthält die Definition und den Code des Funktionentemplates `ToString(const gtype\& data)` . Diese Funktion dient dazu, Datenformate, die keine explizite Umwandlung in ein `string`-Objekt besitzen, über den Umweg des `stream`-Operators `<<` in einen `string` umzuwandeln.

PSA.h erzeugt das `PSAObjectList`-Objekt für das Projekt und initialisiert den Debug-Dumpstream.

os/os.h definiert betriebssystemabhängige Konstanten.

main_template.h implementiert eine Standard main-Funktion. Sie ruft eine vom Benutzer zu erstellende Funktion int run(int argc, char *argv[]) auf. Diese main-Funktion erstellt eine Laufzeitmessung und eine Statistik zu den erstellten `PSAObject`-Objekten.

Anhang F

VHDL-Quelltexte

Quelltext einiger ausgewählter VHDL-Module.

psa_filter_mwd_algorithm_direct

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

use work.testfunctions.all;

-- y[n] = x[n-3] - x[n-M-3] + (1-k)*(x[n-4] + ... + x[n-M-3])
-- k = exp(a), a ~ -1E-4   => (1-k) ~ 1E-4
-- EXP_DECAY = (1-k) * (2**NOBUB) ==>    0 < (1-k) < 2**(EDA-NOBUB)

entity psa_filter_mwd_algorithm_direct is
  generic (
    MAX_WS : positive range 2 to positive'high := 1023; -- maximum window size, number
        of samples (=M), < 1024
    EDA : positive range 1 to 17 := 8; -- accuracy of exp_decay constant (bits)
    NOBUB : natural := 20; --number of blow-up bits
      -- [(2**-(NOBUB-EDA))] gives maximum value of (1-k)
    DW : positive range 1 to 17 := 14 -- data size (bits)
  );
    port (
    CLOCK : in std_logic;
    RESET : in std_logic;
    RESTART : in std_logic;
    SET_ED : in std_logic;
    EXP_DECAY : in std_logic_vector(EDA-1 downto 0); -- value of (1-k)*(2**NOBUB)
    SET_WS : in std_logic;
    DATA_N0_IN : in std_logic_vector(DW-1 downto 0);
    DATA_NM_IN : in std_logic_vector(DW-1 downto 0);
    DATA_OUT : out std_logic_vector(DW+2+max(0, get_bit_length(MAX_WS)-1-NOBUB+EDA)+
        NOBUB-1 downto 0)
  );
end entity psa_filter_mwd_algorithm_direct;

architecture Behavioral of psa_filter_mwd_algorithm_direct is

  signal MEMORY_EXP_DECAY : SIGNED(EDA downto 0);

  signal BUFFER_DATA_N0_IN : SIGNED(DW downto 0);
  signal BUFFER_DATA_NM_IN : SIGNED(DW downto 0);

  signal MEMORY_DELTA_DATA_IN : SIGNED(DW downto 0);
  signal BUFFER_MEMORY_DELTA_DATA_IN : SIGNED(DW downto 0);
  signal MEMORY_DELTA_DATA_IN_MULT : SIGNED(DW+(EDA+1) downto 0);
  signal MEMORY_SUM : SIGNED(DW+(EDA+1)+(get_bit_length(MAX_WS)-1)-1 downto 0);

begin

  algorithm : process(RESET, CLOCK)
  begin
```

```
    if RESET = '1' then
      BUFFER_DATA_N0_IN <= (others => '0');
      BUFFER_DATA_NM_IN <= (others => '0');
      MEMORY_DELTA_DATA_IN <= (others => '0');
      MEMORY_DELTA_DATA_IN_MULT <= (others => '0');
      BUFFER_MEMORY_DELTA_DATA_IN <= (others => '0');
      MEMORY_SUM <= (others => '0');
      DATA_OUT <= (others => '0');
    elsif (CLOCK'event and CLOCK = '1') then
      if (RESTART = '1' or SET_WS = '1') then
        BUFFER_DATA_N0_IN <= (others => '0');
        BUFFER_DATA_NM_IN <= (others => '0');
        MEMORY_DELTA_DATA_IN <= (others => '0');
        MEMORY_DELTA_DATA_IN_MULT <= (others => '0');
        BUFFER_MEMORY_DELTA_DATA_IN <= (others => '0');
        MEMORY_SUM <= (others => '0');
        DATA_OUT <= (others => '0');
      else
    --cycle 1
        BUFFER_DATA_N0_IN <= RESIZE(SIGNED(DATA_N0_IN), DW+1);
        BUFFER_DATA_NM_IN <= RESIZE(SIGNED(DATA_NM_IN), DW+1);
    --cycle 2
        MEMORY_DELTA_DATA_IN <= BUFFER_DATA_N0_IN - BUFFER_DATA_NM_IN; -- = x[n] - x[n-M
            ]
    --cycle 3
        MEMORY_DELTA_DATA_IN_MULT <= MEMORY_DELTA_DATA_IN * MEMORY_EXP_DECAY; -- = (1-k)
            * (x[n] - x[n-M])
        BUFFER_MEMORY_DELTA_DATA_IN <= MEMORY_DELTA_DATA_IN;
    --cycle 4
        MEMORY_SUM <= MEMORY_SUM + MEMORY_DELTA_DATA_IN_MULT; -- = summe(x[n] bis x[n-M
            +1])
        DATA_OUT(DATA_OUT'length-1 downto NOBUB) <= STD_LOGIC_VECTOR(RESIZE(MEMORY_SUM(
            MEMORY_SUM'length-1 downto NOBUB), DATA_OUT'length-NOBUB) + RESIZE(
            BUFFER_MEMORY_DELTA_DATA_IN, DATA_OUT'length-NOBUB));
        DATA_OUT(NOBUB-1 downto 0) <= STD_LOGIC_VECTOR(MEMORY_SUM(NOBUB-1 downto 0));
      end if;
    end if;
  end process algorithm;

  config : process(RESET, CLOCK)
  begin
    if RESET = '1' then
      MEMORY_EXP_DECAY <= SIGNED('0' & EXP_DECAY);
    elsif (CLOCK'event and CLOCK = '1' and SET_ED = '1') then
      MEMORY_EXP_DECAY <= SIGNED('0' & EXP_DECAY);
    end if;
  end process config;

end architecture Behavioral;
```

Listing F.1: psa_filter_mwd_algorithm_direct

psa_bitcounter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

use work.testfunctions.all;

entity psa_bitcounter is
  generic (
    bc_type : string := "old";
    DW : positive := 14 -- data size (bits)
  );
    port (
    CLOCK : in std_logic;
    RESET : in std_logic;
    DATA_IN : in std_logic_vector(DW-1 downto 0);
    DATA_OUT : out std_logic_vector(get_bit_length(DW)-1 downto 0)
  );
end entity psa_bitcounter;

architecture Behavioral of psa_bitcounter is


begin
```

```
  addertype : if (bc_type = "adder") generate
    constant nobpc : positive := 4;

    constant nocounter : positive := (DW+nobpc-1) / nobpc;
    constant noadderlevel : positive := get_bit_length(nocounter-1);
    constant novirtcounter : positive := 2**noadderlevel;

    signal DATA_IN_BUFFER : std_logic_vector((novirtcounter*nobpc)-1 downto 0) := (
        others => '0');
    type resarray is array (noadderlevel downto 0, novirtcounter-1 downto 0) of UNSIGNED
        (get_bit_length(DW)-1 downto 0);
    signal ADDER_RESULTS : resarray;
  begin
    DATA_OUT <= std_logic_vector(ADDER_RESULTS(0,0));
    adder : process(RESET, CLOCK)
      variable lsbit : natural := 0;
      variable buf : std_logic_vector(nobpc-1 downto 0);
    begin
      if (RESET = '1') then
        for i in 0 to noadderlevel loop
          for j in 0 to (2**i)-1 loop
            ADDER_RESULTS(i, j) <= (others => '0');
          end loop;
        end loop;
        lsbit := 0;
        buf := (others => '0');
        DATA_IN_BUFFER <= (others => '0');
      elsif (CLOCK'event and CLOCK = '1') then
        for i in 0 to noadderlevel-1 loop
          for j in 0 to (2**i)-1 loop
            ADDER_RESULTS(i, j) <= (ADDER_RESULTS(i+1, 2*j) + ADDER_RESULTS(i+1, (2*j)
                +1));
          end loop;
        end loop;
        DATA_IN_BUFFER(DW-1 downto 0) <= DATA_IN;
        for k in 0 to novirtcounter-1 loop
          lsbit := nobpc * k;
          buf := DATA_IN_BUFFER(lsbit+nobpc-1 downto lsbit);
          ADDER_RESULTS(noadderlevel, k) <= RESIZE(count_bits(buf), get_bit_length(DW));
        end loop;
      end if;
    end process adder;

  end generate addertype;

  oldtype : if (bc_type = "old") generate
  begin
    bit_counters : process(RESET, CLOCK)
    begin
      if (RESET = '1') then
        DATA_OUT <= (others => '0');
      elsif (CLOCK'event and CLOCK = '1') then
        DATA_OUT <= std_logic_vector(count_bits(DATA_IN));
      end if;
    end process bit_counters;
  end generate oldtype;


end architecture Behavioral;
```

Listing F.2: psa_bitcounter

Literaturverzeichnis

[1] *GRETA*. http://greta.lbl.gov/

[2] *MARS - Mini Array di Rivelatori Segmentati*. http://axpd30.pd.infn.it/MARS/

[3] *Virtex-II Platform FPGA User Guide*. http://direct.xilinx.com/bvdocs/userguides/ug002.pdf

[4] *Virtex-II Platform FPGAs Advance Product Specification*. http://direct.xilinx.com/bvdocs/publications/ds031.pdf

[5] Bronstein ; Semendjajew ; Musiol ; Muehlig: *Taschenbuch der Mathematik*. 2. Verlag Harri Deutsch, 1994

[6] Cline, M. *C++ FAQ LITE*. http://www.parashift.com/c++-faq-lite/index.html

[7] Duchene, G. ; Medina, P. *Slide Report - TMR 2001 User Meeting on Gamma-Ray Tracking Detectors, University of Liverpool, UK*

[8] Eberth, J. et a.: Miniball - A Ge Detector Array for Radioactive Ion Beam Facilities. In: *Progress in Particle and Nuclear Physics* 46 (2001), S. 389–398

[9] Feiten, B. ; Röbel, A. *Digitale Signalverarbeitung, Skript zur Vorlesung WS 96/97 – SS 97*. http://gigant.kgw.tu-berlin.de/KW/lehre/skript/ds/dssk.html. 2001

[10] Gast, W. *Informationen zum SCC-Triggeralgorithmus*. private Korrespondenz

[11] Gast, W. *Slide Report - TMR 2001 User Meeting on Gamma-Ray Tracking Detectors, University of Liverpool, UK*

[12] Gast, W. ; Lieder, R.M. ; Mihailescu, L. ; Rossewij, M.J. ; Brands, H. ; Georgiev, A. ; Stein, J. ; Kröll, Th.: Digital Signal Processing and Algorithms for Gamma-Ray Tracking. In: *IEEE Transactions On Nuclear Science* 48 (2001), 12, Nr. 6, S. 2380–2384

[13] Georgiev, A. ; Gast, W.: Digital Pulse Processing in High Resolution High Throughput Gamma-Ray Spectroscopy. In: *IEEE Transactions On Nuclear Science* 40 (1993), Nr. 4, S. 770–779

[14] Gerl, J. (Hrsg.) ; Korten, W. (Hrsg.): *AGATA - Technical Proposal for an Advanced Gamma Tracking Array for the European Gamma Spectroscopy Communitiy*. ftp://ftp.gsi.de/pub/agata, 9 2001

[15] Goulding, F.S. ; Landis, D.A.: Signal Processing for Semiconductor Detectors. In: *IEEE Transactions On Nuclear Science* 29 (1982), 6, Nr. 3, S. 1125–1141

[16] Helmer, R.G. ; Lee, M.A.: Analytical Functions for fitting peaks from Ge Semiconductor Detectors. In: *Nuclear Instruments and Methods* 178 (1980), S. 499–512

[17] Hinshaw, S.M. ; Landis, D.A.: A Practical Approach to Ballistic Deficit Correction. In: *IEEE Transactions On Nuclear Science* 37 (1990), Nr. 2, S. 374–377

[18] Horowitz, P. ; Hill, W.: *The art of electronics*. Cambridge Univ. Press, 1999

[19] Jordanov, V.T. ; Knoll, G.F.: Digital synthesis of pulse shapes in real time for high resolution radiation spectroscopy. In: *Nuclear Instruments and Methods in Physics Research A* 345 (1994), S. 337–345

[20] Koronov, I. *private Mitteilungen*. 2003

[21] Kröll, Th. ; Bazzacco, D.: Simulation and analysis of pulse shapes from highly segmented HPGe detectors for the gamma-ray tracking array MARS. In: *Nuclear Instruments and Methods in Physics Research A* 463 (2001), S. 227–249

[22] Lauer, M.: *Implementierung von Algorithmen zur Echtzeitpulsformanalyse von HPGe Detektorsignalen*, Ruprecht-Karls-Universität Heidelberg, Diplomarbeit Physik, 2001

[23] Leo, W.R.: *Techniques for Nuclear and Particle Physics Experiments*. 2nd. Springer-Verlag, 1994

[24] Lister, K. *Slide Report - TMR 2000 User Meeting on Gamma-Ray Tracking Detectors, Universität Köln*

[25] Peterson, W.D.: *The VMEbus Handbook*. 3rd. VFEA International Trade Association, 1993

[26] Peterson, W.D. *VMEbus FAQ*. http://www.vita.com/vmefaq/index.html. 2000

[27] Phillips, G.W. ; Marlow, K.W.: In: *Nuclear Instruments and Methods* 137 (1976), S. 525

[28] Prussin, S.G.: Prospects for near state-of-the-art analysis of complex semiconductor spectra in the small laboratory. In: *Nuclear Instruments and Methods* 193 (1982), S. 121–128

[29] Smith, J.O. *Introduction to Digital Filters*. http://www-ccrma.stanford.edu/ jos/filters/

[30] University, Rice. *Signals and Systems*. http://cnx.rice.edu/content/col10064/latest/contents/

[31] Venturelli, R. ; Bazzacco, D. ; Isocrate, R. ; Bellato, M. ; Kröll, Th.: On the linearity of the sampling electronics used by MARS / INFN/LNL. 2002 (182). – Annual Report 2001

[32] Venturelli, R. ; Bazzacco, D. ; Isocrate, R. ; Bellato, M. ; Manea, Ch. ; Kröll, Th.: Data acquisition system for the highly segmented detector MARS / INFN/LNL. 2003 (198). – Annual Report 2002

Danksagung

Mein besonderer Dank gilt Herrn Prof. Dr. Reiner Krücken für sein förderndes Interesse und die fantastischen Möglichkeiten, die er mir während des vergangenen Jahres gegeben hat, insbesondere die Teilnahme an der RIA Summer School in den USA.

Herrn Dr. Thorsten Kröll möchte ich herzlich danken für die Betreuung dieser Arbeit, die Beantwortung aller meiner Fragen und die zahlreichen klärenden Gespräche.

Bedanken möchte ich mich beim Lehrstuhl E12 für die nette Aufnahme in ihrer Mitte und die fruchtbaren Diskussionen. Herrn Michael Böhmer danke ich für seine Hilfe bei der Inbetriebnahme der VMEbus-FPGA-Karte und die unermüdliche Beantwortung meiner Elektronikfragen. Herrn Dr. Mathias Münch sei gedankt für den VMEbus Xilinx-Loader, Herrn Benjamin Sailer für seine Hilfe bei der Formatierung dieses Dokuments.

Dem Lehrstuhl E18 bin ich zu Dank verpflichtet für die Überlassung der VMEbus-FPGA-Karte. Insbesondere gilt mein Dank Herrn Dipl.-Ing. Igor Koronov für die Bereitstellung der VHDL-Sourcen der Karte und seine Hilfe bei VHDL-Fragen aller Art.

Ich danke Herrn Dr. Werner Gast für die wertvollen Informationen zum SCC-Triggeralgorithmus.