Fakultät für Physik und Astronomie
Ruprecht-Karls-Universität Heidelberg

Diplomarbeit
im Studiengang Physik

vorgelegt von
Martin Lauer
aus St.Ingbert

2001

Implementierung von Algorithmen zur Echtzeitpulsformanalyse von HPGe Detektorsignalen

Die Diplomarbeit wurde von Martin Lauer ausgeführt am Max-Planck-Institut für Kernphysik unter der Betreuung von

Herrn Prof. Dr. Dirk Schwalm

Implementierung von Algorithmen zur Echtzeitpulsformanalyse von HPGe Detektorsignalen

Diese Diplomarbeit berichtet über die Implementierung von Algorithmen zur Echtzeitpulsformanalyse. Diese Algorithmen ermöglichen eine Bestimmung des Orts der Hauptwechselwirkung innerhalb eines einzelnen MINIBALL Germaniumdetektormoduls. Ermöglicht wurde dies durch die Verfügbarkeit einer neuartigen Elektronik, dem CAMAC Modul XIA DGF-4C, welche das Vorverstärkersignal digitalisiert, und dieses anschliessend durch digitale Bausteine (FPGA und DSP) weiterverarbeitet.
Die eigenen Algorithmen wurden in Assembler programmiert und in das Hauptprogramm des DSPs eingebunden. Der zur Bestimmung des Radius benutzte Steepest Slope Algorithmus, scheint bei kleinen Radien geringfügig schlechtere Resultate als erwartet zu produzieren. Die zur Extraktion einer Winkelinformation eingesetzten Algorithmen liefern Resultate, die denen der zuvor benutzten offline Analysen entsprechen. Der gesamte Verarbeitungszeitraum des DSPs wird um $\sim$ 20 μs erhöht, gleichzeitig kann aber auf die Auslese der digitalisierten Pulsformen verzichtet werden. Es wurde versucht den Aufbau und die Funktionsweise des Moduls XIA DGF-4C durch reverse engineering zu verstehen, sowie dessen Totzeit aus den, von der Elektronik gelieferten, Parametern zu bestimmen.

Implementation of algorithms for realtime pulse shape analysis of HPGe detector signals

This diploma thesis reports on the implementation of algorithms for realtime pulse shape analysis. These algorithms allow a determination of the position of the main interaction inside a single MINIBALL germanium detector module. This was made possible by the availability of an innovative electronics, the CAMAC module XIA DGF-4C, which digitizes the preamplifier signal and subsequently processes the signal with digital chips (FPGA and DSP).
The algorithms developed were programmed in assembler and incorporated in the main code of the DSP. The steepest slope algorithm used to determine the radial position, seems to deliver slightly worser results at small radii than expected. The algorithms used to extract the angular information provide results equivalent to those received previously by an offline analysis. The total processing time of the DSP increases by $\sim$ 20 μs, but the readout of the digitized waveforms can be omitted. It was attempted to understand the structur and the function of the module XIA DGF-4C by reverse engineering and to determine the deadtime from the parameters delivered by the module.

Danksagung

'Das Problem vieler Beziehungen ist die Kom-mu-ni-ka-tion. Zuviel Kom-mu-ni-ka-tion.'
Homer J. Simpson

Zuerst möchte ich mich bei Prof. Dr. Dirk Schwalm dafür bedanken, dass er mich in seine Arbeitsgruppe Gammaspektroskopie aufgenommen hat und mir die Möglichkeit gab, über dieses Thema zu schreiben.
Danken möchte ich vor allem Herrn Dr. Heiko Scheit und Herrn Dr. Christoph Gund für ihre Geduld, die angenehme Arbeitsatmosphäre, die Mittagspausen beim "Pfälzer", die Unterstützung bei der Inbetriebnahme des Moduls DGF-4C und die ausgezeichnete Betreuung.
Ich danke Herrn Dr. Frank Köck, der mir immer bereitwillig Auskunft gab in Fragen bezüglich der Datenaufnahme und auch sonst Wissenlücken in Rechnerfragen zu füllen wusste.
Mein Dank geht an Herrn Oliver Koschorrek, der sich sorgfältig um unsere Elektronik gekümmert hat.
Ich danke Herrn Albert Rausch vom Physikalischen Institut der Universität Heidelberg für die Unterstützung bei der Inbetriebnahme des fast CAMAC-Controllers und die ständigen Bemühungen diesen zu verbessern.
Ich möchte mich ausserdem bei Herrn Dirk Weisshaar und Herrn Dr. Georg Thomas vom IKP in Köln bedanken, die in Fragen bezüglich Elektronik, Algorithmen und Detektoren immer zur Verfügung standen und uns außerdem großzügig mit Elektronikbauteilen versorgten, wenn gerade mal Not am Mann war.
Desweiteren möchte ich mich bei Herrn Falk Lesser vom Institut für Hardware Informatik der Universität Heidelberg bedanken, da er keine Mühen gescheut hat mir beim Test des FPGA Designs zu helfen. Ausserdem konnte ich von ihm in kurzer Zeit viel über VHDL lernen.
Danken möchte ich auch der Firma XIA, da sie das Interface für den User DSP Code zur Verfügung gestellt hat. Außerdem hat sie es innerhalb eines Jahres nicht geschafft, das Ende eines Runs anzuzeigen. Danke !
Mein innigster Dank gilt meiner Freundin Anke, meiner Familie, meinen Freunden und den Bandkollegen. Without music, life is a journey through a desert.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung und Motivation

> 'I know I'm not usually a praying man, but if you're up there, Please Superman help me!'
> Homer J. Simpson

Das öffentliche Interesse an der Kernphysik scheint auf Restlaufzeiten für Kernkraftwerke, Endlagerung von Brennstäben und Castor Transporte beschränkt zu sein. Aus diesem Grund ist es oft schwierig, einen Nicht-Physiker von der Wichtigkeit der Untersuchung sogenannter exotischer Kerne zu überzeugen. Unter exotischen Kernen versteht man Kerne, die aus viel mehr Protonen als Neutronen bestehen bzw. den entgegengesetzten Fall. Warum interessiert man sich für Kerne die in der Natur nicht vorkommen, die also künstlich erzeugen müssen ?

Dass ein großes akademisches Interesse an der Untersuchung dieser exotischen Kerne besteht, liegt an der Tatsache, dass ein Studium der Struktur exotischer Kerne für ein Verständnis der Entstehung der schweren Elemente und astrophysikalischer Prozesse äußerst wichtig ist. Die Ursache dafür, dass diese Kerne nicht in der Natur vorhanden sind, liegt in der kurzen Lebensdauer der exotischen Kerne. Deshalb sind die exotischen Kerne, die z.B. als Zwischenprodukte bei der Synthese schwerer Elemente vorkommen, in der natürlichen Umgebung nicht – oder besser nicht mehr – vorhanden. Wenn man diese Kerne untersuchen will, muss man sie künstlich erzeugen, d.h. man muss für die passenden äußeren Bedingungen sorgen, dann entstehen exotische Kerne relativ häufig (Man vergleiche nur den Aufwand der betrieben werden muss, um im Labor Bedingungen zu erschaffen, die denen zur Zeit des Urknalls entsprechen sollen. Man erhofft sich davon die Entdeckung eines besonderen Zustands der Materie, dem sog. Quark-Gluon-Plasma.). Um diese Bedingungen zur erzeugen gibt es derzeit zwei Möglichkeiten:

- Fragment Separation (z.B. durch die FRS-Anlage[1] der Gesellschaft für Schwerionenforschung (GSI) in Darmstadt)

- Isotopen Separation (z.B. durch die ISOLDE-Anlage[2] am CERN[3])

[1] Fragment Separator
[2] Isotope Separation On Line
[3] Conseil Européen pour la Recherche Nucléaire, Genf

Da das Max-Planck-Insitut für Kernphysik in Heidelberg (MPI-K) an der Entwicklung des REX-ISOLDE[4] Experiments [1] beiteiligt ist, werde ich auf diese Methode kurz eingehen: Das REX-ISOLDE Experiment nutzt den radioaktiven Strahl, den der 1 bzw. 1,4 GeV Protonen Strahl des PS-Boosters[5] beim Auftreffen auf schwere Target-Materialien wie z.B. Uran durch Fragmentation erzeugt. Die entstehenden Exoten diffundieren aus dem Targetmaterial, werden anschließend durch verschiedenene Methoden ionisiert und auf 60 keV vorbeschleunigt. Dieser Strahl wird durch einen Massenseparator geschickt und dann - als nahezu isotopenreiner Strahl - an ein Experiment weitergeleitet. Das REX-ISOLDE Experiment sammelt und kühlt den eingehenden Strahl in einer Penning-Falle, erhöht den Ladungszustand in einer EBIS[6] und beschleunigt die Isotope anschließend auf eine Maximalenergie von 2.2 MeV·A ($\beta = \frac{v}{c} = 6.8\%$). Der Vorteil dieser Methode liegt in der guten Strahlqualität, die allerdings damit erkauft werden muss, dass man durch dieses Beschleunigungsschema auf exotische Kerne mit Zerfallzeiten $\geq$ 10 ms beschränkt ist.

Ein großes Problem bei Experimenten mit exotischen Strahlen ist die geringe Strahlintensität. Allgemein kann man sagen, dass je exotischer der Strahl ist (d.h. je größer der Überschuss an Protonen bzw. Neutronen), desto geringer ist die Intensität, da es zunehmend unwarscheinlicher wird, dass diese Kerne erzeugt werden. Deshalb musste ein hocheffizienter γ-Detektor entwickelt werden, damit man die radioaktiven Kerne γ-spektroskopisch untersuchen kann. Aus diesem Grund wurde das MINIBALL-Projekt [2] in Angriff genommen, das die Entwicklung und Realisierung eines hocheffizienten γ-Detektors bestehend aus Germaniumzählern zum Ziel hat. Eine Anforderung an den MINIBALL-Detektor ist, dass eine Korrektur der Dopplerverbreiterung der Spektrallinien trotz großer Nähe zum Target möglich sein soll. Dazu ist es nötig, die Raumwinkelauflösung bei möglichst unveränderter Geometrie zu erhöhen. Dies geschieht erstens durch sechsfache Segmentation der äußeren Kontakte der einzelnen Module und zweitens durch Analyse der Form des ansteigenden Teils der – durch ein γ-Quant ausgelösten – Ladungspulse. Dadurch wird eine ca. 60 fache Segmentation eines einzelnen Detektor-Moduls erreicht. Die Pulsform kann in einer neuartigen Elektronik (XIA DGF-4C) direkt – on board – in Realtime analysiert werden, was Gegenstand dieser Arbeit ist. Dadurch ist es nicht mehr nötig, die digitalisierten Pulse auszulesen, was aufgrund der Menge an Daten die Datenaufnahme extrem verlangsamen würde. Stattdessen kann die Auslese auf einige wenige aus der Pulsform extrahierten Parameterwerte beschränkt werden.

[4] **R**adioactive Beam **Ex**periment at ISOLDE
[5] Proton Synchrotron
[6] Electron Beam Ion Source

Kapitel 2

Germaniumdetektoren

'Okay, brain. You don't like me, and I don't like you, but let's get through this thing and then I can continue killing you with beer.'
Homer J. Simpson

2.1 Der MINIBALL-Detekor

Die MINIBALL Detektoren haben aus Kostengründen die gleiche Form wie die EUROBALL Detektoren, welche für eine Entfernung von 42 cm zum Target optimiert wurden. Da man Zerfälle mit geringer Multiplizität untersucht ($M_\gamma < 15$), sollen die MINIBALL Detektoren für hocheffiziente Aufbauten bei Abständen bis hinunter zu 10,6 cm eingesetzt werden, wodurch sich der von einem Modul abgedeckte Raumwinkel vergrößert, was die notwendige Korrektur der Dopplerverbreiterung in dem geplanten Maße nicht mehr erlaubt hätte. Deshalb wurde bei den MINIBALL Detektoren als Neuerung eine sechsfache Segmentierung des äußeren Kontakts eingeführt, die es – zusammen mit der vervielfachten Vorverstärkerelektronik – erlaubt die Signale der einzelne Segmente auszulesen. Durch digitale Pulsformanalyse kann die Granularität eines einzelnen Detektormoduls soweit erhöhen werden, dass eine ausreichende Korrektur der Dopplerverbreiterung (Beitrag durch Dopplerverbreiterung ≈ 1 % E_γ) erreicht werden kann. Ein weiteres Designziel war die Fähigkeit Zählraten $\leq 10^4$ Ereignissen pro Sekunde zu verarbeiten, weshalb eine neuartige kommerzielle Elektronik benutzt wird, die dies durch trapezförmige Filter definierter Länge und einer Pipeline Architektur erreicht (Unter Pipelining versteht man die Unterteilung einer Aufgabe in mehrere zeitlich aufeinanderfolgende und voneinander unabhängige Teilaufgaben. Eine Aufgabe nimmt dann pro Bearbeitungsschritt/Taktzyklus nur noch einen Teil des Designs in Anspruch, sodass gleichzeitig mehrere Aufgaben bearbeitet werden können). Für die absolute full-energy peak (FEP) Effizienz[1] ϵ_{ph} der MINIBALL Detektoren sind folgende Ziele anvisiert [2]: $\epsilon_{ph} \geq 20\%$ bei $E_\gamma = 1.3$ MeV und $\epsilon_{ph} \geq 5\%$ bei $E_\gamma = 11.7$ MeV.

[1] Die FEP-Effizienz ist die Warscheinlichkeit dafür, dass die Energie eines γ-Quants vollständig in einem Detektor nachgewiesen wird. Bezieht man sich dabei auf alle von einer Quelle ausgesandten γ-Strahlen, so nennt man sie die *absolute* FEP-Effizienz. Nimmt man stattdessen bezug auf die mit dem Detektor wechselwirkende Strahlung, so nennt man sie die *intrinsische* FEP-Effizienz.

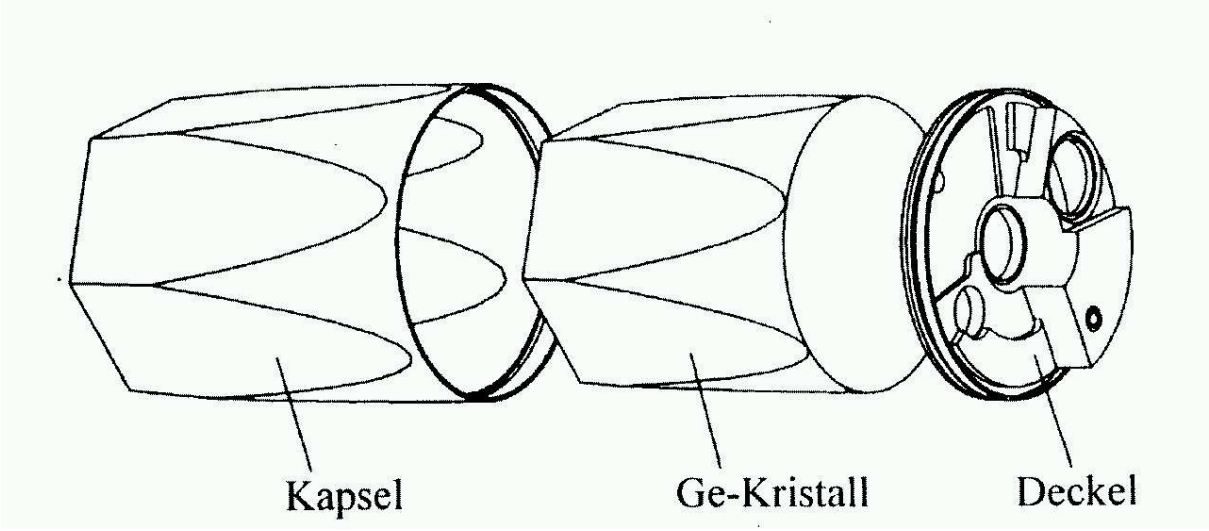

Abbildung 2.1: Aufbau eines MINIBALL Detektormoduls aus Aluminiumkapsel, hochreinem Germaniumkristall (HPGe) und Alu-Deckel.

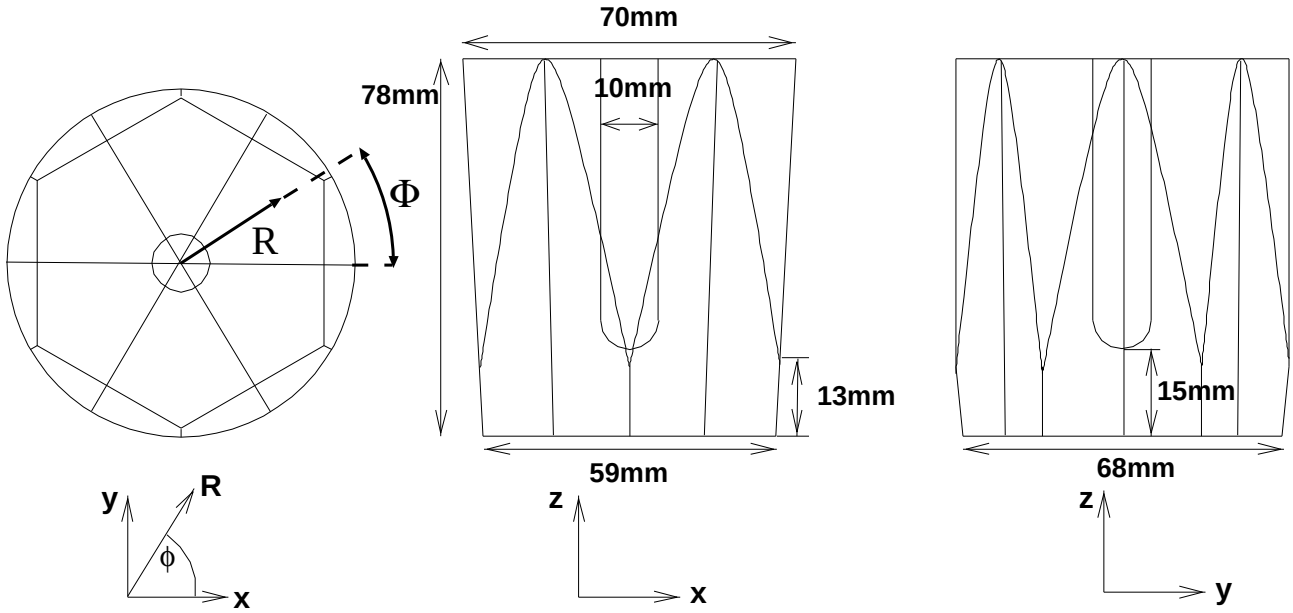

Abbildung 2.2: Geometrie des MINIBALL Kristalls. Man beachte die Definition des Koordinatensystems, sowie die Definitionen von R und Φ. Der Winkel Φ ist mit dem internen Segmentwinkel Φ_i ($0° \leq \Phi_i \leq 60°$) über $\Phi = \Phi_i + (i-1) \cdot 60°$ (i=1..6) verbunden. Das Bohrloch bildet den zentralen Kontakt, an dem die positive Sperrspannung angelegt wird und der deshalb wechselstrommäßig an den Vorverstärker gekoppelt ist. Die Segmentvorverstärker sind hingegen gleichstrommäßig mit den Kontakten 1-6 verbunden, die somit auf (virtuellem) Erdpotential liegen.

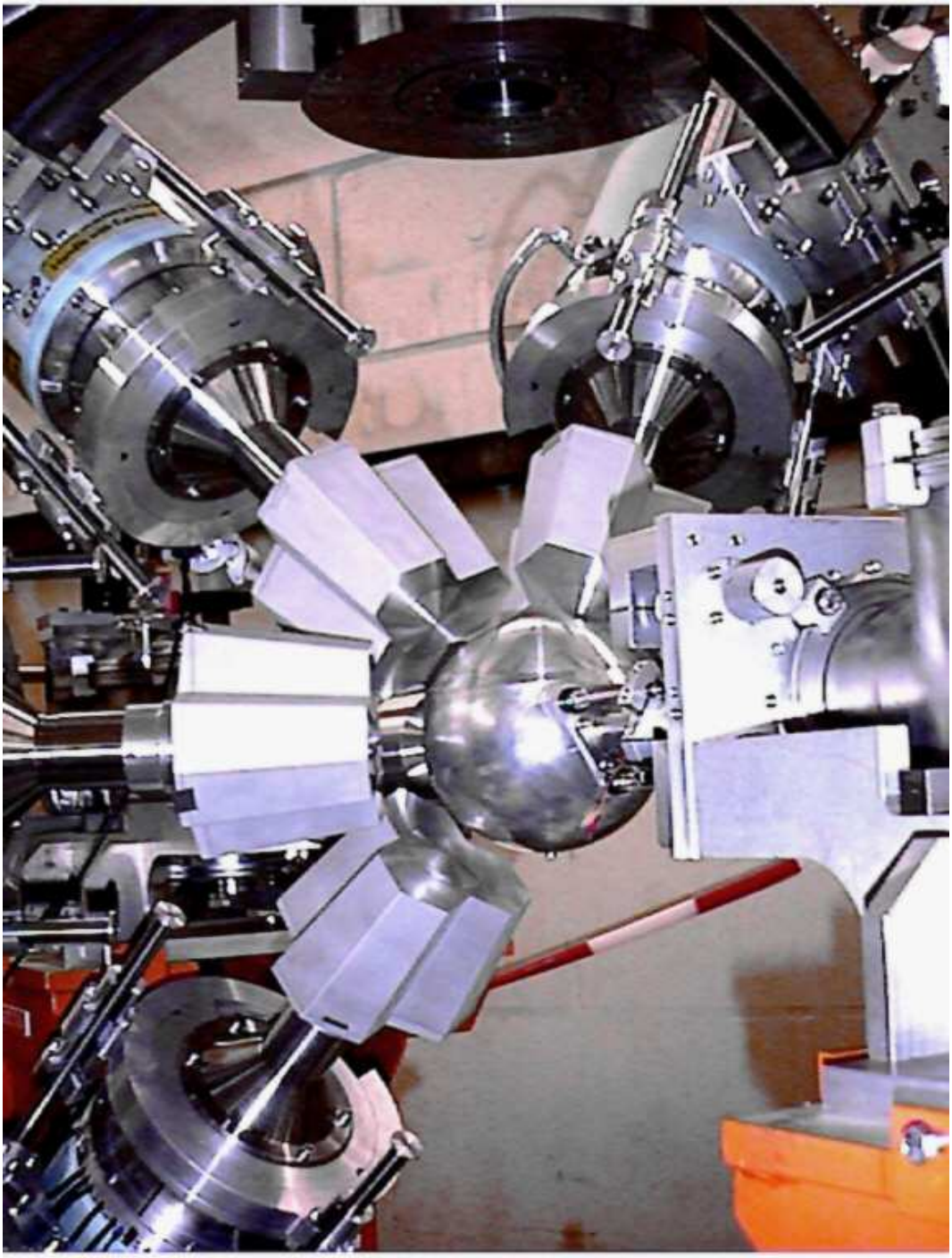

Abbildung 2.3: MINIBALL Testaufbau am IKP in Köln. Sichtbar sind das Strahlrohr, die Targetkammer, vier Triplecluster-Kryostaten und Teile der MINIBALL-Halterung.

Den groben Aufbau eines einzelnen MINIBALL Moduls mit Aluminiumkapsel und Endkappe inklusive der Vakuumdurchführung für das Signal des inneren (zentralen) Kontakts (auch Core oder Bohrung genannt) zeigt Abbildung 2.1. Im Bild ist allerdings der EUROBALL-Typ gezeigt. Die paarweise angeordneten Durchführungen der sechs Segmentsignale, wie sie der MINIBALL-Typ besitzt, fehlen in dieser Abbildung. Der MINIBALL-Kristall besteht aus hochreinem Germanium (HPGe für High Purity Germanium), wobei die Restverunreinigung in der Größenordnung von 10^{10} cm^{-3} liegt. HPGe-Detektoren sind im Vergleich zu den Lithium gedrifteten Germaniumdetektoren (Ge(Li)) einfacher zu handhaben, da diese – um ein driften des Lithiums zu unterbinden – dauerhaft gekühlt bleiben müssen. Die Kapselungstechnologie [3] erlaubt eine einfache Handhabung mehrerer Detektoren in einem einzigen Kryostaten. Das in der Kapsel vorhandene Vakuum schützt außerdem die empfindliche Oberfläche des Kristalls. Natürlich muss auf eine möglichst dünne Wandstärke (0.7 mm) geachtet werden, damit die Absorptionsverlustete in diesem "toten" Material gering bleiben. Die Geometrie des Germaniumkristalls ist in Abbildung 2.2 gezeigt. Der Kristall ist 78 mm lang, hat einen Durchmesser von 68 mm und ist hexagonal geschliffen. Die positive Sperrspannung ($\sim$ 4 keV) liegt an der zentralen Elektrode an, während die Außenkontakte gleichstrommäßig auf Erde liegen.

Der vollständige MINIBALL-Aufbau wird nach derzeitigen Planungen aus insgesamt 40 Detektoren, d.h. 240 Segmenten bestehen. Diese 40 Detektoren werden in acht Triple-Kryostaten und vier Quadruple-Kryostaten eingebaut sein. Im Bild 2.3 ist der Testaufbau am IKP in Köln gezeigt. Von rechts nach links ist der Verlauf des Strahlrohres, welches in die Targetkammer mündet, erkennbar. In Rückwärtsrichtung sind die Dreifachkryostaten positioniert, gehalten durch das von den Universitäten Köln und Straßburg entwickelte Gestell, welches eine Vielzahl verschiedener Anordnungen erlaubt. Der gemeinsame Betrieb mehrerer Detektoren in einem Kryostaten erlaubt durch die Add-Back-Methode eine Erhöhung der Effizienz. Allerdings wird es mit zunehmender Anzahl der Detektoren problematisch für eine ausreichende Kühlung und Verkabelung sorgen. Da zunehmende Größe und Gewicht eine einfache Handhabung unterbinden, wurde ein ursprünglich geplanter Kryostat mit sieben Detekoren wieder verworfen und stattdessen die flexiblere und leichtgewichtigere Lösung mit Kryostaten für drei oder vier Detektoren gewählt. Außerdem sind die äußeren Module nicht mehr auf das Target ausgerichtet, sobald ein von 42cm verschiedener Target-Detektor Abstand gewählt wird, für den der innere Radius des EURO**BALL** ausgelegt ist. Da die Tiefeninformation (die z-Koordinate in Abbildung 2.2) fehlt, wird die Ortsinformation zunehmend unscharf, was eine Dopplerkorrektur im geforderten Maße nicht mehr erlaubt hätte.

2.2 Nachweis von γ-Strahlung

Im Prinzip ist ein Germaniumdetektor eine in Sperrichtung betriebene p.i.n-Diode. Im Falle des MINIBALL Detektors handelt es sich, um ein großes Volumen von hochreinem[2] intrinsischem n-Typ[3] Germanium auf das die p^+- und n^+-Kontakte durch Aufbringung möglichst dünner Schichten von Bor und Lithium hergestellt wurden. Nach [4] hängt die

[2]Die verbleibende Verunreinigung ist in der Größenordnung von 10^{10} Atomen pro cm^3

[3]Die verbleibenden Verunreinigung sind Donatoren

Sperrschichtdicke d für planare Detektoren folgendermaßen von der Sperrspannung V und der Nettoverunreinigung N ab:

$$d = \sqrt{\frac{2\epsilon V}{eN}} \quad , \tag{2.1}$$

wobei mit ϵ die Dielektrizitätskonstante bezeichnet wird. Wenn die angelegte Spannung zur Sättigung der Driftgeschwindigkeit der Ladungsträger (besonders der Löcher) ausreichen soll und gleichzeitig genügend niedrig bleiben muss, sodass es nicht zu spontanen Überschlägen kommt, so bleibt nur noch die Nettoverunreinigung als freier Parameter in Gleichung 2.1 übrig. Hieran erkennt man, dass nur durch die Verwendung von hochreinem Germanium (und Anlegen einer genügend hohen Spannung) die Sperrzone so weit vergrößert werden kann, dass sie einen Detektor von der Größe des MINIBALL Typs nahezu vollständig überstreicht. Allerdings gilt für den MINIBALL-Detektor nicht die einfache Formel 2.1. Stattdessen muss die Feldverteilung durch eine Simulation bestimmt werden, die die besondere Form des MINIBALL Detektors einbezieht. Der gesperrte Bereich stellt die aktive Zone des Germaniumdetektors dar. In ihm können die von Photonen durch Wechselwirkung mit dem Detektormaterial erzeugten freien Ladungsträger durch das elektrische Feld getrennt werden bevor sie rekombinieren.

Die Wechselwirkung von Photonen mit Materie geschieht dabei hauptsächlich durch drei Prozesse [5], wobei beliebige (physikalisch erlaubte) Kombinationen dieser Prozesse stattfinden, bis schließlich die gesamte Energie eines Photons vollständig an den Kristall abgegeben wurde (FEP) oder das verbleibende Photon (Photonen) den Kristall verlassen hat (haben):

1. Photoeffekt: Die vollständige Absorption eines Photons durch ein stark gebundenes Elektron eines Atoms, bei gleichzeitiger Emission des Elektrons und nachfolgender Emission charakteristischer Röntgenstrahlung und/oder Augerelektronen.

2. Compton-Effekt: Die elastische Streuung eines Photons an einem (Hüllen-) Elektron. Dabei verliert das Photon einen Teil seiner Energie.

3. Paarerzeugung: Die spontane Erzeugung eines Elektron-Positron-Paares in der Nähe eines Kerns, wobei das Positron nach seiner Abbremsung in zwei Photonen von je 511 keV zerstrahlt.

Wie in Abbildung 2.4 erkennbar, dominiert der Photoeffekt den linearen Absorptionskoeffizienten bis zu einer Energie von 150 keV. Im mittleren Energiebereich von 150 keV bis 2,5 MeV wird der Absorptionskoeffizient durch den Compton-Effekt bestimmt. Bei noch höheren Energien wird der Prozess der Paarbildung dominant. Für alle Prozesse gilt, dass die Energieabgabe des Photons an den Detektorkristall über die Abbremsung der sekundär erzeugten schnellen Elektronen (Positronen) erfolgt, wobei in den Valenz- und Leitungsbändern des Germaniumkristalls Elektron-Loch-Paare erzeugt werden. Die Abbremsung eines schnellen Elektrons (Positrons) geschieht durch Stoßionisation und – aufgrund der geringen Masse – Emission von Bremsstrahlung. Die Energie bei der der Verlust durch Bremsstrahlung dem durch Stoßionisation entspricht, wird als die *kritische Energie* bezeichnet. Sie ist ungefähr durch [6]:

$$E_c \simeq \frac{800\ \text{MeV}}{Z + 1,2} \tag{2.2}$$

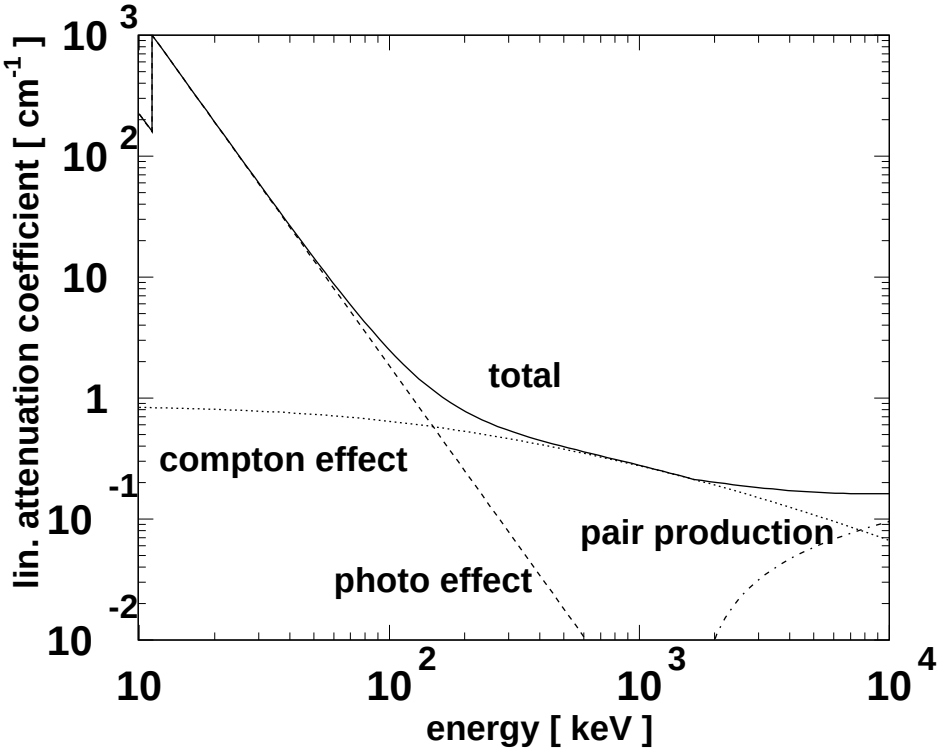

Abbildung 2.4: Absorptionskoefizienten für Germanium.

Z	Ordnungszahl des Materials; für Germanium ist Z=32
m_e	Ruhemasse des Elektrons
A	Atomgewicht des Detektormaterials; für Germanium ist A=72,61
r_e	Klassischer Elektronenradius: r_e=2,817 $\cdot 10^{-13}$ cm
c	Vakuumlichtgeschwindigkeit
β	Geschwindigkeit v des Elektrons/Positrons in Einheiten von c, also $\beta = \frac{v}{c}$
I	mittleres Anregungspotential des Atoms
C	Schalenkorrektur
δ	Dichtekorrektur

Tabelle 2.1: Bethe-Bloch-Parameter

gegeben. Für Germanium mit Z=32 ergibt sich ein Wert von 24,1 MeV. Für γ-Energien in der Größenordnung von einigen MeV (denn nur diese sind für den geplanten Einsatz von MINIBALL relevant) kann man den Verlust durch Bremsstrahlung somit vernachlässigen. Der mittlere spezifische Energieverlust durch Stoßionisation ($-\frac{dE}{dx}$) wird durch die Bethe-Bloch-Formel [6] beschrieben:

$$-\frac{dE}{dx} = 2\pi N_a r_e^2 m_e c^2 \rho \frac{Z}{A}\frac{1}{\beta}\left[\ln\left(\frac{E_{\text{kin}}^2(E_{\text{kin}}+2)}{2(I/m_e c^2)^2}\right) + F(E_{\text{kin}}) - \delta - 2\frac{C}{Z}\right] \quad . \tag{2.3}$$

Hier ist E_{kin} die kinetische Energie des Elektrons/Positrons in Einheiten von $m_e c^2$. F ist gegeben durch

$$F(E_{\text{kin}}) = 1 - \beta^2 + \frac{\frac{E_{\text{kin}}^2}{8} - (2r+1)\ln 2}{(E_{\text{kin}}+1)^2} \tag{2.4}$$

für Elektronen und

$$F(E_{\text{kin}}) = 2\ln 2 - \frac{\beta^2}{12}\left(23 + \frac{14}{E_{\text{kin}}+2} + \frac{10}{(E_{\text{kin}}+2)^2} + \frac{4}{(E_{\text{kin}}+2)^3}\right) \tag{2.5}$$

für Positronen. Die einzelnen Parameter und ihre Bedeutung sind in Tabelle 2.1 aufgelistet.

Durch Integration dieser Gleichung läßt sich die mittlere Reichweite der Elektronen (Positronen) im Germanium bestimmen. Bei E_{kin}=1 MeV liegt diese bei etwa einem mm (bei niedrigeren Energien noch weniger) und bei 10 MeV bei etwa 1 cm. Die Elektron-Loch-Paare, die bei der Abbremsung der sekundären Elektronen (Positronen) entstehen, sind also räumlich auf einige mm lokalisiert. Die Elektron-Loch-Paare werden im Feld der Verarmungszone getrennt bevor sie rekombinieren können und führen zu einem Stromimpuls auf dem zentralen Kontakt, dessen Zeitintegral proportional zu der im aktiven Bereich des Detektors deponierten Energie des primären γ-Quants ist.

Ausgehend vom physikalischen Prozess der Ladungserzeugung, ist es noch von Interesse mit welcher Genauigkeit die Energie eines γ-Quants bestimmt werden kann, wenn die gesamte Energie des γ-Quants im Germaniumkristall absorbiert wurde (Full Energy Peak (FEP) - Ereignisse). Die Auflösung eines HPGe γ-Spektrometers wird hauptsächlich durch die folgenden Punkte bestimmt [7]

1. *Statistik* Die intrinsische Signalverbreiterung Γ_{intr} kommt dadurch zustande, dass beim Prozeß der Ladungserzeugung ein Teil der Energie der Photonen in Vibrationsanregungen des Halbleiters (Phononen) umgesetzt wird. Da es sich bei Germanium

um einen indirekten Halbleiter mit einer Bandlücke von [8] 0,67 eV bei $T = 300$ K bzw. von 0,75 eV bei T = 0 K handelt, sind immer Phononen beim Übergang eines Elektrons vom Valenz- ins Leitungsband beteiligt. Die resultierende Verbreiterung (FWHM) ist gegeben durch

$$\Gamma_{\text{intr}} = 2,35 \cdot E_{\text{paar}} \cdot \Delta N_{\text{paar}} = 2,35 \cdot \sqrt{F E_{\text{dep}} E_{\text{paar}}}\,, \tag{2.6}$$

wobei E_{paar} die durchschnittliche Energie ist, um ein Elektron-Loch Paar zu erzeugen[4]. E_{dep} ist die durch die Wechselwirkung deponierte Energie, N_{paar} die mittlere Anzahl der erzeugten Ladungsträger ($N_{\text{paar}} = \frac{E_{\text{dep}}}{E_{\text{paar}}}$), ΔN_{paar} die statistische Schwankung der Anzahl der erzeugten Ladungsträger, die durch $\Delta N_{\text{paar}} = \sqrt{F \cdot \frac{E_{\text{dep}}}{E_{\text{paar}}}}$ gegeben ist und schließlich ist F der Fano-Faktor [9] dessen Wert für Germanium $F \sim 0,12$ beträgt.

2. *Unvollständige Ladungssammlung* In großen Germaniumdetektoren kann es durch drei Prozesse zu unvollständiger Ladungssammlung kommen. Erstens können Ladungsträger in Gitterfehlstellen gefangen werden (*Trapping*). Falls sie gefangen bleiben, kann dies zu einem Verlust an Signalamplitude oder falls sie wieder freigelassen werden zu einen verlangsamten Anstieg des Pulses führen. Zweitens kann es zu verlängerten Anstiegszeiten (Sammelzeiten) kommen, wenn die Photonen in Bereichen mit geringer elektrischer Feldstärke wechselwirken. Solche Bereiche sind typischerweise die Kanten des Detektors (siehe [10]). Schließlich können Ladungsträger auch noch durch Rekombination verloren gehen. Dieser Beitrag wird im fogenden mit Γ_{cc} bezeichnet.

3. *Ballistisches Defizit*[5] Die Signalverbreiterung durch das ballistische Defizit Γ_{bd} ist proportional zur Amplitude und Variation der Anstiegszeit (Abhängig von der Position der Wechselwirkungen) des Signals. Dieser Beitrag dominiert somit die Auflösung bei *hohen* Energien.

4. *Rauschen der Elektronik* Die Signalverbreiterung durch das Rauschen der Elektronik Γ_{elek} ist unabhängig von der Energie der Photonen und dominiert damit die Auflösung bei *kleinen* Energien.

Somit stellt sich die gesamte Signalverbreiterung folgendermaßen dar:

$$\Gamma^2_{\text{total}} = \Gamma^2_{\text{intr}} + \Gamma^2_{\text{cc}} + \Gamma^2_{\text{bd}} + \Gamma^2_{\text{elek}} \tag{2.7}$$

Um eine möglichst hohe Auflösung und damit ein möglichst kleines Γ_{total} zu erreichen, müssen die Beiträge 2 – 4 minimiert werden, was einen sorgfältigen Bau der Germaniumdetektoren und der nachfolgenden Vorverstärker erfordert. Das ballistische Defizit kann darüber hinaus (wenigstens teilweise) durch Analyse des Detektorsignals korrigiert werden. Die intrinsische Auflösung von Germanium (Gleichung 2.6), die bei E_γ= 1 MeV etwa [9] $\Gamma_{\text{intr}} = 1,4$ keV beträgt, kann natürlich nicht unterschritten werden.

[4]Sie ist die Summe aus der effektiven Bandlücke (0,93 eV), dem mittleren Energieverlust durch Phononproduktion (0,08) und der mittleren kinetischen Energie des Elektron-Loch Paares (1,95) und beträgt 2,96 eV bei 77 K.

[5]Zur Definition siehe Kapitel 3.2

	Elektronen (e)	Löcher (h)
E_0	275 V/cm	210,5 V/cm
β	1,32	1,36
μ_0	$3{,}6 \cdot 10^4$ cm^2/Vs	$4{,}2 \cdot 10^4$ cm^2/Vs

Tabelle 2.2: Parameter des isotropen Fits (nach [4]).

2.3 Pulsformanalyse zur Erhöhung der Detektorgranularität

Um die Granularität des MINIBALL-Moduls zu erhöhen, wird – neben der sechsfach Segmentierung des äußeren Kontakts des Germaniumzählers – eine Pulsformanalyse durchgeführt. Ausführliche Simulationen [11] und erste Messungen [11, 12] haben gezeigt, dass damit im Mittel eine 60 bis 100 fache Unterteilung des vom einzelnen Detektormodul überdeckten Raumwinkels erreichbar ist.

2.3.1 Signalentstehung

Das Ergebnis der Wechselwirkung eines Photons mit dem Detektor ist (sind) eine (mehrere) Ladungswolke(n) die durch das elektrische Feld im Detektor getrennt werden, bevor die Elektronen mit den Löchern rekombinieren können. Dieses Feld kann durch Lösen der Poisson Gleichung für das *elektrostatische* Potential Φ_{sc}[6] berechnet werden

$$\Delta\Phi_{\mathrm{sc}}(\vec{r}) = -\frac{\rho(\vec{r})}{\epsilon_0\epsilon_{\mathrm{r}}} = -\frac{eN_c}{\epsilon_0\epsilon_{\mathrm{r}}} \quad , \tag{2.8}$$

wobei ϵ_0 die dielektrische Konstante des Vakuums, ϵ_{r} die dielektrische Konstante von Germanium ($\epsilon_{\mathrm{r}} = 16$), $\rho = eN_c = const.$ die Raumladung und N_c die Konzentration an Verunreinigungen im Detektor ist. Aus dem Potential errechnet sich das elektrische Feld nach

$$\vec{E}_{\mathrm{sc}}(\vec{r}) = -\vec{\nabla}\Phi_{\mathrm{sc}}(\vec{r}) \quad . \tag{2.9}$$

Unter dem Einfluß des Feldes $\vec{E}_{\mathrm{sc}}(\vec{r})$ driften die Ladungsträger auf die Kontakte zu. Ihre Driftgeschwindigkeit als Funktion des Feldes ist für einen isotropen Fit an die Messdaten (der die Richtungabhängigkeit der Driftgeschwindigkeit nicht einbezieht) gegeben durch

$$\vec{v}^{\,e,h}_{\mathrm{drift}}(\vec{E}) = \frac{\mu_0^{e,h}\vec{E}_{\mathrm{sc}}}{\left[1 + \left(\frac{|\vec{E}_{\mathrm{sc}}|}{E_0^{e,h}}\right)^{\beta_{e,h}}\right]^{\frac{1}{\beta_{\mathrm{e,h}}}}} \quad , \tag{2.10}$$

wobei die entsprechenden Werte in Tabelle 2.2 angegeben sind.

Ist die Driftgeschwindigkeit bekannt, lässt sich der Weg, den die Ladungen zurücklegen, bei bekannter Anfangsposition $\vec{r}_0 = \vec{r}^{\,e,h}(t=0)$ durch

$$\frac{d}{dt}\,\vec{r}^{\,e,h}(t) = \vec{v}^{\,e,h}_{drift}(\vec{E}(\vec{r}^{\,e,h}(t))) \tag{2.11}$$

[6] sc: space charge, da dieses Potential von der Raumladung abhängt

berechnen.

Für die induzierte Ladung auf den Kontakten spielt die Raumladung und damit Φ_{sc} keine Rolle [13]. Um die sich zeitlich ändernde induzierte Ladung auf den Elektroden (und damit das Stromsignal des Germaniumdetektors) berechnen zu können, bedient man sich des *weighting* Potentials Φ_{weight}. Das weighting Potential berechnet sich durch Null setzten der Raumladung und des Potentials auf allen Elektroden mit Ausnahme des betrachteten Kontakts. Für die verbleibende Elektrode wird das Potential von 1 Volt angenommen. Es ergibt sich [14, 15], dass sich der induzierte Ladungspuls q_i^{ind} auf der i-ten Elektrode als Funktion der N[7] driftenden Ladungen q_j^{drift} nach

$$q_i^{\text{ind}}(t) = -\sum_{j=1}^{N} q_j^{\text{drift}} \cdot \Phi_{\text{weight,i}}(\vec{r}_j(t)) \tag{2.12}$$

berechnen lässt, woraus sich direkt der zugehörige Strom ableitet:

$$i_i^{\text{ind}}(t) = \frac{\mathrm{d}}{\mathrm{dt}}\, q_i^{\text{ind}}(t) = -\sum_{j=1}^{\mathrm{N}} q_j^{\text{drift}} \left[\vec{\nabla}\Phi_{\text{weight,i}}(\vec{r}_j(t)) \cdot \vec{v}_{\text{drift}}(\vec{E}(\vec{r}_j(t)))\right] \tag{2.13}$$

$\vec{E}_{\text{weight,i}}(\vec{r}_j(t)) = \vec{\nabla}\Phi_{\text{weight,i}}(\vec{r}_j(t))$ ist das weighting Feld. Das Stromsignal des Detektors wird schließlich durch die Vorverstärkerelektronik zu einem Ladungssignal aufintegriert, um eine optimale Energieauflösung zu erhalten. Diese berechnet sich demnach durch Faltung mit der Impulsantwort H(t) des Vorverstärkers:

$$q_i^{\text{VV}}(t) = \int_0^t i_i^{\text{ind}}(t') \cdot H(t-t')dt' \quad . \tag{2.14}$$

Typische Driftgeschwindigkeiten sind kleiner als 10^7 cm/sec, sodass sich Sammelzeiten – und damit ein Stromfluss über einen Zeitraum – von einigen 100 ns ergeben. Da die Integrationszeit des Vorverstärkers viel kleiner ist ($\sim$ ns), kann das Stromsignal, zumindest teilweise, durch Ableitung aus dem Ladungssignal des Vorverstärkers gewonnen werden. Wie in Formel 2.13 erkennbar, hängt die Form des Detektorpulses vom Driftweg der Ladungen und damit von der ursprünglichen Position der Ladungserzeugung ab. Diese Information ist ebenfalls im (Vorverstärker-) Ladungspuls enthalten. Durch Analyse des ansteigenden Teiles des (Vorverstärker-) Ladungspulses kann somit auf die Position der Wechselwirkung eines γ-Quants zurückgeschlossen werden.

2.3.2 Bestimmung der Wechselwirkungkoordinaten R und Φ

In langer und detaillierter Vorarbeit [16, 17, 10, 11, 12] wurden Verfahren entwickelt, die es erlauben den Radius R und den Winkel Φ der ersten Wechselwirkung eines γ-Quants im MINIBALL-Detektor zumindest näherungsweise zu bestimmen.

Man stelle sich zur Erläuterung zunächst ein Ereignis vor, bei dem das primäre γ-Quant seine gesamte Energie in einer einzigen Wechselwirkung im gesperrten (aktiven) Bereich des MINIBALL Detektors deponiert (z.B. Photoeffekt, E_γ=1 MeV). Dies sei in Segment

[7] Erinnerung: Jede Wechselwirkung erzeugt *zwei* driftende Ladungswolken – Elektronen und Löcher

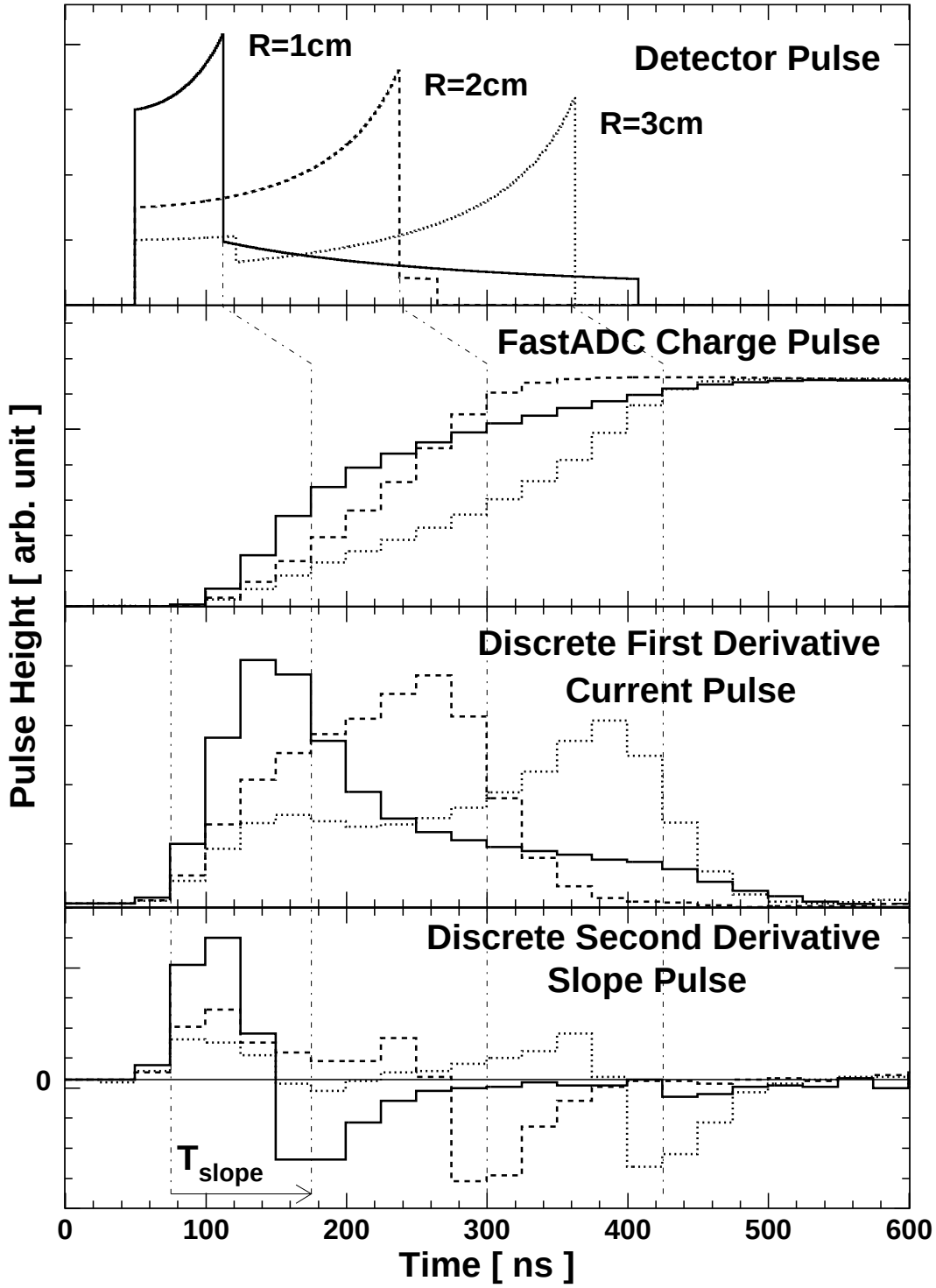

Abbildung 2.5: In der oberen Abbildung sind die Strompulse des MINIBALL Detektors für drei verschiedene Wechselwirkungsradien gezeigt. Man erkennt, dass die Zeit bis zum Auftreffen der Elektronen auf den inneren Kontakt zunimmt und die Driftzeit der Löcher abnimmt. Ebenfalls zu erkennen ist, dass der Beitrag der Elektronen deutlich überwiegt. Direkt darunter ist der digitalisierte Vorverstärkerladungspuls gezeigt, der zur Pulsformanalyse benutzt wird. Aus diesem läßt sich durch Ableitung wieder das Stromsignal zurückgewinnen, was in der dritten Box zu sehen ist. Ganz unten ist die Ableitung des Stromsignals gezeigt. Der Zeitpukt des steilsten Abfalls des Stromsignals, welcher die Radiusinformation liefert, ist mit dem Zeitpunkts des Minimalwertes dieses Signals korreliert.

Nummer 2 geschehen. Das erzeugte schnelle Elektron erzeugt nun seinerseits lokal (Erinnerung: $E_{kin} = 1$ MeV $\rightarrow$ r $\sim$ 1 mm) Elektron-Loch-Paare, welche durch das elektrische Feld getrennt werden und entlang der Feldlinien in Richtung der Kontakte driften. Die driftenden Ladungen erzeugen nach Gleichung 2.13 auf den Kontakten einen Strom. Es werden ideale Vervorstärker benutzt, d.h. es handelt sich in Gleichung 2.14 um eine echte Integration.

Die radiale Auflösung

Die radiale Position R der Wechselwirkung wird durch Analyse des Signals des Core-Kontakts bestimmt. Für den zentralen Kontakt (Kontakt Nummer 7), auf den die Elektronen zudriften, da an ihm im Falle des MINIBALL Detektors die positive Sperrspannung anliegt, ergibt sich für das oben gewählte Beispiel der Ausdruck

$$i_7^{\mathrm{ind}} = q_e^{\mathrm{drift}} \left[\vec{\nabla} \Phi_{\mathrm{weight},7}(\vec{r}_e) \cdot \vec{v}_{\mathrm{drift}}(\vec{E}(\vec{r}_e)) \right] - q_l^{\mathrm{drift}} \left[\vec{\nabla} \Phi_{\mathrm{weight},7}(\vec{r}_l) \cdot \vec{v}_{\mathrm{drift}}(\vec{E}(\vec{r}_l)) \right] \quad , \quad (2.15)$$

wobei die Zeitabhängigkeit der besseren Lesbarkeit wegen nicht nochmal explizit aufgeführt wurde. In Abbildung 2.5 sind im oberen Teil die Strompulse für drei verschiedene Wechselwirkungsradien gezeigt. Da die Löcher in Richtung des äußeren Kontakts des Segments Nummer 2 driften, wird ihr Einfluß auf den Gesamtstrompuls mit steigender Distanz zum zentralen Kontakt geringer. Dies erklärt sich aus der Tatsache, dass das weighting Feld des zentralen Kontakts in Richtung der äußeren Kontakte hin abnimmt (Endwert 0 V) und seinen Maximalwert natürlich am zentralen Kontakt erreicht (1 V). Treffen nun die Elektronen auf den zentralen Kontakt, so verschwindet plötzlich der durch ihre Drift verursachte Strom. Haben die Löcher den äußeren Kontakt schon vorher erreicht (Wechselwirkung näher am Segmentkontakt als am zentralen Kontakt, gilt natürlich nur für gleiche Driftgeschwindigkeiten von Elektronen und Löcher und radiale Driftwege), so endet der Stromfluß im Detektor. Aus der Zeit zwischen Beginn des Strompulses und Ende des Strompulses läßt sich in diesem Falle, bei konstanter Driftgeschwindigkeit und radialem Driftweg, die Distanz der Wechselwirkung zum zentralen Kontakt berechnen. Hat die Wechselwirkung in der Nähe des zentralen Kontaktes stattgefunden, so fließt, nach dem Auftreffen der Elektronen auf den zentralen Kontakt, immer noch der durch die Löcher verursachte Strom. Durch das Auseinanderdriften der Ladungswolken befinden sich die Löcher aber in Bereichen des Detektors mit geringerer weighting Feldstärke. Die entscheidende Idee ist nun, die *Änderung* des Stromsignales $\frac{\mathrm{d}}{\mathrm{dt}} i$ zu untersuchen. Aus Gleichung 2.15 folgt direkt, dass

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{dt}} i_7^{\mathrm{ind}}(t) \;=\; & \frac{\mathrm{d}}{\mathrm{dt}} q_e^{\mathrm{drift}} \cdot \left[\vec{\nabla} \Phi_{\mathrm{weight},7}(\vec{r}_e(t)) \cdot \vec{v}_{\mathrm{drift}}(\vec{E}(\vec{r}_e(t))) \right] \\ & + q_e^{\mathrm{drift}} \cdot \triangle \Phi_{\mathrm{weight},7} \cdot v_{\mathrm{drift}}^2 \\ & + q_e^{\mathrm{drift}} \cdot \vec{\nabla} \Phi_{\mathrm{weight},7} \cdot \vec{\nabla} \vec{v}_{\mathrm{drift}} \cdot \vec{v}_{\mathrm{drift}} \\ & + q_e^{\mathrm{drift}} \cdot \vec{\nabla} \Phi_{\mathrm{weight},7} \cdot \frac{\mathrm{d}}{\mathrm{dt}} \vec{v}_{\mathrm{drift}} \\ & - \text{Beitrag der Löcher} \; , \end{aligned}$$

ist. Da für das weighting Potential die Raumladung ρ zu Null angenommen wird, d.h. $\triangle \Phi_{\mathrm{weight},7} = 0$, und unter der Annahme, dass die Driftgeschwindigkeit konstant und isotrop

ist, folgt der einfache Ausdruck

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{dt}} i_7^{\mathrm{ind}}(t) &= -q_e^{\mathrm{drift}} \cdot \left[\vec{E}_{\mathrm{weight},7}(\vec{r}_e(t)) \cdot \vec{v}_{\mathrm{drift}}(\vec{E}(\vec{r}_e(t)))\right] \delta(t - t_e^{\mathrm{drift}}) \\ &\quad + q_l^{\mathrm{drift}} \cdot \left[\vec{E}_{\mathrm{weight},7}(\vec{r}_l(t)) \cdot \vec{v}_{\mathrm{drift}}(\vec{E}(\vec{r}_l(t)))\right] \delta(t - t_l^{\mathrm{drift}}) \quad ,\end{aligned}$$

wobei mit t_e^{drift} bzw. t_l^{drift} die Driftzeit der Elektronen bzw. Löcher bis zu den entsprechenden Kontakten bezeichnet wird. Hieraus liest sich ab, dass im Augenblick des Auftreffens der Elektronen auf den zentralen Kontakt, die Ableitung des Stromes ein absolutes Minimum hat, da beide driftenden Ladungen q_e und q_l gleich groß sind und das weighting Feld in der Nähe des Core-Kontakts, d.h. für die driftenden Elektronen, größer als das für die Löcher ist.

Da die Driftgeschwindigkeit der Elektronen in dem relevanten Feldbereich nahezu isotrop und konstant ist [11], ist die Zeit t_e^{drift}, d.h. die Zeit zwischen Beginn des Pulses und dem Minmum von $\frac{d}{dt} i_7(t)$ (T_{slope} in Abbildung 2.5), proportional zum Abstand R des Wechselwirkungspunkts vom zentralen Kontakt.

Die Winkelauflösung

Die Winkelinformation Φ wird durch Analyse der Signale der benachbarten Segmente gewonnen, in unserem Beispiel (Wechselwirkung in Segment 2) sind dies die Segmente 1 und 3. Von Bedeutung ist hier das weighting Feld dieser Segmente am Ort der driftenden Ladung im getroffenen Segment. Das weighting Feld eines äußeren Kontakts ist in der Abbildung 3.12 in [11] gezeigt.

Beginnend vom Ort der Wechselwirkung $\vec{r}_e(0) = \vec{r}_l(0)$ driften die Ladungen innerhalb des Segments 2 zum inneren Kontakt bzw. zu dem Segmentkontakt zu. Die auf den benachbarten Kontakten j (j$\neq$2) induzierte Ladung (ideale Vorverstärker) berechnet sich nach Formel 2.12 zu:

$$q_j^{\mathrm{ind}}(t) = -q_e^{\mathrm{drift}} \cdot \Phi_{\mathrm{weight,j}}(\vec{r}_e(t)) - q_l^{\mathrm{drift}} \cdot \Phi_{\mathrm{weight,j}}(\vec{r}_l(t)) \quad , \tag{2.16}$$

wobei $\vec{r}_e(t)$ bzw. $\vec{r}_l(t)$ die Positionen der Elektronen und Löcher nach der Driftzeit t bezeichnen und $|q_e^{\mathrm{drift}}| = |q_l^{\mathrm{drift}}|$ ist. Wenn sowohl die driftenden Elektronen als auch die Löcher an dem zentralen Kontakt bzw. dem Segmentkontakt 2 angekommen sind, gilt

$$q_i^{\mathrm{ind}} = 0 \quad \text{für j}\neq 2, \tag{2.17}$$

da $\Phi_{\mathrm{weight,j}}(\vec{r}_e(t_e^{\mathrm{drift}})) = \Phi_{\mathrm{weight,j}}(\vec{r}_l(t_l^{\mathrm{drift}})) = 0$. Obwohl nach Ablauf der Driftzeiten die auf den benachbarten (j$\neq$2) Segmentkontakten induzierte Ladung Null wird, wird zuvor sehr wohl eine messbare Ladung auf den Segmentkontakten induziert. Durch detaillierte Simulationen [11] wurde herausgefunden, dass sich aus dem Verhältnis der auf den unmittelbar benachbarten Segmenten induzierten Ladung eine Winkelinformation gewinnen läßt. Dazu wird das Verhältnis der maximalen Ladungspulsamplituden ($\max(|q_{\mathrm{ind}}|)$) auf den Segmentkontakten 1 und 3 gebildet und das Ergebnis logarithmiert. Die so erhaltenen Funktion ist, bei gegebenem Abstand R der Wechselwirkung vom zentralen Kontakt, nahezu linear mit dem internen Segmentwinkel korreliert. Da das den Wechselwirkungspunkt enthaltene Segment über die auf dem Segmenkontakt gesammelte Ladung identifizierbar ist, hat man mit dem internen Segmentwinkel auch den Gesamtwinkel Φ bestimmt.

Die Hauptwechselwirkung

Wie in 2.2 diskutiert, werden γ-Quanten im allgemeinen in mehreren Stufen absorbiert, wodurch die Energie an verschiedenen Orten im Detektor deponiert wird. Um die Flugrichtung des primären γ-Quants zu bestimmen, denn diese wird zur Korrektur des Dopplereffekts benötigt, müsste also eigentlich die Position der ersten Wechselwirkung im Detektor bestimmt werden. Es hat sich herausgestellt, dass dies im allgemeinen nicht möglich ist. Das Konzept besteht deshalb darin, anstelle der ersten Wechselwirkung die Position der *Hauptwechselwirkung* (HW) [17] zu bestimmen. Die HW ist definiert als die Wechselwirkung eines FEP-Ereignisses, bei der der größte Anteil der Energie des eingestrahlten Photons deponiert wird und bei der deshalb die meiste Ladung erzeugt wird.

Detaillierte Simulationsrechnungen [17, 10, 11] haben gezeigt, dass die Koordinaten R und Φ der Hauptwechselwirkung eine hinreichend gute Näherung für die entsprechenden Koordinaten der ersten Wechselwirkung sind. Zu den größten Abweichungen kommt es bei Energien um 300 keV. Bei kleineren Energien beginnt der Photoeffekt den linearen Absorptionskoeffizienten zu dominieren (d.h. erste WW $\equiv$ HW), bei höheren Energien wird das primäre γ-Quant in seiner Flugrichtung zunehmend weniger beeinflußt, wenn die HW nicht die erste Wechselwirkung ist.

Statt der ersten Wechselwirkung wird man also versuchen, wenigstens die Koordinaten der HW aus den Detektorsignalen zu extrahieren. In offline Analysen wurde bereits gezeigt [17, 10, 11, 12], dass dies durch eine Verallgemeinerung der oben, für lokalisierte Ereignisse, diskutierten Verfahren möglich ist. Die Realisierung dieses Verfahrens in Echtzeit unter Benutzung des XIA DGF-4C Moduls der Firma X-ray Instrumentation Associates [18] war das Ziel der vorliegenden Arbeit.

2.3.3 Algorithmen zur Bestimmung von R und Φ

Der Radiusalgorithmus: Steepest Slope des Strompulses

Der Steepest Slope - Algorithmus basiert auf der Tatsache, dass in dem Augenblick, indem die zur Hauptwechselwirkung gehörenden Ladungsträger den inneren Kontakt erreichen, der größte Beitrag zum Strom verschwindet. Der Wegfall des Stromes der Hauptwechselwirkung verursacht den größten Abfall des Stromsignals. Um diesen Zeitpunkt festzustellen, muss nach dem absoluten Minimum der ersten Ableitung des Strompulses gesucht werden. Dies erfolgt über den Zeitraum von Beginn des Pulses bis zur vollständigen Sammlung aller Ladungen. Da die Driftgeschwindigkeit konstant ist, ist die Zeit T_{slope} zwischen Beginn des Pulses (T_0, Erzeugung aller Ladung) und dem größten Abfall des Strompulses (Auftreffen der Elektronen auf den Core) ein Maß für die Länge des Driftweges der Elektronen zum Core-Kontakt und somit ein Maß für den Abstand zur inneren Bohrung. Dieser Algorithmus liefert einen Wert für den Radius R aus Abbildung 2.2.

Die gegenwärtige Vorverstärkerelektronik liefert ein Ladungssignal, welches digitalisiert wird. Aus diesem digitalen Ladungssignal kann man, durch Bildung der zweiten Ableitung aus den ADC Samples,

$$\mathrm{PSA(n)} = \mathrm{ADC(n)} - 2 \cdot \mathrm{ADC(n-1)} + \mathrm{ADC(n-2)} \quad , \tag{2.18}$$

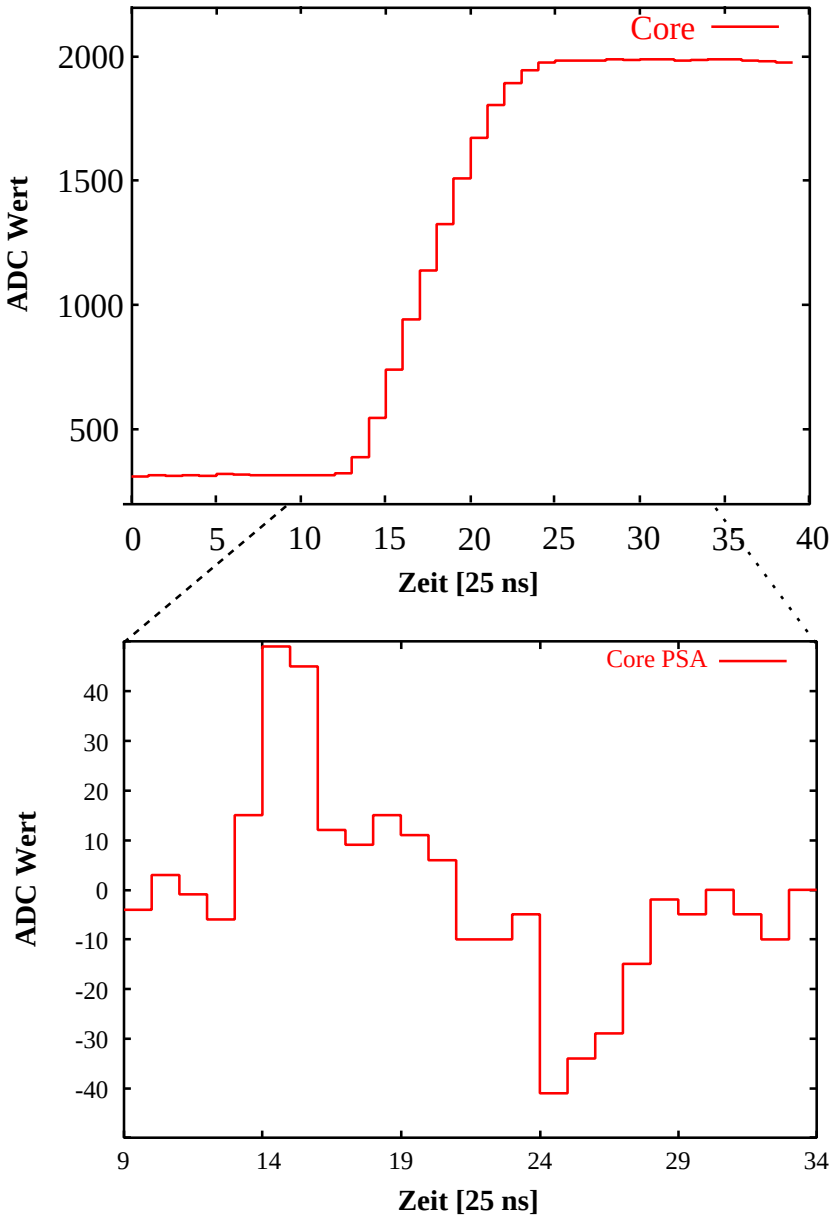

Abbildung 2.6: Ladungssignal des Cores (oben) plus Inhalt des zugehörigen on-board PSA Puffers (unten). Die on-board Pulsformanalyse speichert die zweite Ableitung des Coresignals erst ab der Endposition der Baselineberechnung. Es muß deshalb ein Offset von 9 Samples hinzuaddiert werden.

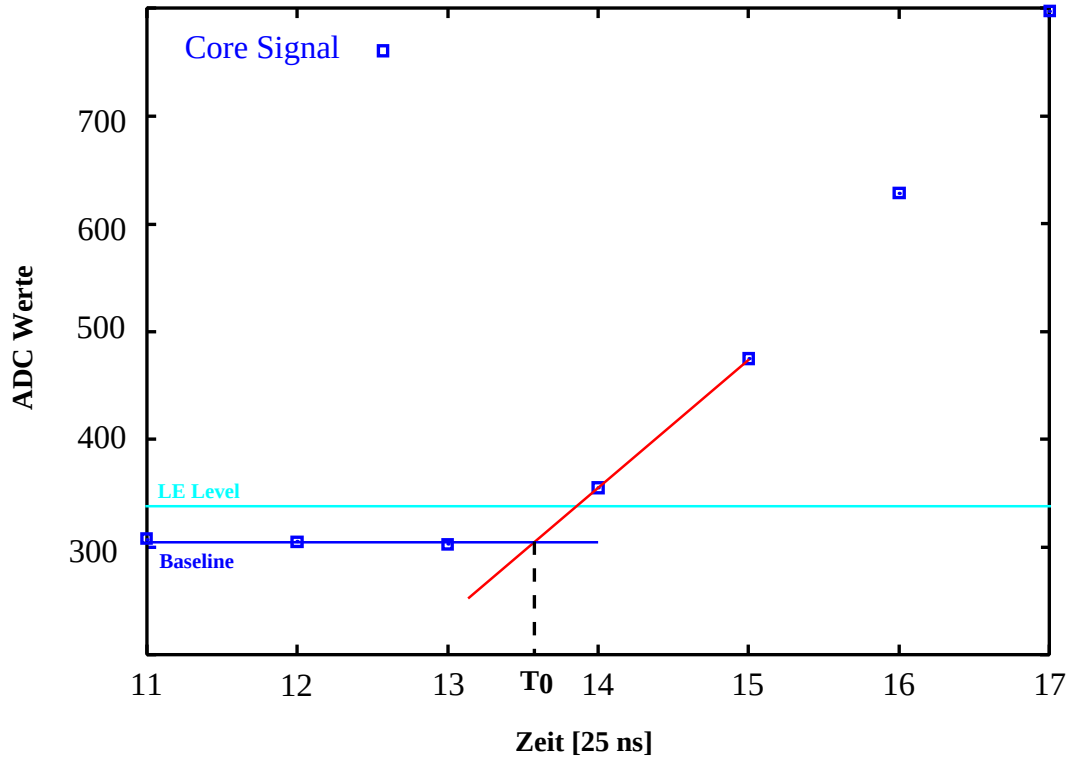

Abbildung 2.7: Schema des Startalgorithmus.

die Position des Steepest Slope im Stromsignal finden. Eine genauere Position des Steepest Slope wird durch Einbeziehung der beiden benachbarten Samples des "Slope"-Signals ermittelt, indem ein Polynom zweiten Grades durch diese Punkte gelegt wird [19]. In Abbildung 2.6 ist ein Core-Signal und seine zweite Ableitung (die allerdings erst ab Position 9 beginnt) gezeigt. Der Steepest Slope des Stromsignals findet zum Zeitpunkt 24 statt.

Der Beginn des Pulses wird nicht aus dem Steepest Rise (Zeitpunkt 14), sondern durch ein abgewandeltes LE-Verfahren direkt aus dem Ladungspuls bestimmt. Da der Ladungspuls direkt digitalisiert wird, ist die Differentationszeit durch die Samplingrate festgelegt, kann also nicht an den Vorverstärker angepasst werden. Das Ladungssignal enthält ausserdem die volle Bandbreite der ADCs. Da die Pulsform eines MINIBALL-Detektorsignales in Abhängigkeit des Ortes der Wechselwirkung variieren kann, ist es naheliegende die Schwelle des LE-Triggers möglichst niedrig zu legen. In [20] wird allerdings darauf hingewiesen, dass die Form des Pulses zu Beginn durch den Vorverstärker und seine Zeitkonstante τ bestimmt ist. Erst nach ungefähr 1.5 τ wird der Puls durch das eigentliche Detektorsignal dominiert, vorher wird der Anstieg durch den Vorverstärker begrenzt. Somit sollte ein LE-Trigger mit extrem niedriger Schwelle ein positionsunabhängiges Zeitsignal liefern können, sofern sein Anstieg die Anstiegsfähigkeiten des Vorverstärkers übertrifft.

Zur Zeit werden zwei ADC-Samples (Die niedrige Schwelle rechtfertigt den linearen Ansatz sofern diese zu Samples, die im Intervall $[0,\tau]$ liegen, führt. Ein Übergang zu höheren Ordnungen ist mit vermehrtem Rechenaufwand machbar.) und zwei Parameter benutzt: Ein Parameter gibt die Höhe einer Schwelle an, die zur Detektion des Pulses benötigt wird (Ideal: erster ADC-Wert > Rauschen). Der zweite Parameter gibt an, aus wievielen ADC-

Werten (beginnend ab Trace-Start) der DSP den Mittelwert der Baseline ermitteln soll. Ist der Wert der Baseline bekannt, so wird der Puls nach dem Sample abgesucht, dessen ADC-Wert größer als Baseline plus Schwellenwert ist. Das Ergebnis der lineare Interpolation vom Pulswert über der Schwelle auf das Niveau der Baseline gibt die Zeit des Pulsbeginns T_0 an. In Abbildung 2.7 ist der Vorgang zur Bestimmung der Startzeit skizziert.

Der Winkelalgorithmus

Dieser Algorithmus wird benutzt, um eine Information über den Winkel Φ_i innerhalb eines Segments i zu erhalten. Dabei wird ausgenutzt, dass es der MINIBALL-Detektor durch die Segmentierung der äußeren Kontakte erlaubt, aus den einzelnen Segmentsignalen den maximalen Betrag der in diesem Segment deponierten Ladung zu bestimmen. Das Segment, indem die höchste Ladung deponiert wurde, wird der HW zugeordnet. Durch die Pulsformanalyse der induzierten Ladungen auf den Segmentkontakten, die dem die HW enthaltenen Segments benachbart sind, kann zusätzlich der innere Segmentwinkel Φ_i bestimmt werden. Der Algorithmus berechnet:

1. Für Segmente, die selbst nicht getroffen wurden, das absolute Maximum des Ladungssignals $|q|_{\max}$. Ein typisches Beispiel ist in Abbildung 2.8 gezeigt. Es handelt sich um ein **O**ne **S**egment **E**vent (OSE), d.h. nur eines der sechs Segmente des MINIBALL Detektors wurde getroffen. In diesem Fall wurde Segment Nummer 2 getroffen und auf den nichtgetroffenen Nachbarn 1 und 3 wird für einen Zeitraum von ca. 250 ns eine induzierte Ladung gemessen. Die Höhen dieser Signale werden bestimmt, indem die betragsmäßig größte Differenz zwischen ermitteltem Baselinewert und Pulswert bestimmt wird. Diese Berechnung wird für alle Segmente ausgeführt, in denen eine Energie deponiert wurde, die kleiner ist, als eine einstellbare Schwelle.

2. Für Segmente, die selbst getroffen wurden (d.h. es gab eine Wechselwirkung in diesem Segment und es wurde Energie deponiert) wird zuerst durch Subtraktion eines (berechneten) linear ansteigenden Signals der induzierte Teil des Segmentsignals gewonnen. Als Amplitude des berechneten Signals wird der von der XIA-Elektronik gelieferte Energiewert benutzt. Die Dauer des Coresignals definiert die Länge des Anstiegs und wird, zusammen mit der Energie als Pulshöheninformation, zur Berechnung der Steigung benutzt. Für den induzierten Teil des Segmentsignals wird ebenfalls der Wert des absoluten Maximums bestimmt. Diese Näherung funktioniert allerdings nur innerhalb der ersten Hälfte des Detektorpulses zufriedenstellend. Danach beginnt der Beitrag des induzierten Signals abzunehmen und verliert dadurch seinen Einfluß auf das Segmentsignal. In Abbildung 2.9 erkennt man, dass das Signal in Segment 4 wegen des negativen Beitrags der induzierten Ladung, die durch die im benachbarten Segment 3, in dem die HW stattgefunden hat, verursacht wird, erst später anzusteigen beginnt. Für das nichtgetroffene Segment 2 wird die maximale Ladung wie unter 1 beschrieben bestimmt. Solche Ereignisse werden als **N**eighbouring **S**egment **E**vents (NSE) bezeichnet. In diesem Beispiel handelt es sich um zwei benachbarte Segmente, weshalb dieser Typ kurz als 2 NSE bezeichnet wird.

   In der vorliegenden Arbeit wurden diese Analyseschritte in der XIA Elektronik implementiert und können damit in Echtzeit ausgeführt werden.

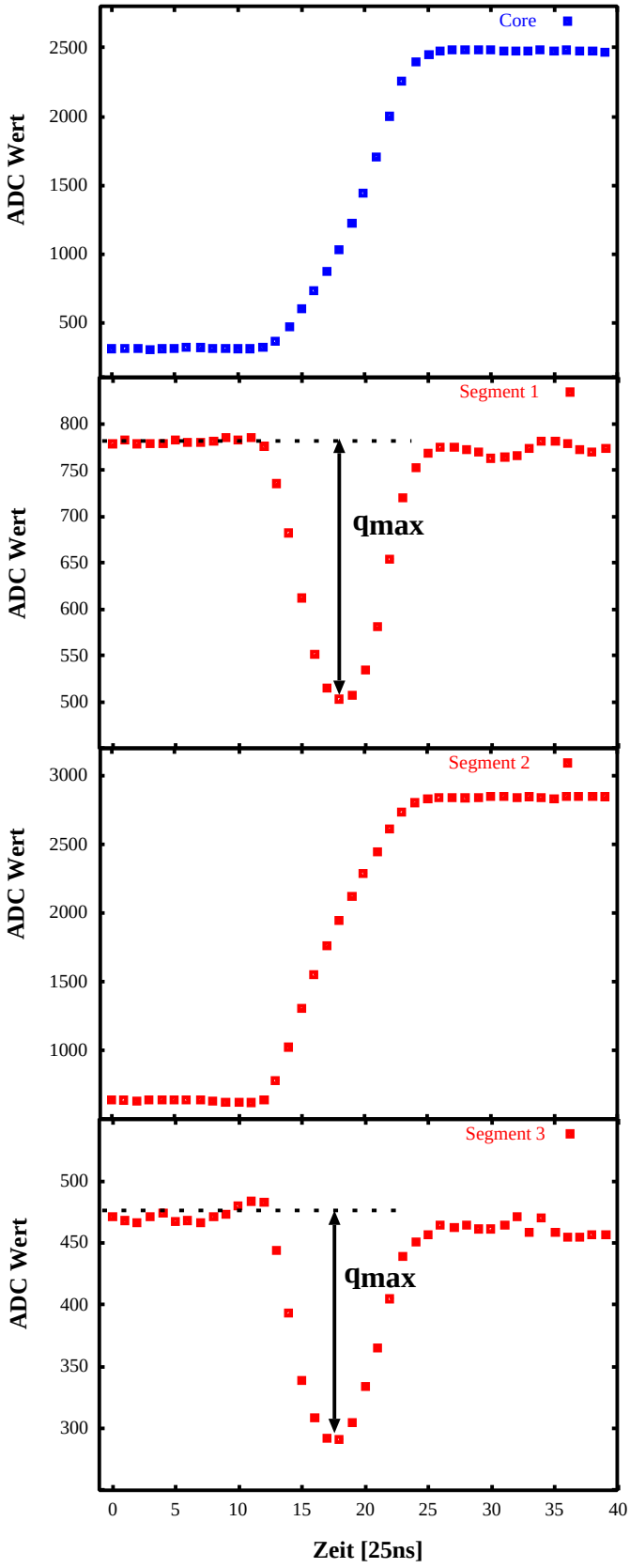

Abbildung 2.8: Ein One Segment Event (OSE). Nur ein Segment, Nummer 2, wurde von einem γ-Quant getroffen. Die Höhe der induzierten Ladung in den Segmenten 1 und 3 liegt etwa eine Größenordnung unter der eingestrahlten Energie in Segment 2, was aus der Stärke des weighting Feldes der benachbarten Segmenten resultiert.

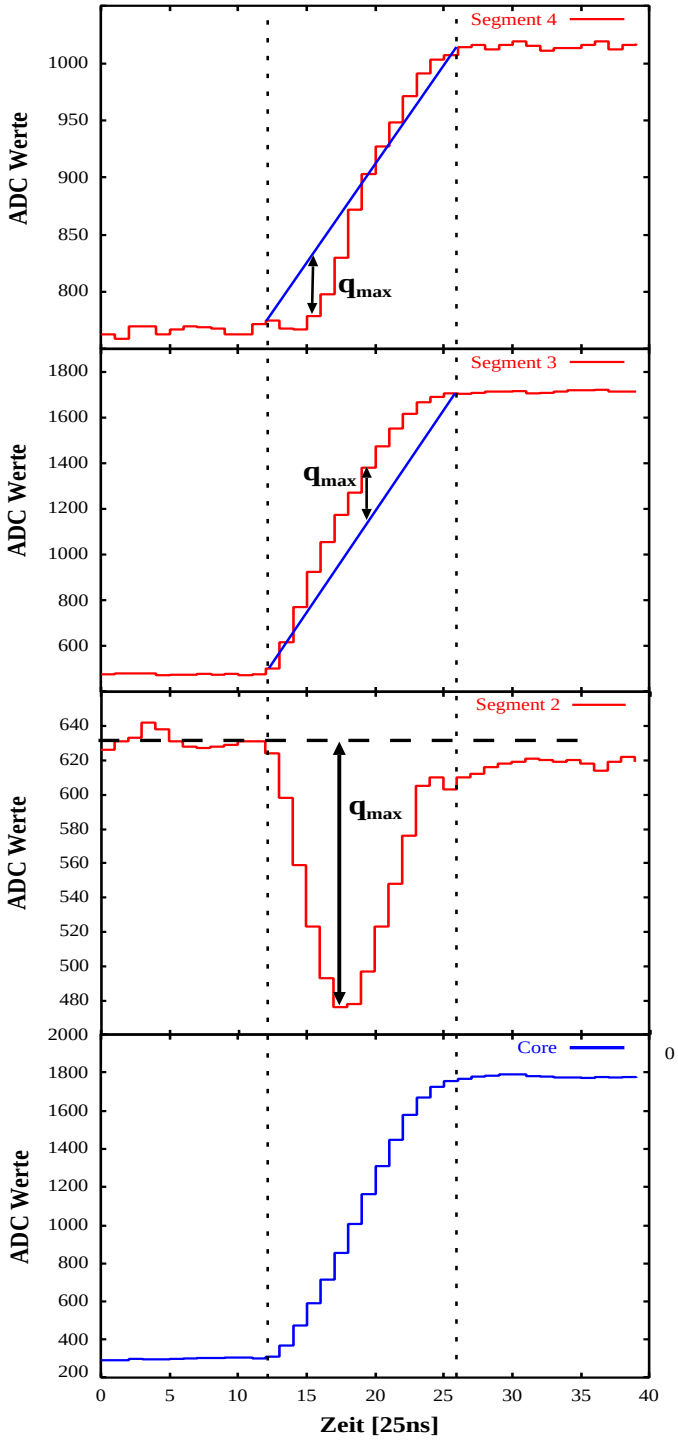

Abbildung 2.9: Ein Neighbouring Segment Event (NSE). In diesem Falle wurde Energie in den benachbarten Segmenten 3 und 4 deponiert. Für diese beiden Segmente wird die induzierte Ladung nach Methode 2 berechnet, für Segment Nummer 2 nach Methode 1. In Segment Nummer wird 3 die größte Energie gemessen. Infolgedessen wird die Winkelinformation durch den Host Computer aus den induzierten Ladungen der Segmente 2 und 4 bestimmt. In Segment 4 ist deutlich der verzögerte Anstieg des Signals zu erkennen. Ursache für die auf Kontakt 4 induzierte Ladung, ist die im benachbarten Segment 3 deponierte Ladung.

3. Für dasjenige Segment, indem die größte Ladung deponiert wurde, wird das Verhältnis der induzierten Ladung der direkt benachbarten Segmente bestimmt und davon der Logarithmus gebildet. Dieser Schritt muss, da für einen MINIBALL-Detektor (je sieben Kanäle) zwei DGF-4C Karten (vier Kanäle mit je einem Core- und drei Segmentsignalen) benötigt werden und keine Möglichkeit besteht, Information zwischen den Modulen auszutauschen, von einem an die Datenaufnahme angeschlossenen Computer ausgeführt werden. Der Wert von $\log(\frac{|q-|_{\max}}{|q+|_{\max}})$ ist ein Maß für den Winkel Φ_i im Inneren des Segments mit der maximalen Ladungsdeposition. Dabei bezeichnet man mit $-$ das Segment, welches den niedrigeren Index hat und mit $+$ das Segment, welches den höheren Index hat. In Abbildung 2.9 ist Segment Nummer 4 das Segment mit dem höheren Index und Segment Nummer 2 das Segment mit dem niedrigeren Index.

Kapitel 3

Signalverarbeitung

> 'Kids, you tried your best and you failed miserably. The lesson is, never try again.'
> Homer J. Simpson

Dieses Kapitel gibt eine Einführung in die Begriffe und Verfahren, die bei der Verarbeitung von Germaniumdetektorsignalen auftreten. Eine ausführliche Behandlung findet sich in [9]. Grundlage der digitalen Signalverarbietung ist die Diskretisierung und Quantisierung analoger Signale durch Analog-Digital-Wandler (ADC). Im Idealfall handelt es sich dabei um die Abtastung eines kontinuierlichen Signals zu äquidistanten Zeitpunkten. In der Realität wird die Präzision durch die endliche Messdauer eingeschränkt. Desweiteren muss darauf geachtet werden, dass Frequenzen oberhalb der halben Abtast-/Samplingfrequenz aus dem Signal entfernt werden, da diese ansonsten in das betrachtete Frequenzband gespiegelt werden, was zu Verfälschungen des Signals führt (Aliasing, Abtasttheorem, Nyquistfrequenz). In Kapitel 4 wird das Modul DGF-4C beschrieben, mit dem es erstmals möglich ist, die Signale eines Germaniumdetektors mittels digitaler Bausteine (FPGA, DSP) zu bearbeiten. Zur Energiemessung wird ein digitaler Filter benutzt, dessen ballistisches Defizit mittels Pulsformanalyse korrigiert wird. Um einen maximalen Durchsatz zu erreichen, wird der Rechenaufwand auf DSP und FPGA verteilt, wodurch sich die effektive Totzeit minimieren läßt.

3.1 Finite Impulse Response (FIR) Filter

Die Signalverarbeitung unterscheidet zwischen zwei Arten von Filtern: Finite und Infinite Impulse Response Filtern (FIR und IIR) [21]. Der Unterschied besteht darin, dass bei IIR Filtern der Ausgang auf den Eingang rückgekoppelt wird, wohingegen FIR-Filter nur auf vorangegangenen oder aktuellen Werten operieren. Gibt es keine Eingabe mehr, so endet die Ausgabe des FIR Filters nach endlicher und wohldefinierter Zeit (Filterlänge). Die Struktur eines FIR Filters ist in Abbildung 3.1 gezeigt. Das aktuelle Sample $x(n)$ wird mit dem zugehörigen Koeffizienten $c(1)$ multipliziert und auf ein Verzögerungselement (Delay) gegeben. Verzögerungselemente werden in der digitalen Signalverarbeiten mit z^{-1} bezeichnet. Aus diesem Verzögerungselement wird der vorangegangene Wert $x(n-1)$ ausgelesen und mit seinem Koeffizienten $c(2)$ multipliziert. Als *Filtertap* bezeichnet man die Einheit

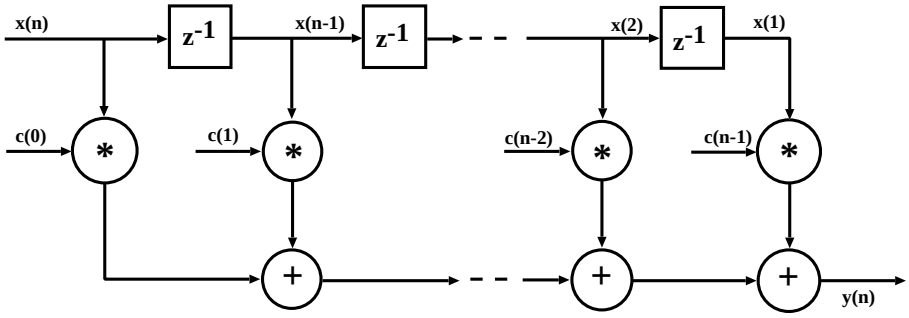

Abbildung 3.1: Schema eines Finite Inpulse Response (FIR) Filters. Der Ausgang $y(n)$ ist gegeben durch $y(n) = x(n) \cdot c(0) + x(n-1) \cdot c(1) + ... + x(1) \cdot c(n-1)$. Die gebräuchliche Schreibweise eines Delay-Elements ist z^{-1}.

bestehend aus einem Delay-Element plus Multiplikation mit dem Koeffizienten und abschließender Summation des Ergebnisses. Die Summe aller Produkte stellt den Filter dar. Die Länge eines Filters wird in der Einheit Taps gemessen. Die Übertragungseigenschaften eines FIR Filters sind bedingt durch die Anzahl der Taps und die Wahl der Filterkoeffizienten. Mathematisch stellt sich ein FIR Filter der Länge N als Produkt eines Datenvektors $\vec{x}$ mit dem Koeffizientenvektor $\vec{c}$ dar:

$$y(n) = \vec{c} \cdot \vec{x} = \sum_{k=0}^{N-1} c(k)\, x(n-k) = c(k) \; * \; x(n) \tag{3.1}$$

Der letzte Ausdruck in Gleichung 3.1 ist eine verkürzte Schreibweise der (diskreten) Faltung, die durch diese Gleichung definiert ist. Der Ausgabewert y zum Zeitpunkt n ergibts sich also durch eine Faltung der Filterkoeffizienten mit den (verzögerten) Eingabewerten. Die *Impulsantwort* eines Filter ist definiert als die Ausgabe des Filter nach Eingabe eines einzelnen Samples mit Wert 1. Nach Gleichung 3.1 ergeben sich für diesen Fall ($x(0) = 1$, sonst $x = 0$) die Filterkoeffizienten selbst. Die Impulsantwort entspricht exakt den Filterkoeffizienten.

In diesem Zusammenhang soll noch die *Stufenantwort* definiert werden. Sie ist die Antwort eines System auf die Eingabe eines Θ(n)-Pulses, d.h. für $n < 0$ ist die Eingabe Null, sonst gleich Eins. Die Bedeutung der FIR-Filter für die Verarbeitung von Germaniumdetektorsignalen ist schon lange bekannt, da sie durch ihre definierte Länge eine höheren Durchsatz erlauben. Durch geeignete Wahl der Koeffizienten läßt sich zudem die Ausgangspulsform so manipulieren, dass eine ausgezeichnete Energieauflösung erreicht wird. Einige rekursive Algorithmen (also die Differenz zwischen dem n-ten und $(n-1)$-ten Ausdruck), um verschiedene Pulsformen aus einem stufenförmigen Signal zu erhalten, sind in [22] aufgeführt. Wie dort erwähnt, ist für Detektorsignale mit einer langen Abfallszeit eine Stufenfunktionen eine gute erste Näherung.

3.2 Ballistic deficit

Das ballistische Defizit einer Messung ist nach [23] definiert als der Verlust an Pulshöhe am Ausgang eines pulsformenden Netzwerks (Vorverstärker, Shaper), der auftritt, wenn man

einen Puls mit von Null verschiedener Anstiegszeit t_{rise} auf den Eingang legt. Zusätzlich erreicht der Puls am Ausgang sein Maximum mit einer Verzögerung $\Delta T = \tau_{\text{bd}}$ gegenüber dem idealen Rechteckpuls. Die Höhe dieses Maximums wird – durch die üblicherweise verwendeten Peak Sensing ADCs – also zu verschiedenen Zeitpunkten gemessen. Wird mit V die Pulshöhe am Ausgang des Pulsformers und mit $T_0 = \tau_{peak}$ die Anstiegzeit des Shaper-Signales für Rechtecksignale bezeichnet, so ergibt sich der Verlust ΔV an Höhe zu:

$$\Delta V = V_0(\tau_{\text{peak}}) - V^{\text{real}}(\tau_{\text{peak}} + \tau_{\text{bd}}) \quad . \tag{3.2}$$

$V^{\text{real}}(t)$ – und damit das ballistische Defizit – ergibt sich allgemein aus der Faltung des Strompulses $i(t)$ mit der Impulsantwort $h'(t)$ des pulsformenden Netzwerkes:

$$V^{\text{real}}(t) = \int_0^t i(t)\, h'(t-\tau) d\tau. \tag{3.3}$$

Es soll betont werden, dass es die *Variation* in der Anstiegszeit und damit in τ_{bd} ist, die zur Verbreiterung der Spektralinien führt. Qualitativ gilt, dass das ballistische Defizit groß ist, wenn das Verhältnis $t_{\text{rise}}/\tau_{\text{peak}}$ groß ist, wobei mit τ_{peak} die Anstiegzeit des Shapers bezeichnet wird. Da t_{rise} eine nicht beeinflussbare Eigenschaft des Detektors ist, kann durch erhöhen von τ_{peak} der Effekt des ballistischen Defizits – unter Verlust an Raten-Performance – reduziert werden. In der Literatur [7, 24, 25] sind einige Methoden und Schaltungen aufgeführt um das ballistische Defizit eines pulsformenden Netzwerkes nachträglich zu korrigieren. Ein Vergleich zweier Methoden findet sich in [26]. Beide Methoden beruhen auf der Beziehung

$$\frac{\Delta V}{V_0} = \left(\frac{\Delta T}{T_0}\right)^2, \tag{3.4}$$

zwischen dem relativen ballistischen Defizit $\frac{\Delta V}{V_0}$ und dem Quadrat der relativen Verzögerung des Peakwertes. Diese Beziehung gilt unter den Annahmen, dass alle Detektorpulse die gleiche Form nur auf unterschiedlichen Zeitskalen haben und dass der Shaperpuls im Maximum durch eine Parabel angenähert werden kann. Während die Goulding/Landis-Methode direkt die Verzögerung mißt und einen Korrekturwert nach Formel 3.4 addiert, arbeitet die Hinshaw-Methode mit zwei Kanälen unterschiedlicher Anstiegszeit, wodurch die wahre Amplituden mit Hilfe der Kenntnis der Differenz der Amplituden in den beiden Kanälen bestimmt werden kann.

Ballistic Deficit Correction durch Pulsformanalyse

Die Korrektur des ballistischen Defizits der Firma XIA ist für Resitiv-Feedback Vorverstäker, die auch für die MINIBALL Detektoren verwendet werden, entwickelt worden. Da die Signale des Germaniumdetektors direkt nach dem Vorverstärker digitalisiert werden, ist anzunehmen, dass die einzige Pulsformung die des Integrationsglieds ist. Die Impulsantwort $h'(t)$ des Vorverstärkers ist

$$h'(t-t') = \frac{1}{\tau} e^{-\frac{t-t'}{\tau}} \quad . \tag{3.5}$$

Durch Analyse der digitalisierten Pulsformen, läßt sich das ballistische Defizit auf eine elegante Weise berechnen und somit korrigieren. Fließt über den Zeitraum der Ladungssammlung t_{cc} der Strom i(t), dann erhält man durch eine Integration die Ladung $q = \int_0^{t_{cc}} i(t')dt'$.

Das Ladungssignal des Vorverstärkers ergibt sich durch Faltung der Impulsantwort $h'(t)$ mit dem Strom $i(t)$:

$$q(t) = \int_0^{t_{cc}} i(t') \cdot h'(t-t')dt' = \int_0^{t_{cc}} i(t') \cdot \frac{1}{\tau} e^{-\frac{t-t'}{\tau}} dt' = e^{-\frac{t}{\tau}} \cdot \int_0^{t_{cc}} i(t') \cdot \frac{1}{\tau} e^{\frac{t'}{\tau}} dt' \tag{3.6}$$

Der Verlust ΔV an Höhe im Vergleich zur idealen Integration berechnet sich, wenn man die Normierung aus Gleichung 3.5 berücksichtigt ($h'(t) = \frac{1}{\tau} \cdot \Theta(t) \cdot \Theta(\tau - t)$), zu:

$$\begin{aligned} \Delta V &= q^{ideal} - q^{VV} = \int_0^{t_{cc}} \frac{1}{\tau} \cdot i(t')dt' - e^{-\frac{t}{\tau}} \cdot \int_0^{t_{cc}} i(t') \cdot \frac{1}{\tau} e^{\frac{t'}{\tau}} dt' & (3.7) \\ &= \int_0^{t_{cc}} \frac{1}{\tau} \cdot i(t') \left[1 - e^{-\frac{t-t'}{\tau}}\right] dt' & (3.8) \end{aligned}$$

Da $t_{cc} \ll \tau$ kann man dieses Integral folgendermaßen annähern

$$\begin{aligned} \Delta V &\approx -\int_0^{t_{cc}} \frac{1}{\tau} \cdot i(t') \cdot \frac{t-t'}{\tau} dt' = -\int_0^{t_{cc}} \frac{1}{\tau} \cdot i(t') \cdot \frac{t}{\tau} dt' + \int_0^{t_{cc}} \frac{1}{\tau} \cdot i(t') \cdot \frac{t'}{\tau} dt' & (3.9) \\ &= -\frac{t}{\tau} \cdot \frac{q(t_{cc})}{\tau} + \left[\frac{q(t)}{\tau} \cdot \frac{t}{\tau}\right]_0^{t_{cc}} - \int_0^{t_{cc}} \frac{1}{\tau^2} \cdot q(t')dt' & (3.10) \\ \Delta V &= -\frac{1}{\tau} \int_0^{t_{cc}} \frac{q(t')}{\tau} dt' & (3.11) \end{aligned}$$

Damit ist gezeigt, dass das ballistische Defizit durch Integration des Ladungspulses über den Zeitraum der Ladungssammlung bestimmt werden kann. Durch Addition des Verlustes zum Energiefilterwert kann somit das ballistische Defizit vermindert werden. Der zusätzliche Faktor $\frac{1}{\tau}$ kommt durch die gewählte Normierung zustande q $\rightarrow \frac{q}{\tau}$. Es soll betont werden, dass diese Herleitung *keine* Annahmen über die Form des Ladungs- bzw. Strompulses benötigt hat und somit allgemeingültig ist.

3.3 Pileup

Die Signale eines Germaniumdetektors ereignen sich – aufgrund ihres statistischen (radioaktiven) Ursprungs – zufällig und sind deshalb durch einen beliebigen zeitlichen Abstand gekenzeichnet. Die Beeinträchtigung eines der Messung der Energie eines Pulses durch vorangegangene/nachfolgende Pulse bezeichnet man als *pileup*. Dabei unterscheidet man tail und peak pileup. Mit tail pileup werden alle diejenigen Ereignisse bezeichnet, bei denen der Puls auf dem tail des vorangehenden Pulses beginnt. Die Spektren zeigen dann Peaks die zu niedrigen bzw. hohen Energie hin verbreitert sind. Entstehen zwei Pulse innerhalb so kurzer Zeit, dass sie nicht mehr als einzelne Pulse erkannt werden können, so nennt man dies pulse pilup. Infolgedesse werden die beiden Pulse nicht ihren zugehörenden Peaks zugeordnet, sondern einem Summenpeak, womit das Spektrum verfälscht wird.

Mithilfe analoger Standardelektronik implementiert sich eine 'pileup rejection', durch Aufspaltung des Signals in einen schnellen und einen langsamen Zweig. Der langsame Zweig wird zur Energiemessung benutzt, der schnelle Zweig zählt die Anzahl der Impulse. Gab es ein Ereignis, so wird für eine bestimmte Zeit lang der schnelle Zweig auf weitere Ereignisse

hin überwacht. Falls es innerhalb dieses Zeitinterwalls zu weiteren Ereignissen kommt, so wird das ursprüngliche Ereignis verworfen. Das korrekte Funktionieren dieses Verfahrens hängt natürlich von der Zeitauflösung des schnellen Zweigs ab, sowie der Pulshöhenschranke ab der Pulse als Ereignisse erkannt werden, die ihrerseits durch das Rauschen im schnellen Zweig nach unten hin begrenzt wird. Der Zeitraum der Überwachung hängt ab von der Zeit T_1 bis aus dem Signal des langsamen Zweigs der Energiewert extrahiert wird und der Zeit T_2 bis das Signal vollständig abgeklungen ist. Um eine unverfälschte Energiemessung zu erhalten wird im allgemeinen zu einem vorangehenden Event ein Abstand von $T_1 + T_2$ und zu einem nachfolgenden Event der Abstand von T_1 eingehalten. Wichtig ist die Erkenntnis, dass das pile up Verhalten entscheidend von der Form des Pulses abhängt und somit eine der Anwendung gerecht werdende Pulsform ausgewählt werde sollte.

3.4 Totzeit

Als Totzeit wird die Zeit nach der Annahme eines Ereignisses durch das Datenaufnahmesystem (Detektor, Vorverstäker, Elektronik, Auslese, Computer) bezeichnet, in der keine weiteren Ereignisse angenommen werden können. Zwei Modelle werden üblicherweise aufgeführt, wenn das Totzeitverhalten von Datenaufnahmesystemen beschrieben werden soll: paralysierbar und nicht-paralysierbar. Reale Systeme sind im Allgemeinen Mischformen der beiden Grundtypen. Der Einfachheit halber sei im folgenden eine feste Totzeit mit der Verarbeitung eines akzeptierten Events verbunden. Findet nun innerhalb dieser Totzeit ein weiteres Ereignis statt, so führt dies im paralisierbaren Fall zur Verlängerung der gesamten Totzeit um ein weiteres Totzeitintervall beginnend ab dem Zeitpunkt des neuen Events. Im nicht paralysierbaren Fall wird das neue Event zwar ebenfalls nicht verarbeitet, allerdings wird die gesamte Totzeit nicht verlängert. Bezeichnet n die wahren Ereignisrate, m die verarbeitete Ereignisrate und τ die Totzeit des untersuchten Systems, so ergeben sich die folgende Zusammenhänge [4]:

$$m = \frac{n}{1 + n\,\tau} \quad \text{im nicht-paralysierbaren Fall} \tag{3.12}$$

und

$$m = n\,e^{-n\,\tau} \quad \text{im paralysierbaren Fall.} \tag{3.13}$$

3.5 Rauschen

Für das Verständnis der Funktionsweise der Energiemessung durch FIR-Filter ist es nützlich, einige Grundlagen der Rauschanalyse und der Rauschreduktion durch pulsformende Netzwerke aufzuführen. Eine ausführliche Darstellung ist bei [27] zu finden.

Zwei Arten von Rauschquellen werden hierbei unterschieden: *step noise* (auch *parallel noise*) und *delta noise* (auch *impulse noise* oder *series noise*). Alle anderen Rauscharten werden vernachlässig, da für sie keine einfache Beschreibung existert.

Ersterer liegt im nichtidealen Verhalten von Detektor und Vorverstärker begründet und kann als eine parallel zum Detektor geschaltete (Rausch-) Stromquelle aufgefasst werden (Dem perfekten Detektorsignal wird das Rauschsignal überlagert). Sie repräsentiert

alle Rauschquellen im Eingangskreis des Vorverstärkers. Die im Eingangskreis fließende Ladung wird in der Eingangskapazität (inkl. Detektor) integriert und erscheint dann als Spannungsstufe am Eingang des Vorverstärker.

Die zweite Art Rauschen wird durch den Verstärkungsprozeß verursacht. Eine durch den Vorverstärker fließende (Rausch-) Ladung verursacht einen kurzen (Delta-) Strompuls am Ausgang des Vorverstärkers, was durch eine Spannungsquelle, welche kurze Deltapulse liefert, in Serie mit dem Detektor (aber im Anschluß an die integrierende Kapazität) dargestellt werden kann.

Das verrauschte Signal wird direkt im Anschluß an den Vorverstärker durch den Pulsformer (Shaper) verarbeitet. Es soll angenommen werden, dass der Ausgang des Pulsformers zu einer festen Zeit T_{mess} gemessen wird. Ausserdem soll angenommen werden, dass jedes Noise Event die gleiche Menge an Ladung erzeugt. Um den Effekt des Pulsformers auf einen Eingangspuls bestimmen zu können, müssen die Stufenantwort $h(t)$ und die Impulsantwort $h'(t)$ (= Filterkoeffizienten) (auch *weighting function* oder *residual function*) bekannt sein. Unter der Annahme, dass Rauschstufen mit einer Rate von n_s pro Sekunde stattfinden, ergibt sich für normale Statistik, dass die mittlere Anzahl von Rauschstufen im Zeitintervall dt gleich $\overline{N} = n_s\, dt$ ist. Von Bedeutung ist nicht der konstante Offset, sondern die Fluktuation. Die mittlere quadratische Abweichung ergibt sich zu $\langle n^2 \rangle = n_s\, dt$. Somit ist die mittlere quadratische Abweichung des *gemessenen* Signals als Folge von step noise im Zeitintervall dt zur Zeit t bevor T_{mess} gegeben durch $\langle n^2 \rangle \cdot h^2(t) = n_s\, h^2(t)\, dt$. Der Effekt aller vorangegangener step noise Ereignisse ergibt durch Integration. Zur Definition des *Noise Index* werden nur Eigenschaften des Pulsformers ($h(t)$ bzw. $h'(t)$) verwendet. Nach Normierung ergibt sich (S: Amplitude des Shapers):

$$\langle N_s^2 \rangle = \frac{1}{S} \int_0^\infty \left(h(t)\right)^2 dt \qquad \text{Step Noise Index} \tag{3.14}$$

sowie durch ähnliche Überlegungen

$$\langle N_\Delta^2 \rangle = \frac{1}{S} \int_0^\infty \left(h'(t)\right)^2 dt \qquad \text{Delta Noise Index .} \tag{3.15}$$

Da nur die Eigenschaften des Pulsformers eingehen, sind diese Gleichungen zum Vergleich der verschiedenen Shaper geeignet. Allerdings wird keine Aussage über absolute SNR[1]-Werte gemacht. Für einen Trapezfilter mit einer Anstiegszeit T_1, einer "flat top" Zeit von T_{F}, Abfallzeit von T_2 und einer Amplitude von Eins erhält man:

$$\langle N_s^2 \rangle = \int_0^{T_2} \left(\frac{t}{T_2}\right)^2 dt + \int_0^{T_{\text{F}}} (1)^2\, dt + \int_0^{T_1} \left(\frac{t}{T_1}\right)^2 dt = T_{\text{F}} + \frac{\text{T}_1 + \text{T}_2}{3} \quad , \tag{3.16}$$

und

$$\langle N_\Delta^2 \rangle = \int_0^{T_2} \left(\frac{1}{T_2}\right)^2 dt + \int_0^{T_1} \left(\frac{1}{T_1}\right)^2 dt = \frac{1}{T_1} + \frac{1}{T_2} \quad , \tag{3.17}$$

woraus für eine symmetrische Dreiecksform ($T_1 = T_2 = T_{\text{P}}$ und $T_{\text{F}} = 0$) $\langle N_s^2 \rangle = 0.667\, T_{\text{P}}$ und $\langle N_\Delta^2 \rangle = \frac{2}{T_{\text{P}}}$ folgt. Folgende Schlüsse lassen sich aus den diesen Ergebnissen ziehen:

[1] Signal to Noise Ratio

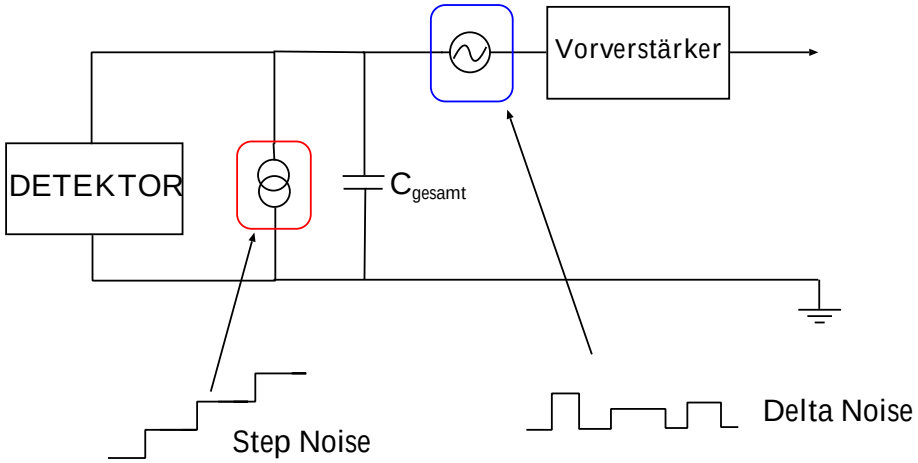

Abbildung 3.2: Ersatzschaltbild zur Erläuterung der Rauschquellen.

- Step Noise ist proportional zur Länge des Filters
- Delta Noise ist umgekehrt proportional zur Filterlänge
- Bei vorgegebener Filterlänge erreichen symmetrische Pulsformen die besten Ergebnisse

Als Gesamtrauschindex wird für symmetrische Pulsformen gerne die Größe $\sqrt{\langle N_s^2 \rangle \langle N_\Delta^2 \rangle}$ verwendet. Es ist bekannt, dass das optimale Signal zu Rausch Verhältnis (SNR) für eine Pulsform erreicht wird, die einer an der $t = 0$-Achse gespiegelten Exponentialfunktion $e^{-\frac{t}{\tau}}$ entspricht. Diese Pulsform erreicht im obigen Noise Index einen Wert von 1, den Optimalwert. Für den oben erwähnten Dreiecksfilter ergibt sich ein Wert von 1.16. Zwar verschlechtert sich das SNR ein wenig gegenüber der Idealform, allerdings endet der Dreicksfilter nach zweimal T_P, wohingegen die Exponentialfunktion nie Null wird. Eine gaussförmige Pulsform erreicht einen Wert von 1.3 und ist damit einem Dreiecksfilter unterlegen.

3.6 Digitale Signal Prozessoren (DSP)

In jedem PC findet sich heutzutage ein leistungsfähiger Mikroprozessor. Diese Prozessoren (z.B. die Intel Pentium Reihe und AMD Athlon) sind wahre Allroundtalente. Ob Text-, Bild- oder Tonverarbeitung, es gibt keine Anwendung die nicht mit ihnen implementiert werden kann. Den Preis den man für diese Vielfalt bezahlen muss ist hoch: Der Mikrochip muss eine sehr flexible und aufwändige Architektur haben. Selbst sogenannte **R**educed **I**nstruction **S**et **C**omputer (RISC) haben einen Befehlsatz von einigen hundert Instruktionen. Um den Anforderungen moderner Multimedia-Anwendungen gerecht zu werden, ist dies immer noch nicht ausreichend, weshalb aktuelle und zukünftige Prozessoren mit spezialisierten Rechenwerken (und speziellen Instruktionen) aufwarten können, die speziell für Signalverarbeitung optimiert sind (Stichworte: SIMD, VLIW, speziell Intels MMX [28] bzw. ISSE [29], AMDs 3DNow! [30] und Motorolas AltiVec Technologie [31]). Ganz im Gegensatz dazu sind die **D**igitalen **S**ignal **P**rozessoren (DSP) speziell für die Verarbeitung

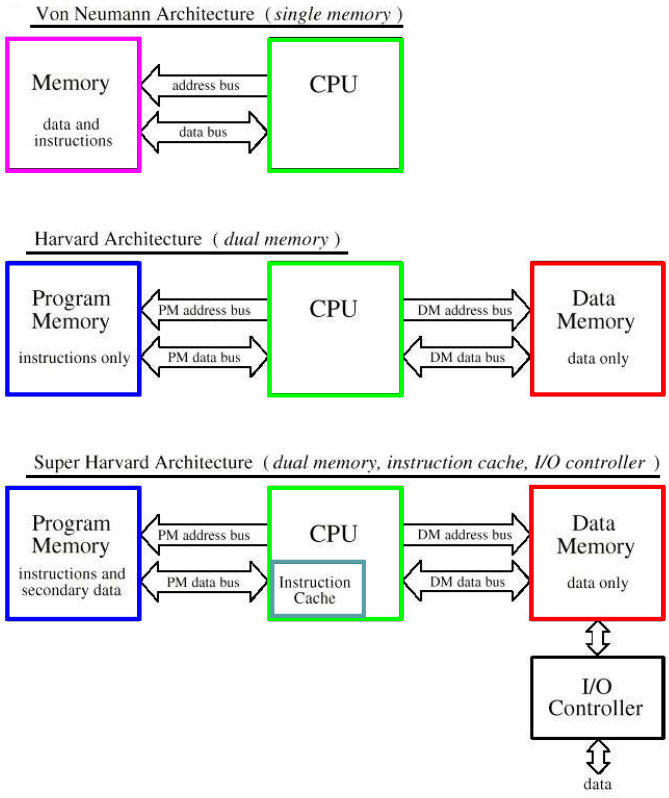

Abbildung 3.3: Vergleich verschiedener Prozessor Architekturen [32]

von Signalen entwickelt worden und weisen deshalb einige architekturale Merkmale auf, die sie von normalen Mikroprozessoren (GPP: **G**eneral **P**urpose **P**rocessor) unterscheiden.

3.6.1 Allgemeiner Aufbau und Eigenschaften

Digitale Signalverarbeitung führt immer auf eine kennzeichnende Operation: Multiply and Accumulate, kurz MAC genannt. Zur Berechnung eines einzelnen Taps eines FIR-Filters müssen der neue Samplewert und entsprechende Koeffizient miteinender multipliziert werden und das Ergebnis zu der Summe der vorangegangenen Filtertaps addiert werden. Innerhalb eines Taktzyklus müssen zwei Werte aus dem Speicher gelesen werden. Dies ist für einen GPP (aufgrund der begrenzten Speicherbandbreite) nicht leicht umzusetzen, da dieser eine Von Neumann Architektur (John von Neumann, 1903-1957) hat, bei der es nur einen gemeinsamen Daten- und Programmspeicher gibt (allerdings haben einige GPP getrennte Daten- und Programmcaches). Deshalb implementieren DSPs eine modifizierte Harvard Architektur (Howard Aiken, 1900-1973), bei der es getrennte Speicher für Daten (DM: Data Memory) und Programmierung (PM: Program Memory) gibt. Damit hat sich die Speicherbandbreite verdoppelt, innerhalb eines Taktes kann nun sowohl auf den Programmspeicher als auch auf den Datenspeicher zugegriffen werden. Die Modifizierung besteht darin, dass entgegen der ursprünglichen Definition der reinen Harvard Architektur, innerhalb des Programmspeichers auch Daten gelagert werden können. Abbildung 3.3 stellt

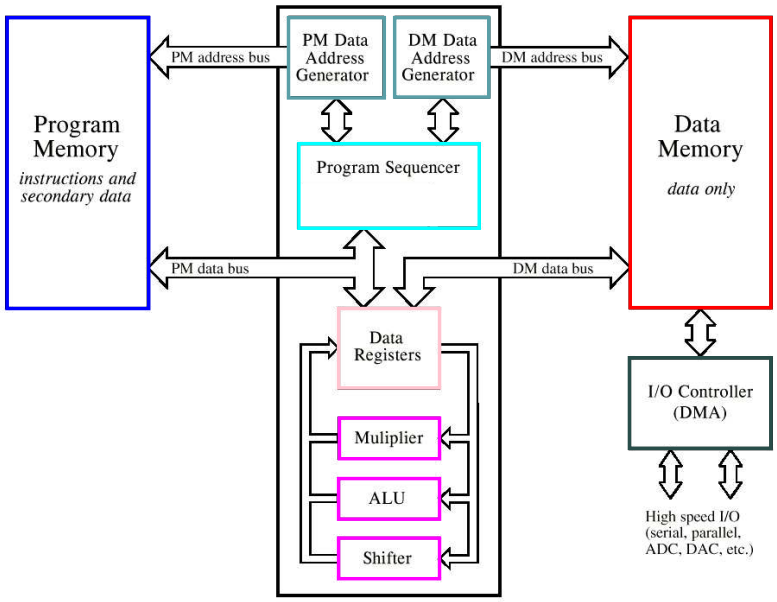

Abbildung 3.4: Schematischer Aufbau eines digitalen Signal Prozessors [32]

die verschiedenen Architekturen gegenüber. Einen Schritt weiter geht die Firma Analog Devices mit ihrer **S**uper **H**arvard **Arc**hitektur (SHARC), indem sie einen Cachespeicher für Instruktionen einführt, womit die Ausführung von Schleifen beschleunigt wird, da nun die ganze Speicherbandbreite für Daten (DM) und Koeffizienten (PM) zur Verfügung steht. Spezielle Hardware sorgt ausserdem für schnelle Schleifenbehandlung, d.h. initieren und testen der Abbruchsbedingung erfolgt nebenbei. Diese Eigenschaft wird im DSP Jargon als zero overhead looping bezeichnet. Alle DSPs haben ausserordentliche I/O-Fähigkeiten, was kein Wunder ist, wird doch von DSP Anwendungen Echtzeitverhalten gefordert, da man es mit einem durchgehenden Strom (streaming application) an Daten zu tun hat. Der Speicherzugriff erfolgt dabei ohne Belastung des Prozessors über **D**irect **M**emory **A**ccess (DMA) des eigenständigen I/O-Controllers. Eine weitere Anforderung an DSPs ist die Vorhersagbarkeit ihres Verhaltens. GPPs haben meistens einen Cachespeicher auf den sie schnellen Zugriff haben. Cache misses (die angefragten Daten stehen nicht im Cache) führen dazu, dass ein Zugriff auf den Hauptspeicher ausgeführt werden muss, der einige Taktzyklen benötigt. Somit ist die Geschwindigkeit der Ausführung vom Inhalt des Cachespeichers und somit von der Vorgeschichte abhängig. Allerdings haben GPPs Taktfrequenzen im GHz Bereich erreicht, wohingegen klassische DSP-Architekturen eine Größenordnung darunter liegen. DSP-Echtzeitanwendungen haben allerdings ein festes Zeitfester, dass nicht überschritten werden darf. Aus diesem Grund sind Caches auf klassischen DSP-Architekturen unüblich, da das Verhalten durch die Programmiererung festgelegt sein soll und muss. Auf DSPs läuft im Gegensatz zu GPPs kein Betriebssystem, stattdessen wird ein festes Programm ausgeführt.

Abbildung 3.4 zeigt eine detailliertere Darstellung einer DSP-Architektur. Auf der linken und rechten Seite sind die Daten und Programmspeicher erkennbar, die beide über ihre eigenen Adress- und Datenbusse mit dem Prozessorkern verbunden sind. Beide Adres-

sleitungen werden über unabhängige und eigenständige **D**ata **A**ddress **G**enerators (DAG) angesteuert. In GPPs fällt diese Aufgabe dem Programm Sequenzer (auch Program Counter PC) zu (Adressierungsarten: PC relativ, absolut, Register direkt/indirekt). Durch die verteilte Speicherung wurde die Aufspaltung in zwei eigenständige Adressierungeinheiten notwendig. Ausserdem benutzen DSP-Anwendung häufig zirkulare Buffer[2]. Das korrekte Funktionieren wird (Aktuallisierung der Schreib- und Lesepointer) – nach Konfiguration durch den Programmierer – durch die DAG-Hardware sichergestellt. Desweiteren wird automatisch innerhalb eines Taktes die nächste Adresse durch Addition des *Adress Modifiers* generiert. An Rechenwerken finden sich eine Arithmetic/Logic Unit (ALU), ein Multiplier/Accumulator (MAC) und Shifter-Einheit. Allen Rechenwerken stehen zwei vollständige und unabhängige Registersätze zur Verfügung. Damit kann zur Interrupt[3] Behandlung durch einfaches umschalten stets auf "frische" Register zugegriffen werden, wohingegen GPPs den Inhalt ihrer Register "retten" und wiederherstellen müssen bevor sie zur Interruptbehandlung übergehen können. Dieses *fast context switching* prädestiniert DSPs für schnelle Interruptbehandlung. Alle Rechenwerke sind ausserdem in der Lage ihre Operationen innerhalb eines einzigen Taktes auszuführen, was den DSPs die Fähigkeit verleiht eine vollständige MAC-Operation als sog. Multifunction[4] in einem Taktzyklus auszuführen. Dies erlaubt die Berechnung eines vollständigen Filtertaps pro Taktzyklus.

Von äusserster Wichtigkeit bei der Verarbeitung von Signalen ist die numerische Präzision mit der die einzelnen Operationen ausgeführt werden. Deshalb werden DSPs unterteilt in 16 bzw. 32 Bit und fixed point bzw. floating Typen. Zudem gibt es MAC-Register mit erweiterter Bit-Breite, was ein vielfaches Überlaufen von Berechnungen erlaubt.

3.6.2 Der ADSP-2181

Auf dem CAMAC Modul DGF-4C befindet sich der ADSP-2181 Prozessor von Analog Devices. Er gehört zur ADSP-2100 Familie, deren Mitglieder die gleiche Basis Architektur haben. Somit sind die folgenden Erläuterungen auch für den, auf der Revision D des Moduls DGF-4C verwendeten, ADSP-2183 zutreffend. Der ADSP-2181 ist zuständig für die Konfiguration der Karte und die Steuerung der Datenaufnahme. Alle aufwändigeren Operation wie Pulsformanalysen und Energiefilterkorrekturen werden durch den DSP ausgeführt. Da dieser DSP auch zur Implementierung der eigenen Algorithmen benutzt wurde, wird sein Aufbau im folgenden beschrieben.

Der digitale Signalprozessor ADSP-2181 ist ein 16 Bit, fixed point DSP, d.h. Datenbusse und Datenregister sind 16 Bit breit und die Rechenwerke können direkt nur mit 16 Bit Daten arbeiten. Will man mit 32 Bit Zahlen rechnen, so müssen die oberen und unteren 16 Bit getrennt verarbeitet werden. Als Signalprozessor ist der ADSP-2181 mit den typischen Merkmalen eines DSPs ausgestattet. In einem einzigen Taktzyklus kann der ADSP-2181

- eine neue Programm Adresse generieren,

[2]Die Speicherelemente sind ringförmig organisiert, d.h. auf das letzte Element folgt wieder das erste Element des Puffers.

[3]Ein Interrupt ist ein asynchrones Ereignis, ausgelöst durch Hard- oder Software. Die Arbeit des Prozessors wird unterbrochen und eine spezielle Interrupt Routine ausgeführt.

[4]Eine einzige Instruktion setzt sich aus mehreren Instruktionen zusammen.

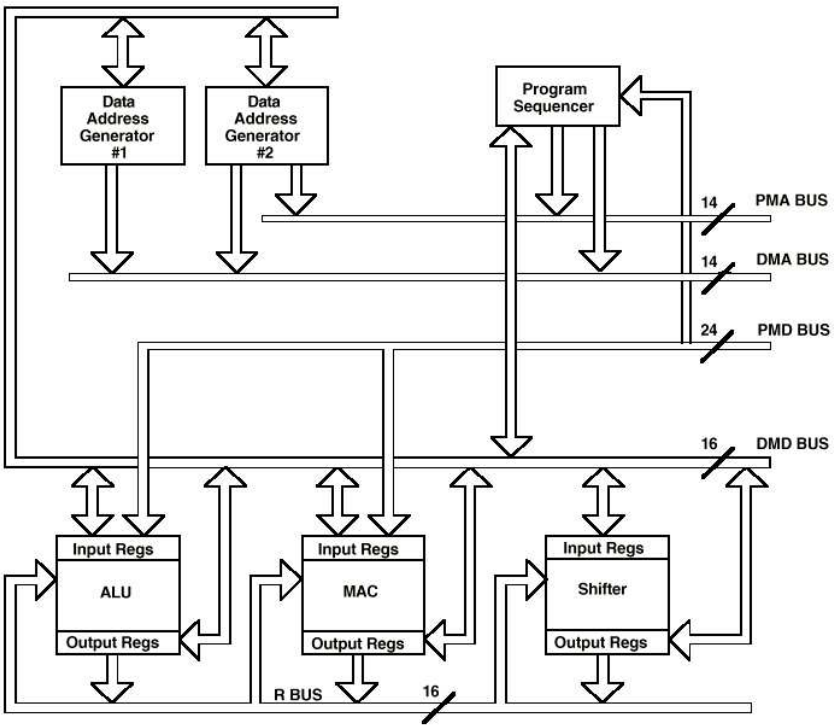

Abbildung 3.5: Vereinfachter Aufbau des ADSP-2181 DSPs [33].

- eine neue Instruktion laden (Instrukionen sind 24 Bit Worte (PM)),
- ein oder zwei Speicherzugriffe ausführen,
- ein oder zwei Adresspointer aktualisieren,
- eine Rechenoperation ausführen,
- Daten über den DMA Port empfangen/senden,
- Daten über den seriellen Port empfangen/senden und
- den Timer erniedrigen.

Er erreicht bei einem Takt von 20 MHz eine Zykluszeit von 25 ns, was zu einer Performance von 40 MIPS (Millionen Instruktionen[5] pro Sekunde) führt. Der DSP verfügt über eine ALU, eine MAC-Einheit und eine Shift-Einheit. Pro Takt/Instruktion kann nur eine dieser Einheiten aktiv sein. Die ALU arbeiten mit 16 Bit Daten und liefert 16 Bit Ergebnisse, wohingegen der Multiplizierer der MAC-Einheit ($2^{16} \cdot 2^{16} = 2^{32}$) 32 Bit breite Ergebnisse liefern kann. Zusätzlich verfügt der Addierer der MAC-Einheit über ein 8 Bit breites Überlaufregister, womit erst nach 256 Überläufen Daten verloren gehen. Sowohl ALU als auch MAC-Einheit verfügen über Feedback-Register. Ebenso kann das Ergebnis einer ALU/MAC/Shifter Operation direkt als Eingabe in den Recheneinheiten verwendet

[5]Dazu gehören auch die Multifunktionsinstruktionen.

werden. Da das Ergebnis nicht in den Speicher weggeschrieben wurde, kann somit direkt von Register zu Register gearbeitet werden. Dazu gehört die Eigenschaft, dass Register zu Beginn des Taktes gelesen werden und am Ende des Taktes beschrieben werden. Dadurch führen Operationen, die ein und dasselbe Register als Input und als Output verwenden zu korrekten Ergebnissen. Die Shift-Einheit kann ein 16 Bit Datenwort beliebig innerhalb des 32 Bit Ergebnisregisters plazieren. Dabei sind logische und arithmetische Schiebeoperation möglich. Zur Vergrößerung der numerischen Präzision unterstützt der Shifter das sog. Block-Floating-Point Format. Dabei wird ein Datensatz auf seinen Maximalwert hin untersucht und dieser anschliessend auf die maximale Anzahl an Nachkommastellen hin normalisiert und der Schiebewert als Exponent gespeichert. Der restliche Datensatz wird nun um die gleiche Anzahl von Bits nach links geschoben, alle Daten benutzen den gleichen Exponenten. Die Geschwindigkeit des On-Chip Speichers (der Programmspeicher ist dual-ported, d.h. er verfügt über zwei Daten- und Adressleitungen) reicht für 3 Zugriffe pro Takt aus: Zwei Operanden (DM und PM) sowie eine Instruktion (PM). Es stehen jeweils 16 k (16 bzw. 24 Bit) Worte an internem Speicher zur Verfügung. Der interne DMA Port ist asynchron und kann während der Ausführung von Instruktionen beschrieben werden. Beide DAGs und der Programmsequenzer erlauben Zugriffe auf internen sowie externen Speicher. Unterstützt werden zweimal 8k externer Programmspeicher (24 Bit) und zweimal 8k externer Datenspeicher (16 Bit). Da der ADSP-2181 nur einen 14 Bit breiten Adressbus besitzt (DM und PM), wird jeweils ein 8 k Block des externen Speichers über die oberen 8 k des internen Speichers geblendet.

3.6.3 Assembler

Der ADSP-2181 wurde in der Maschinensprache Assembler programmiert. Dies ist, soweit es DSPs betrifft, immer noch der Normalfall. Es besteht zwar die Möglichkeit den DSP in der Hochsprache C zu programmieren, allerdings erreichen die C-Compiler nicht die Geschwindigkeit und Effizienz wie ein handoptimierter Assembler Code. Die DSP-Assemblersprache unterscheidet sich durch die strukturellen Unterschiede zwischen DSP und GPP von der Assemblersprache eines normalen PC-Prozessors. So muss bei einem Speicherzugriff beispielsweise angegeben werden, ob die Daten im Programm- oder im Datenspeicher liegen. Ein weiterer Unterschied sind die Multifunctions. Im Falle des ADSP-2181 gibt es Instruktionen, die gleichzeitig eine ALU/MAC-Operation mit zwei Speicherzugriffen (PM und DM) ermöglichen. Erst diese erlauben die effiziente Programmierung von Filtern. Zur Erläuterung dienen die in den Abbildungen 3.6 und 3.7 gezeigten Ausschnitte aus dem Usercode. Dabei wird ersichtlich, dass die DSP-Assemblersprache durch die Benutzung einer algebraischen Syntax relativ leicht zu lesen ist. Das erste Beispiel ist ein Filter, der die zweite Ableitung bildet und nach dem Minimum sucht. Das zweite Beispiel zeigt, wie man in Assembler bedingt Verzweigungen erstellt.

Bildung der zweiten Ableitung

Die ersten beiden Instruktionen in Abbildung 3.6 initialisieren die Filterkoeffizienten. Die zweite Ableitung ($\triangle t \equiv 1$) wird durch $x(n) - 2 \cdot x(n-1) + x(n-2)$ gebildet, wodurch sich die Filterkoeffizienten ergeben. Die nächste Instruktion kopiert den Wert an der Adresse

```
 1   sr0=1;
 2   sr1=2;
 3   my0=dm(i0,m1);
 4   cntr=dm(Analen)
 5   mr=sr0*my0 (uu), my0=dm(i0,m1);
 6   do differloop until ce;
 7    mr=mr-sr1*my0 (uu), my0=dm(i0,m2);
 8    mr=mr+sr0*my0 (uu), my0=dm(i0,m1);
 9    ar=mr0-ay1, dm(i1,m1)=mr0;
10    if lt call negativeslope;
11   differloop: mr=sr0*my0 (uu), my0=dm(i0,m1);
```

Abbildung 3.6: Bildung der zweiten Ableitung

im Datenspeicher (DM), auf die das Adressregister `i0` zeigt, in das Register `my0`. Das Register `l`[6] muss für den jeweiligen Speicherzugriff korrekt initialisiert sein. In unserem Fall (kein zirkularer Buffer) mit dem Wert Null, was durch den XIA Code sichergestellt wird. Gleichzeitig wird mit diesem Befehl das Adressregister um den Wert von `m1`[7] (in unserem Falle ist `m1`=1) erhöht. Somit kann der nächste Zugriff auf den Datenspeicher direkt mit `i0` erfolgen. Die letzte Instruktion vor der Schleife ist eine Multifunction, die sich aus zwei, durch ein Komma getrennten, Teilen zusammensetzt.

Die `mr` Register, das sind `mr0` und `mr1` mit jeweils 16 Bit Breite und das `mr2` Register mit 8 Bit, falls es zu einem Überlauf kommt, gehören zur MAC-Einheit des DSPs. Der Wert der Multiplikation wird in die `mr` Register geschrieben und gleichzeitig ein neuer Wert in das `my0` Register geladen. Die Angabe `(uu)` zeigt an, das es sich um die Multiplikation vorzeichenloser Zahlen handelt. Für ALU-Operationen (Addieren, Subtrahieren) muss dies aufgrund der Darstellung im 2er Komplement nicht angegeben werden. Bei Multiplikationen/Division muss das Vorzeichen des Ergebnisses allerdings explizit berechnet werden. Die folgende Schleife wird nun solange ausgeführt, bis die in Zeile 4 an das Zählregister[8] übergebene Anzahl von Iterationen ausgeführt worden ist. Das Ende der Schleife wird durch das Label `differloop` angezeigt. Die Schleife führt die restlichen Operationen aus, wobei benutzt wird, dass bei MAC Operation der aktuelle Wert der `mr` Register durch Rückführung mit in die Rechnung einbezogen werden kann (Akkumulation). Die ALU Operation in Zeile 9 prüft, ob der neue Wert kleiner als ein Vergleichswert ist und wenn ja merkt sie sich diesen als neuen Vergleichswert. Daneben wird der Wert der zweiten Ableitung in einen eigenen Buffer im Datenspeicher gesichert (Adresspointer i1, Zeile 9).

Die bedingte Verzweigung

Um in Abhängikeit eines Rechnergebnisses entsprechenden Programmcode auszuführen, werden durch die Recheneinheiten bestimmte Flags gesetzt. Dies zeigen beispielsweise an,

[6] L wie LENGTH
[7] M wie MODIFY
[8] `cntr` wie Counter

```
1   ax0 = dm(ChanNum);
2   ar = pass ax0;
3   if eq jump SteepestSlope1;
4   ar=ar-1;
5   if eq jump Winkelanalyse1;
6   ar=ar-1;
7   if eq jump Winkelanalyse2;
8   ar=ar-1;
9   if eq jump Winkelanalyse3;
```

Abbildung 3.7: DSP Code

ob das Ergebnis null, negativ oder die Rechnung übergelaufen ist. Die erste Instruktion des Beispiels in Abbildung 3.7 lädt den Wert der Variablen `ChanNum` aus dem Datenspeicher in das Register `ax0`. Nun wird `ax0` an das ALU Register `ar` übergeben, wobei durch die `pass` Instruktion (keine Rechenoperation, aber Komparatortests) sichergestellt ist, dass ein Statusbit gesetzt wird. Falls das Ergebnis der "Rechenoperation" (die in diesem Fall eine reine Übergabe ist) Null ergibt, wird das zero Status (AZ) Bit im arithmetic status register (ASTAT) gesetzt. Dieses Bit wird bei der folgenden `if` Instruktion abgefragt. Dies ist durch die Angabe von `eq` (wie equal) in der Instruktion enthalten. Wenn die Auswertung ergibt, dass `ChanNum` gleich Null ist, so wird an die im Programmcode angegebene Adresse mit Label `SteepestSlope1` gesprungen. Falls nicht wird `ChanNum` um eins erniedrigt und nochmal getestet. Ist in `ChanNum` beispielsweise der Wert 3 gespeichert, so ergibt der Test in Zeile 3, dass `ar` ungleich Null ist. Ebenso die Test in Zeile 5 (`ar`=2) und Zeile 7 (`ar`=1). Erst in Zeile 9 hat `ar` den Wert 0 erreicht. Infolgedessen springt das Programm an die Programmadresse des Labels `Winkelanalyse3`.

3.7 Field Programmable Gate Arrays (FPGA)

Traditionsgemäß werden für die Digitale Signalverarbeitung die nach ihrem Haupteinsatzgebiet benannten Digitalen Signal Prozessoren verwendet. Heutzutage werden aber auch zunehmend FPGAs anstelle der DSPs verwendet, was von den FPGA-Hersteller durch die Bereitstellung optimierter Entwicklungstools bzw. fertigen DSP-Cores [34] und speziellen FPGA Architekturen [35] unterstützt wird, da die Verwendung der FPGA Technologie durch ihre Nebenläufigkeit/Parallelität eine hohe Performance bei gleichzeitig hoher Flexibilität verspricht. Dies macht sie zu einem ernsthaften Konkurrenten der DSPs. Zukünftige DSPs werden im Gegenzug ebenfalls aus mehreren parallel arbeitenden Recheneinheiten bestehen (z.B. [36]).

3.7.1 Allgemeiner Aufbau und Eigenschaften

FPGAs [37] gehören zur Familie der *rekonfigurierbaren* Hardwarebausteine, d.h. ihre Funktionalität wird nicht durch den Hersteller, sondern durch den Anwender festgelegt und kann

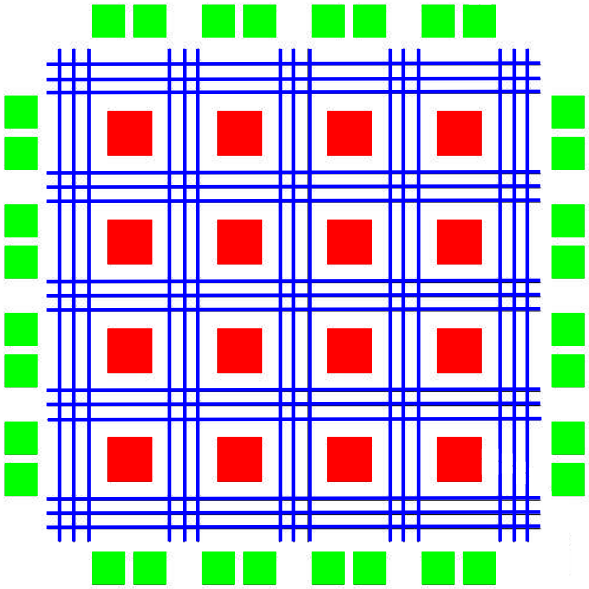

Abbildung 3.8: Schematischer Aufbau eines auf SRAM Technologie basierenden FPGAs [37]. Die Logikelemente, die die Funktionsgeneratoren enthalten, sind rot eingezeichnet und matrixartig angeordnet.Um diese herum befinden sich die Ein-/Ausgabeeinheiten (grün), welche mit den Pads des FPGAs verbunden sind. Dazwischen befindet sich eine Vielzahl an Verbindungsmöglichkeiten. Sowohl Logik, als auch die Verbindungen sind nicht festgelegt, sondern konfigurierbar.

darüberhinaus bei den auf SRAM[9]-Technologie basierenden FPGAs unendlich oft geändert werden. Die Konfiguration geschieht entweder über einen einzuspielenden *bitstream* oder über ein *Configuration PROM*[10]. Typischerweise werden FPGAs anstelle von ASICs[11] verwendet, wenn die Stückzahlen niedrig sind, kurze Produktionszyklen gefragt sind oder eine flexible Architektur benötigt wird (Teile der Architektur können auch zur Laufzeit geändert werden, was Computer mit variablem Befehlssatz ermöglicht [38]).

FPGAs sind "weiche" Hardware, sie sind (re-) programmierbar/konfigurierbar. Erst die Programmierung legt die Eigenschaften eines FPGAs fest. Dazu stellt das FPGA programmierbare Logikeinheiten, programmierbare Verbindungen und programmierbare Ein-/Ausgabeeinheiten zur Verfügung. Schematisch läßt sich dies in Abbildung 3.8 erkennen. Im allgemeinen existiert ein Feld von Logikeinheiten (rot), um welches die Ein-/Ausgabeeinheiten (grün) gruppiert sind. Verbunden werden diese Blöcke untereinender durch ein Netz verschiedener Leitungstypen (blau) unterschiedlicher Länge. Ein grundsätzliches architekturales Merkmal eines FPGAs läßt sich unter dem Begriff "sea of gates" [39] zusammenfassen: Im Gegensatz zu einem gewöhnlichen Mikroprozessor, bei dem die funktionalen Einheiten (z.B. Rechenwerke, Cache und I/O) auch räumlich getrennt gruppiert sind, sind diese bei den FPGAs über die gesamte Chipfläche "verstreut". Der im folgenden vorgestellte Xilinx FPGA hat eine Matrix Struktur.

Der Aufbau einer Logikeinheit ist von Hersteller zu Hersteller verschieden, was sich in der unterschiedlichen Namensgebung ausdrückt, z.B. nennt die Firma XILINX [40] ihre Logikeinheiten Configureable Logic Bocks (CLB) und die Firma Lucent [41] ihre Einheiten Programmable Function Units (PFU). Unterscheidungsmerkmale sind beispielsweise Anzahl der Logik- und Ein-/Ausgabeeinheiten, Anzahl und Größe der Funktionsgeneratortabellen (Look-Up-Tables), Anzahl der Register pro Logikblock und Variabilität der Signalwege zwischen und innerhalb der Funktionseinheiten.

Mit den Look-Up-Tables läßt sich *jede* beliebige Logik implementieren, da man die Wahrheitstafeln einer zu realisierenden Logik direkt in den SRAM-Speicher der Logikblöcke abbilden kann und das Problem sich somit auf die Adressierung der entsprechenden Zelle reduziert hat. In Abbildung 3.9 ist der CLB eines XILINX FPGAs aus der SPARTAN Serie gezeichnet. Man erkennt, dass eine Look-Up Table (LUT) des FPGAs vier unabhängige Eingänge besitzt. Somit muß eine LUT 2^4 Speicherzellen haben, ist also ein $16 \cdot 1$ Bit Speicher. Alle SRAM-FPGAs erlauben es deshalb mittlerweile, dass ein Logikblock zu einem reinen Speicher konfiguriert werden kann. Ein Eingang einer LUT entspricht einer Spalte Eingabeseite der Wahrheitstabelle und der Ausgang der Ergebnisspalte der Wahrheitstafel. Der Speicher der LUT wird nun so konfiguriert, dass bei den benötigten Eingabekombinationen (linke Seite in der Wahrheitstabelle) das gewünschte Ergebnis am Ausgang anliegt (rechte Seite der Wahrheitstabelle). Für ein UND mit 4 Eingängen enthalten alle Speicherzellen den Wert Null mit Ausnahme der Speicherzelle mit der Adresse 1111, die Eins enthält. Ohne ein einziges Logikgatter zu benutzen, nur mit Speicherzellen, lassen sich somit logische Operationen umsetzen. Rekonfigurierbare Bausteine, deren Logikblöcke aus UND/ODER-Gattern bestehen, und deshalb besonders schnell sind, sind

[9] static random access memory
[10] Programmable Read Only Memory
[11] Application Specific Integrated Circuits

ebenfalls gebräuchlich , sog. PLDs[12]. Im Gegensatz dazu ist bei einer LUT die Signallaufzeit unabhängig von der Logikoperation.

Aufwändige Operationen werden durch Kombination mehrer Logikblöcke umgesetzt. Leider ist die Verzögerung durch die Verbindung dieser Blöcke nicht fest, sondern abhängig von der Art der Leitung und der Länge der damit hergestellten Verbindung. Im allgemeinen hat man es mit getakteten Schaltungen zu tun, d.h. die berechneten Werte müssen synchron zu einer Flanke des Taktsignals übergeben werden. Damit dies möglich ist, enthält jeder Logikblock Register, die mit beliebigen (internen und externen) CLK Signalen beschaltet werden können. Ein Problem beim Schaltungsentwurf für FPGAs besteht nun darin, das Design so im FPGA zu plazieren und verbinden, dass die durch die Taktfrequenz erlaubten Signallaufzeiten von Register (im Logikblock x) zu Register (im Logikblock y) nicht überschritten werden[13]. Bei hohen Taktfrequenzen ist es deshalb üblich, das Prinzip des *Pipelining* [42] anzuwenden. Eine Operation wird dabei in kleinere Teiloperationen zerlegt, deren Teilergebnisse getaktet an die nächste Operation übergeben werden. Das Design läuft dann mit der geforderten Taktfrequenz, allerdings erhöht sich die Latenzzeit[14] um die Anzahl der Pipelinestufen. In der Signalverarbeitung muss ein durchgehender Datenstrom mit einer bestimmten Frequenz vom ADC übernommen und verarbeitet werden, dabei ist es unerheblich welche Latenz der kontinuierliche Strom an Ergebnissen hat.

3.7.2 Das XILINX XCS30/20-FPGA

Das XIA DGF-4C Modul enthält in der benutzten Revision fünf FPGAs: Pro Kanal einen XCS30 FPGA mit $24 \cdot 24$ CLBs der den Energiefilter und den Triggerfilter enthält und einen XCS20 mit $20 \cdot 20$ CLBs der das CAMAC-Interface bereitstellt. Im Bild 3.9 erkennt man, dass ein CLB aus zwei LUT mit je 4 unabhängigen Eingängen und einer LUT mit drei Eingängen besteht. Deshalb können z.B. gleichzeitig zwei beliebige und unabhängige Funktionen von vier Variablen und eine mit drei Variablen oder eine beliebige Funktion mit fünf Variablen implementiert werden. Eine Besonderheit ist die *Fast Carry Logic*, die es erlaubt extrem schnelle Addierer/Subtrahierer zu bauen, da das Carry-Signal direkt aus der F/G-LUT an den übergeordneten CLB weitergegeben wird. Ein CLB kann zu einem 32·1 Bit bzw. zweimal 16·1 Bit Speicher konfiguriert werden. Die Ausgänge der LUTs können entweder direkt auf den Ausgang einer CLB gelegt werden oder über ein zwischengeschaltetes Register mit Set/Reset Eingang. Der Signalweg wird durch Multiplexer gesteuert, die zwischen den Eingangsfunktionsgeneratoren (F- und G-LUT) und der H-LUT, sowie zwischen H-LUT und den Registern plaziert sind.

Die Vielzahl der Routingmöglichkeiten innerhalb des FPGAs sind in Abbildung 3.10 gezeigt. Grundsätzlich gibt es drei Arten von Verbindungen: je acht vertikale und horizontale single length lines, vier vertikale und horizontale doubles length lines und long lines. Single length lines verbinden direkt benachbarte PSMs (**P**rogrammable **S**witching **M**atrix). Damit lassen sich die CLBs mit ihren direkten Nachbarn verschalten, es gibt allerdings keine direkte Verbindung zwischen zwei CLBs. Eine Verbindung verläuft immer über eine PSM, die im Detail in Abbildung 3.11 gezeigt ist. Dadurch ist eine Verbindung über

[12] Programmable Logic Device

[13] Timing Constraints, kritischer Pfad

[14] Zeit zwischen dem Beginn einer Operation und der Übergabe des Ergebnisses

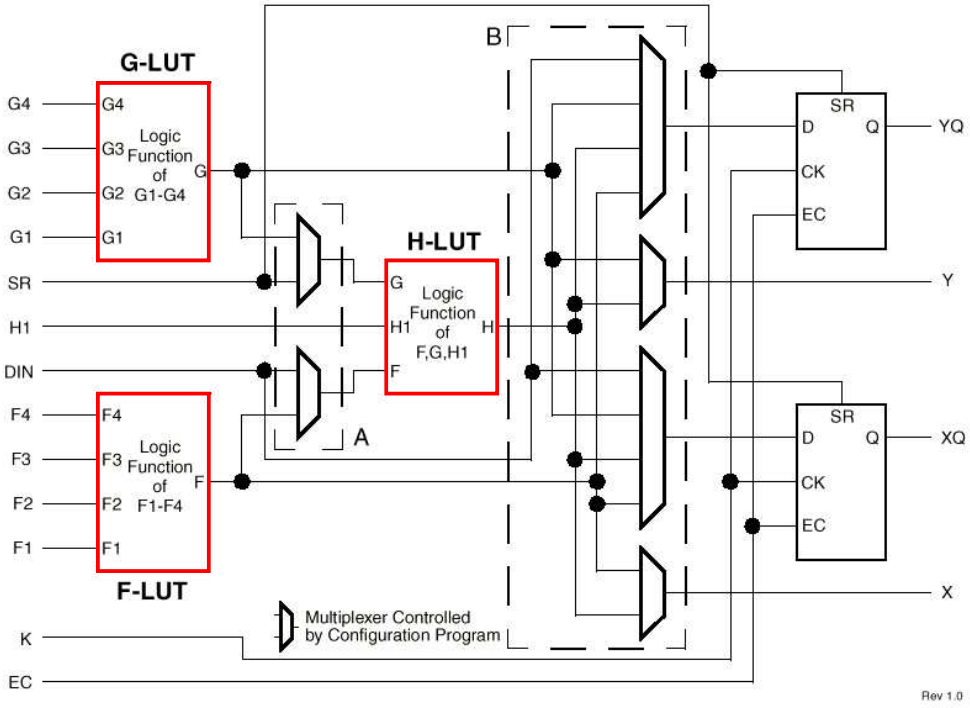

Abbildung 3.9: Aufbau der CLB im XILINX SPARTAN FPGA [40]. Die beiden Funktionsgeneratoren G und F haben jeweils vier Eingänge. Ihre Ausgänge können mittels Multiplexer wahlweise mit dem Eingang des Funktionsgenerator H, einem Register oder direkt mit einem Ausgang (X bzw. Y) des CLBs verbunden werden.

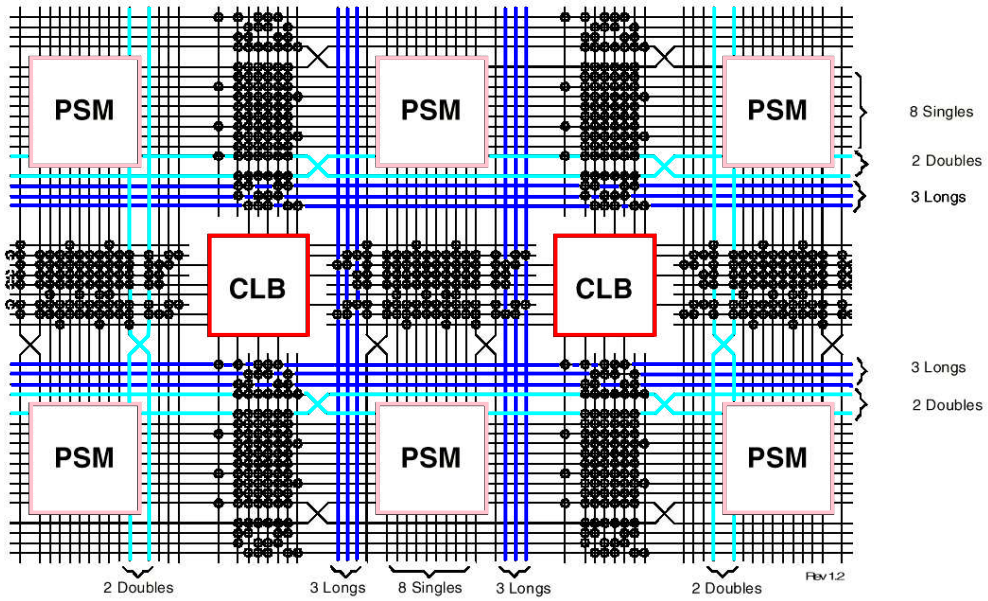

Abbildung 3.10: Routingresourcen im XILINX Spartan FPGA [40]. Man beachte, dass es keine direkte Verbindung zwischen benachbarten CLBs gibt. Deshalb verläuft eine Verbindung immer über eine Programmable Switching Matrix (PSM). Hier kommt ebenfalls die SRAM-Technologie zum Tragen. Erstens wird sie benutzt zum durchschalten der Transistoren, die die Verknüpfung zweier Leitung innerhalb der PSM steuern und zweitens zur Ansteuerung eines Multiplexers, der die Verbindung zwischen den Leitungen (single length, double length und long wires) und dem Logikblock (CLB) herstellt. Letztere sind als schwarze Punkte in der Abbildung eingezeichnet. Durch die Verwendung der SRAM-Technologie sind alle Verbindung konfigurierbar.

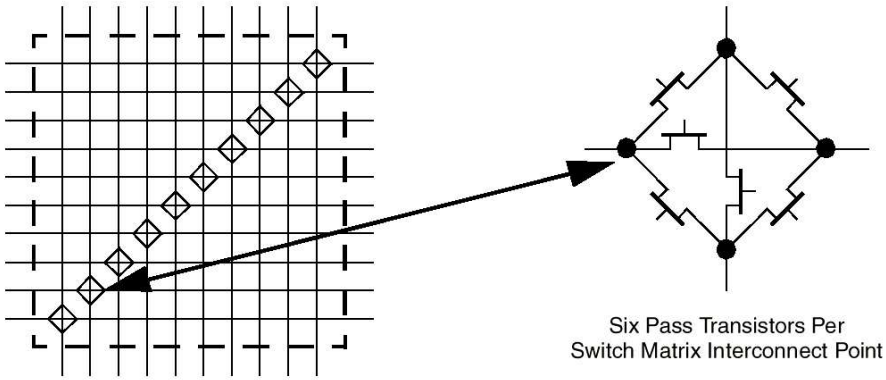

Abbildung 3.11: Programmable Switching Matrix (PSM) der XILINX Spartan FPGAs [40]
.

größere Entfernungen mittels single lenght lines mit einer hohen Verzögerung verbunden. Mittels double length lines verbindet man einen CLB mit seinem übernächsten Nachbarn. Double lenght lines lassen deshalb eine PSM aus. Die long lines verlaufen nie über eine PSM und haben daher die geringste Verzögerung. Sie werden deshalb für zeitkritische Signale benutzt. Nicht eingezeichnet sind die davon unabhängigen globalen Netzwerke, die zur Verteilung von Clk- oder Steuersignalen bzw. extrem zeitkritischen Signalen benutzt werden, sowie das fast carry-Netzwerk, mit dessen Hilfe äußerst schnelle Addierer implementiert werden können. In der Zeichnung der PSM (Abbildung 3.11) erkennt man, wie mit Hilfe von sechs Transistoren das horizontale/vertikale routing zustande kommt.

Die I/O-Einheiten (IOB=Input/Output Block) stellen die Verbindung zwischen der implementierten Logik und den Pins des FPGAs her. Ein IOB kann sowohl reine Input bzw. Ouput als auch bidirektionale Signale behandeln. Diese können wiederum direkt oder über ein Register weitergeleitet, wobei für Ein- und Ausgabe zwei getrennte Clk-Signale verwendet werden können. Die Ausgabesignale können wahlweise auf TTL oder CMOS-Level eingestellt werden.

3.7.3 VHDL - eine Hardwarebeschreibungssprache

VHDL [43, 44] ist eine Hardwarebeschreibungssprache, deren Entwicklung vom amerikanischen Verteidigungsministerium initiert wurde (daher die Ähnlichkeit zur Programmiersprache ADA). Der Name VHDL leitet sich ab von **V**HSIC **H**ardware **D**escription **L**anguage, wobei VHSIC für das **V**ery **H**igh **S**peed **I**ntegrated **C**ircuits Programm des amerikanischen Verteidingungsministeriums steht. Ziel war es eine Sprache zur Dokumentation elektronischer Systeme zu entwickeln, was den Firmen IBM, Texas Instruments und Intermetrics aufgetragen wurde. 1987 wurde die Sprache VHDL von der IEEE[15] als IEEE 1075-1987 zum ersten Mal standardisiert. Infolge der hohen Komplexität kann nur ein kurzer Einblick in diese Sprache gewährt werden. Details sind der angegeben Literatur zu entnehmen.

Der Entwurf einer Schaltung mit VHDL kann auf drei Entwurfs**ebenen** stattfinden: Algorithmischer, Register-Transfer und Logikebene. Dabei unterscheidet man zwei verschiedene Entwurfs**sichten**: Verhaltens- und strukturale Sicht. Auf der algorithmischen Ebene wird ein System durch parallele Prozesse beschrieben, die in Form von Funktionen, Prozeduren und Prozessen implementiert werden. Dies wird in der strukturalen Sichtweise durch Modellierung abstrakter Funktionsblöcke die über Signale miteinander verbunden sind geschehen, wohingegen das Verhalten in der algorithmische Darstellung durch logischen Operationen angewandt auf Variablen und Signalen beschrieben wird. Die Register-Transfer Ebene (RTL) ist durch die Verwendung von Takt und Reset Signalen gekennzeichent, Operation werden in Abhängigkeit des Wertes oder der Flanke des Taktsignals ausgeführt. Statt Variablen werden Register verwendet. In der strukturalen Sichtweise wird die Verbindung einzelner Elemente über Signale umgesetzt, in der Verhaltenssicht werden hauptsächlich endliche Automaten (state machine) benutzt. Auf der Logikebene werden Systeme durch ihren Aufbau aus Grundelementen wie UND- oder ODER-Gattern beschrieben, wobei nun auch zeitliche Eigenschaften wie Signallaufzeiten berücksichtigt werden. Die Verhaltenssicht bedient sich der Booleschen Arithmetik, um die Operationen darzustellen. In der

[15] Institute of Electrical and Electronics Engineers

```
1   entity steepestslope is
2   port(adc_sample                    :  in  std_logic_vector(11 downto 0); -- adc samples
3        fasttrigger                   :  in  std_logic;                -- enable
4        analyselen                    :  in  std_logic_vector(15 downto 0); -- variable
5        userdelay                     :  in  std_logic_vector(15 downto 0); -- variable
6        clk                           :  in  std_logic;                -- 40 Mhz clock
7        reset                         :  in  std_logic;                      -- ansynchroner reset
8        outputvalid                   :  out std_logic;                -- data valid
9        ergebnis                      :  out std_logic_vector(15 downto 0));-- TimeToSteepestSlope
10  end steepestslope;
```

Abbildung 3.12: Beispielcode zur Entity Anweisung in VHDL.

```
1   architecture behavioral of register is
2   mein_register: process (clk, reset, new_val)
3   begin
4          if reset = '1' then
5                          output <= "0000000000000000";
6          elsif clk'event and clk='1' then
7                  if new_val = '1' then
8                          output $<=$ uhrzeit;
9                  end if;
10         end if;
11  end process mein_register;
12  end behavioral;
```

Abbildung 3.13: Beispielcode für eine Architekturbeschreibung eines Register.

strukturalen Sicht werden die Informationen aus den Herstellerbibliotheken verwendet, in denen die spezifischen Eigenschaften der Grundelemente (Geometrie, Verzögerungszeit) definiert sind.

Jede VHDL Beschreibung beginnt mit der Definition der Schnittstellen. Die *Entity* enthält die Definition der Ein- und Ausgänge, sowie eventuelle Definitionen von Konstanten. Dies soll am Beispiel 3.12 erläutert werden. In der Entity namens `steepestslope` werden Ein- und Ausgänge des Typs `std_logic_vector` mit unterschiedlicher Bitbreite durch die `port`-Anweisung definiert. Der Datentyp `std_logic(_vector)` ist in der IEEE-Bibliothek definiert und kann die Werte 0, 1, X (undefiniert) und Z (hochohmig) annehmen. Kommentare werden in VHDL mit `--` eingeleitet. Nachdem die Schnittstellen deklariert sind, kann die Funktionalität programmiert werden. Dies geschieht innerhalb der *Architecture*, wobei sowohl Verhaltensbeschreibungen als auch strukturale Modellierungen erlaubt sind. Einer Entity können mehrere Architekturen zugeordnet werden.

Eine wichtige Anweisung ist die Deklaration eines Prozesses (*process*). Der kurze VHDL-Code in Abbildung 3.13 modelliert ein 16 bit breites Register mit asynchronem Reset. Die erste Anweisung zeigt an, dass es sich um die Beschreibung einer Architektur namens `behavioral` der Entity `register` handelt. Danach wird der Prozeß definiert, wobei sich in Klammern die Liste aller Signale befindet auf die der Prozess sensitiv ist, d.h. auf deren Änderung hin der Prozess ausgeführt wird. Innerhalb des Prozesses werden in einer if-Schleife die Ereignisse von Interesse abgefragt. Ist das `reset` Signal 1 so wird das Register auf Null gesetzt. Ändert sich der Wert des Clocksignals (abgefragt über das Attribut `event` mit dem Attributoperator `'`) auf 1, so wird, falls auch das Signal `new_val` den Wert 1 hat, der Wert am Eingang das Registers übernommen. Das Ganze wird als ein Regi-

ster mit vorgeschaltetem Multiplexer, dessen Eingänge das rückgeführte Ausgangssignal des Registers und das Signal `uhrzeit` sind, implementiert. In Abhängigkeit vom Wert des Signals `new_val` wird entweder ein neuer Wert oder der alte Wert übernommen. Wertzuweisungen an Signale geschehen mit dem Operator `<=`, Zuweisungen an Variablen mit dem Operator `:=`. VHDL erlaubt es, einer Entity mehrere Architekturen zuzuordnen. Der Vorteil liegt darin, dass einfache Modelle zur Simulation des Verhaltens im Anfangsstadium eines Entwurfs mit fortschreitender Entwicklung durch komplexere und realistischere Beschreibungen bei gleicher Funktionalität ausgetauscht werden können. Die Konfiguration (*Configuration*) legt fest, welche Architektur einer Entity verwendet wird. Strukturale Modellierung geschieht durch Aufbau eines Modells aus Komponenten die in kompilierten Bibliotheken zur Verfügung stehen. Das Standardbibliotheksverzeichnis ist `WORK`. Um Komponenten einzubinden gibt es die `component` Anweisung. Sie entspricht im Aufbau der `entity`-Anweisung. Von einen einmal mit der `component` Anweisung eingebundenes Modell können beliebig viele Instanzen benutzt werden.

Das Beispiel 3.14 zeigt ein strukturales Model auf der Register-Transfer-Ebene (mit Taktsignal). Die erste Anweisung bindet alle benötigten Bibliotheken ein. Die `use`-Anweisung funktioniert nach dem Schema `library_name.package_name.item`. Von Zeile 2 bis 4 werden aus IEEE Bibliothek drei vollständige Packages mit kompilierten VHDL-Einheiten eingebunden. Danach werden mit der entity-Anweisung die Ein- und Ausgänge definiert. Innerhalb der Architekturbeschreibung werden zuerst einige interne Signale definiert, die weiter unten dazu benutzt werden die einzelnen Subkomponenten zu verbinden. Danach werden die Komponenten eingebunden, in diesem Fall ein Addierer und ein Subtrahierer aus der Synopsys DesignWare Bibliothek und ein Register. Die benötigte Anzahl an Komponenten wird anschliessend instanziert und über die `port map`-Anweisung "verdrahtet". Das Signal `sig_zero` wird dauerhaft auf 0 gehalten, um die Synopsys Komponenten zu konfigurieren. In der `configuration`-Anweisung wird festgelegt für welche Instanzen der Komponente `register` welches Element aus der Bibliothek `WORK` verwendet wird.

Eine vollständige VHDL-Beschreibung kann mit einem VHDL-Simulator getestet werden. Dazu muss eine Testbench erstellt werden, die das zu testende Design mit Testvektoren versorgt und eventuell auch schon die Auswertung des Outputs vornimmt. Die erstellte VHDL-Beschreibung wird kompiliert und dabei auf korrekte Semantik überprüft. Das kompilierte Design wird als Komponente in die Testbench eingebunden, die selbst als Entity ohne Ein- und Ausgänge beschrieben wird. Nun wird die Testbench kompiliert und anschliessend mit einem Simulator z.B. Synopsys 1076 VHDL Debugger (vhdldbx) aufgerufen. Nach Ablauf der Simulation, der durch setzen von Breakpoints gesteuert werden kann, kann die Antwort des Designs auf die Stimuli z.B. mit dem Synopsys Waveform Viewer betrachtet und mit den Erwartungen verglichen werden. Entspricht das Ergebnis den Erwartungen, kann zur Synthese des Designs übergegangen werden. Unter Synthese versteht man nach [45] die Transformation einer Verhaltensbeschreibung in eine Strukturbeschreibung aus Elementen niedrigerer Ebene. Die Kunst besteht in der Eingabe eines gut synthetisierbaren VHDL-Codes. Das Ergebnis liegt meistens in Form einer Netzliste mit Bauteilen aus einer technologieunabhängigen (generischen) Bibliothek vor. Erst im nächsten Schritt - dem Technology Mapping - legt man sich auf eine bestimmte Technologie (z.B. FPGA XCS30 der Firma XILINX) fest. Hierbei werden die generischen Bauteile durch diejenigen der ausgewählten Technologie ersetzt und die Netzliste erneut optimiert. Letztendlich müssen die

```
1  LIBRARY IEEE, synopsys, DWARE, DW01, WORK;
2  USE IEEE.std_logic_1164.ALL;
3  USE IEEE.std_logic_signed.ALL;
4  USE IEEE.std_logic_arith.ALL;
5  use WORK.all;
6  use Synopsys.attributes.all;
7  use DWARE.DWpackages.all;
8  use DW01.DW01_components.all;
9
10 -- ENTITY
11 entity steslofir is
12 port(adc                          :  in  std_logic_vector(11 downto 0);  -- adc samples
13      clk                          :  in  std_logic;                      -- clock
14      reset                        :  in  std_logic;                      -- ansynchroner reset
15      ergebnis                     :  out std_logic_vector(13 downto 0)); -- ergebnis
16 end steslofir;
17
18 -- ARCHITECTURE
19 architecture structural of steslofir is
20 signal sig_zero, carry_out                          : std_logic;
21 signal sig1, sig2, sig3                             : std_logic_vector(11 downto 0);
22 signal sig4, sig4a, sig5, sig5a                     : std_logic_vector(12 downto 0);
23 signal pre_ergebnis                                 : std_logic_vector(13 downto 0);
24
25 attribute implementation of add_instance : label is "cla";  -- verwende carry look ahead addierer
26
27 component DW01_add is
28 generic(width  :  integer);
29 port( A, B     :  in std_logic_vector((width-1) downto 0);
30       CI       :  in std_logic;
31       SUM      : out std_logic_vector((width-1) downto 0);
32       CO       : out std_logic);
33 end component DW01_add;
34 component DW01_sub is
35 generic(width  :  integer);
36 port( A, B     :  in std_logic_vector((width-1) downto 0);
37       CI       :  in std_logic;
38       DIFF     : out std_logic_vector((width-1) downto 0);
39       CO       : out std_logic);
40 end component DW01_sub;
41 component regis is
42 generic(width : integer);
43 port (reg_eingang            :  in std_logic_vector((width-1) downto 0);
44       clk, reset             :  in std_logic;
45       reg_ausgang            : out std_logic_vector((width-1) downto 0));
46 end component regis;
47
48 begin
49 sig_zero <= '0';
50 -- adder plus pipeline register
51 add_instance : DW01_add
52 generic map(width => 12)
53 port map(sig1, sig3, sig_zero, sig5(11 downto 0), sig5(12));
54 adderpipe: regis
55 generic map(width => 13)
56 port map(sig5, clk, reset, sig5a);
57 -- shifter pipe
58 sig4(12 downto 1) <= sig2(11 downto 0);
59 sig4(0) <= '0';
60 shiftpipe: regis
61 generic map(width => 13)
62 port map(sig4, clk, reset, sig4a);
63 -- substraction plus pipeline register
64 sub_instance : DW01_sub
65 generic map(width => 13)
66 port map(sig5a, sig4a, sig_zero, pre_ergebnis(12 downto 0) , pre_ergebnis(13));
67 subpipe: regis
68 generic map(width => 14)
69 port map(pre_ergebnis, clk, reset, ergebnis);
70 -- filter taps
71 regi1 : regis
72 generic map(width => 12)                                    -- tap 1
73 port map(adc, clk, reset, sig1);
74 regi2 : regis
75 generic map(width =>12)                                     -- tap 2
76 port map(sig1, clk, reset, sig2);
77 regi3:  regis
78 generic map(width =>12)                                     -- tap 3
79 port map(sig2, clk, reset, sig3);
80 end structural;
81
82 -- CONFIGURATION
83 -- synopsys translate_off
84 configuration steslofir_CFG of steslofir is
85 for structural
86        for regi1, regi2, regi3, subpipe, adderpipe, shiftpipe : regis
87                use configuration WORK.regis_CFG;
88        end for;
89 end for;
90 end steslofir_CFG;
91 -- synopsys translate_on
```

Abbildung 3.14: Beispiel einer strukturalen Architektur Beschreibung. Dieser Code zeigt, wie die zur Bildung der zweiten Ableitung nötigen Komponenten, nämlich Addierer und Subtrahierer, eingebunden und durch Signale verknüpft werden.

Komponenten des Designs noch auf einer vorgegeben (Chip-)Fläche oder innerhalb eines FPGAs plaziert und verbunden werden, das sog. Place and Route.

Vorteile von VHDL sind die vielfältigen Modellierungsmöglichenkeiten, der hohe Grad an Abstraktion und die technologieunabhängige Beschreibung, sodass ohne große Codeänderungen ein Design auf verschiedene Technologien portierbar ist. Es gibt allerdings noch keinen Standard zur Modellierung analoger Schaltkreise bzw. gemischter (analog und digitaler) Systeme. Von Nachteil ist weiterhin, dass die verschiedenen Syntheseprogramme jeweils nur einen Teil der standardisierten VHDL Instruktionen unterstützen und unterschiedliche Synthesestrategien anwenden, was zu einem von den jeweils verwendeten Tools abhängigen Programmierstil führt. Für den SYNOPSYS DESIGN COMPILER [46] beispielsweise sind Syntheseergebnisse verschiedener VHDL-Konstrukte in [45] aufgeführt.

Kapitel 4

Das CAMAC-Modul XIA DGF-4C

'Just because I don't CARE doesn't mean I don't understand.'
Homer J. Simspon

Die digitale Verarbeitung der Signale eines Germaniumdetektors wurde schon von mehreren Gruppen in Angriff genommen. In [47] wird ein analoger Pulshöhenmesser mit digitaler Pulsformanalyse (durch einen DSP) kombiniert. In [48] wird ein relativ aufwändiges System vorgestellt in dem, mit Ausnahme des Triggers, alle weiteren Verarbeitungsschritte in DSP- bzw. FPGA-Technologie implementiert sind. Die Anwendung digitaler Filter ermöglicht geringere Totzeiten und durch Pulsformanalyse können nachträglich Korrekturen wie Ballistic Deficit, CFD-Algorithmen oder Algorithmen zur Ortsbestimmung ausgeführt werden. Die MINIBALL Kollaboration entschied sich für die Verwendung des kommerziellen CAMAC-Moduls DGF-4C der Firma X-Ray Instrumentation Associates Inc. (XIA). Dieses Modul ist durch die Implementierung einfacher Algorithmen günstig herzustellen. Das folgende Kapitel gewährt einen kurzen Einblick in Aufbau und Funktionsweise, wobei dies kein Ersatz für die mitgelieferten Handbücher [18] sein soll. Da die von XIA implementierten Rechenprozeduren und Korrekturen Firmengeheimnisse sind, können über diese nur Vermutungen angestellt werden. Es könnte sich deshalb als hilfreich erweisen, die entsprechenden Patentschriften zu Rate zu ziehen.

4.1 Überblick und Aufbau

Der prinzipielle Unterschied zwischen analoger und digitaler Signalverarbeitung ist in Abbildung 4.1 gezeigt. Anstatt den Maximalwert des Hauptverstärkers/Shapers mittels eines Peak-Sensing-ADCs zu digitalisieren, wird der vollständige Vorverstärkerpuls digitalisiert und durch digitale Filter geformt. Um der Digitalisierung gerecht zu werden, muss das Rauschen durch den analogen Shaper auf weniger als ein LSB (Least Significant Bit) reduziert werden. Die digitale Methode muss, um eine ausreichende Zeitauflösung erreichen zu können, eine hohe Samplingrate verwenden, da sie auf einem eingeschränkten Datensatz operiert. In der gegenwärtigen Revision findet auf dem Modul DGF-4C ein 12 Bit ADC von Analog Devices mit einer Samplingfrequenz von 40 MHz Verwendung, d.h. alle 25 ns liefert der ADC ein weiteres Sample. Damit keine Frequenzen oberhalb der halben Samplingfrequenz durch Aliasing die Signalqualität verschlechtern, befindet sich als erste

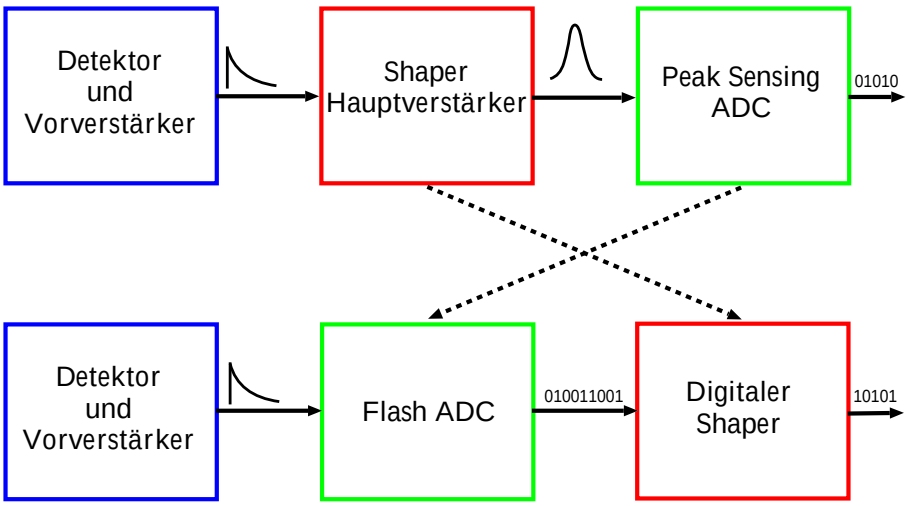

Abbildung 4.1: Unterschied zwischen analoger und digitaler Signalverarbeitung . Die analoge Variante formt den Puls mittels analoger Schaltkreise. Ein Gaussfilter ist analog einfach zu implementieren, hat allerdings eine schlechtere SNR-Performance als ein Trapezfilter. Die digitale Methode operiert auf den gesampelten Daten des Vorverstärkers. Die Pulsform kann durch entsprechende Wahl der Filterkoeffizienten manipuliert werden.

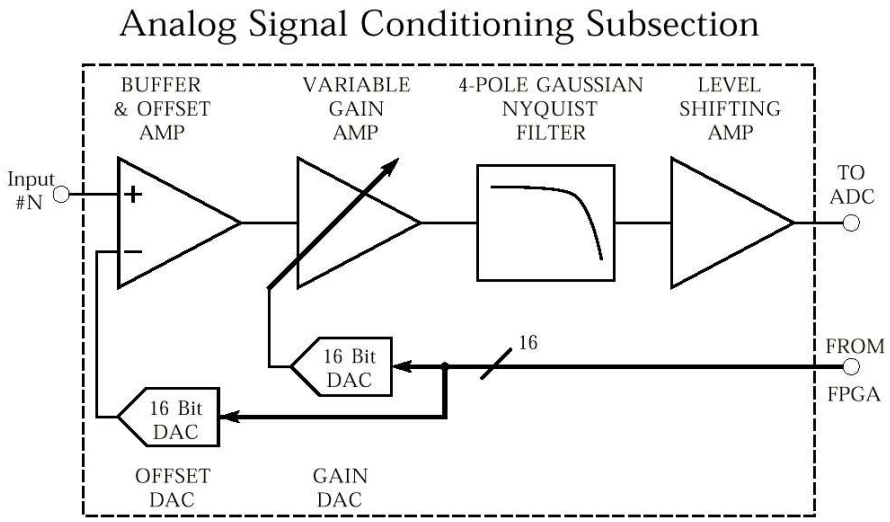

Abbildung 4.2: Schematischer Aufbau des analogen Teil des Moduls DGF-4C [18] . Das einkommende Vorverstärkersignal ist in Offset und Verstärkung durch Programmierung der DACs manipulierbar. Anschließend wird durch einen Filter die Bandbreite des Signals der Samplingrate entsprechend reduziert und schließlich der Signallevel an den ADC angepasst. Der ADC erwartet ein Eingangssignal mit einer Amplitude von 1 V Peak zu Peak, welches um +2,4 V zentriert ist, da der ADC nur mit +5V versorgt wird.

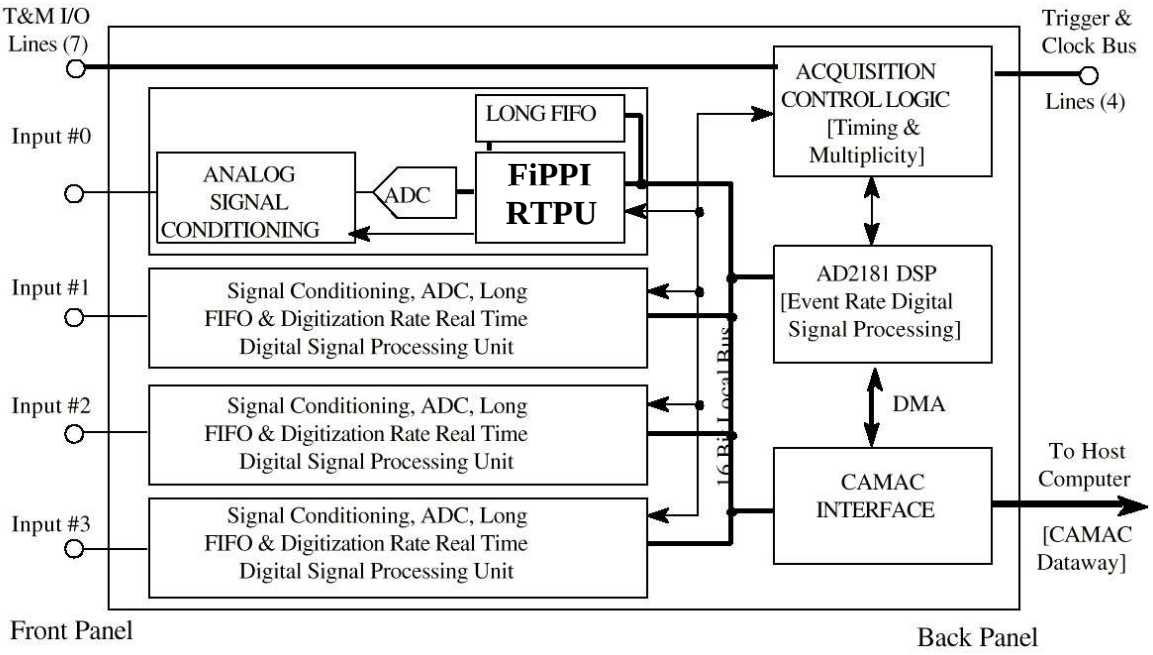

Abbildung 4.3: Schematischer Aufbau des Moduls DGF-4C [18] . Die vier Kanäle sind eingangsseitig gleich aufgebaut. Für den Kanal 0 ist die Anordung nochmal im Detail gezeigt. Der DSP kann über einen Bus mit den FPGAs in Verbindung treten. Das CAMAC-Interface sorgt für die Verbindung zwischen Host-PC und Modul.

Baugruppe nach dem Eingang die Analog Signal Conditioning (ASC)-Einheit. Sie ist schematisch in Abbildung 4.2 gezeigt. Sie dient dazu das einkommende Signal in Offset und Verstärkung den Erfordernissen anzupassen und weiterhin dazu Frequenzen oberhalb 20 MHz durch einen Filter zu entfernen. Als letzte Stufe vor dem ADC durchläuft das Signal einen Verstärker mit festem Gain, der dazu dient das gesamte Signal in den sensitiven Bereich des ADCs zu schieben. Wie aus der Übersicht erkennbar (Abbildung 4.3) werden die digitalen Daten des ADCs aufgespalten: eine Kopie erhält der FIFO und eine Kopie der FiPPI[1] (auch RTPU[2] genannt). Diese Einheiten sind in vierfacher Ausführung auf dem Board vorhanden.

Der 1024 Samples lange FIFO dient zur Speicherung der ADC Daten. Diese können durch den DSP zur anschliessenden Pulsformanalyse ausgelesen werden. Der FiPPI (ein XILINX XCS30 FPGA) nutzt das Signal zweifach (Vergleiche Abbildung 4.5). Mithilfe eines kurzen Dreiecks-/Trapezfilters generiert er einen Eventtrigger, sobald dieser digitale Filter eine einstellbare Schwelle überschritten hat (LE-Trigger). Alternativ ist es mittlerweile auch möglich einen (sog. Hardware-) CFD-Trigger (mit einer festen Schwelle von 25 % und variablem Delay) zu benutzten. Ein längerer Trapezfilter ersetzt den Shaper des analogen Äquivalents. Sein Ausgangssignal liefert die Energieinformation. Beide Filter werden von einem endlichen Automaten ("Pileup Inspector") auf eine ausreichende Isolation der Pulse hin überwacht.

[1] **F**ilter **P**rocessor and **P**ileup **I**nspector
[2] **R**eal **T**ime **P**rocessing **U**nit

Alle vier Kanäle werden von einem digitalen Signalprozessor (ADSP-2181) überwacht, der die Filter- und FIFO-Daten auslesen und verarbeiten muss. Desweiteren stellt der DSP den Ausgabepuffer zur Verfügung. In einem von XIA vorgegebenen Format stehen dort die Eventdaten zur Auslese bereit. Mithilfe eines ebenfalls in FPGA-Technologie implementierten CAMAC-Interfaces können Daten über den CAMAC-Bus an einen PC übertragen werden. Die Programmierung des DSPs und der FPGAs erfolgt ebenfalls über den CAMAC-Bus. Damit größere Experimente (MINIBALL) mit dieser Elektronik durchgeführt werden können, muss es eine Möglichkeit geben Information zwischen den einzelnen Karten auszutauschen. Zu diesem Zweck gibt es einen PECL[3]-Bus am rückwärtigen Ende eines jeden Moduls über dessen Leitungen Clk- und Triggersignale (FPGA und DSP Trigger) über mehrere Karten hinweg verteilt werden können. Module, welche keinen eigenen FPGA-Eventtrigger erzeugen (das sind beispielsweise die Karten, die nur mit den Segmentsignalen des MINIBALL Detektors verbunden sind), werden über diesen Bus mit den Triggersignalen einer anderen Karte versorgt. An der Frontblende befinden sich Trigger-, Busy-, Multiplicity-Ausgänge und First Level- und Second Level-, Synch, Multiplicity-Eingänge. Der Second Level-Trigger funktionieren in der vorliegenden Revision noch nicht vollständig. Allerdings speichert die aktuelle Revision die Ankunfstzeit eines Second Level Triggers (48 Bit). Der Trigger-Ausgang liefert ein Triggersignal welches durch den DSP generiert wird, wohingegen der Multiplicity-Ausgang mit dem schnellen Fast-Trigger-Signal des FiPPI verbunden ist. Damit alle Karte gemeinsam starten, werden die Busy-Signale aller beteiligten Karten auf ein Oder gegeben und das Ergebnis auf den Synch-Eingang der Karten gelegt ("Busy-Synch-Loop"). Findet an diesem Eingang ein Übergang von 1 nach 0 statt, so wird die Datenaufnahme auf dieser Karte gestartet und, falls ein entsprechendes Bit gesetzt ist, die "Wanduhr" (16 Bit DSP-Timer wird zu 48 Bit Zähler aufaddiert) der Datenaufnahme zurückgesetzt. Die Warteschleife dieser Methode ist 0.1 μs (4 DSP Instruktionen) lang.

Der digitale Ansatz hat die folgenden Vorteile:

- Alle dem ADC folgenden Baugruppen (FIFO, FPGA und DSP) arbeiten mit exakten Kopien der digitalisierten Wellenform. Die Pulsform kann somit nachträglich untersucht werden.

- Noch bevor ein Event endgültig akzeptiert ist, kann eine erste Verarbeitung (Energiefilter) stattfinden.

- Durch die frühe Digitalisierung entsteht keine Totzeit, nur eine erhöhte Latenzzeit (ADC Pipeline $\sim$ 2.5 Takte), wohingegen bei analoger Standardelektronik, die erst nachdem ein Ereignis akzeptiert wurde den Shaper digitalisiert, zusätzliche Totzeit entsteht, da sich die Digitalisierungzeit zur Filterlänge hinzuaddiert.

- Die FiPPIs (FPGAs) und der DSP sind reprogrammierbar, d.h. ihre Funkionalität ist nicht festgelegt und kann (auch durch den End-Benutzer) erweitert werden.

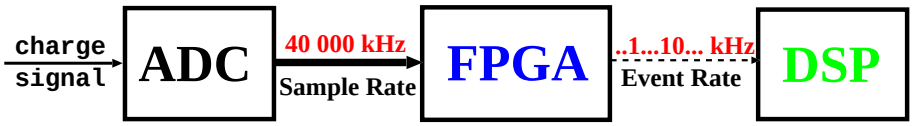

Abbildung 4.4: Prinzip der Datenreduktion durch den FiPPI-Trigger . Der ADC erzeugt aus dem ankommenden Datenstrom alle 25 ns ein 12 Bit Datenwort. Der FPGA analysiert diesen Datenstrom in Echtzeit. Entdeckt dieser ein Ereignis, so liest der DSP die Filterdaten aus dem FPGA und beginnt mit der weiteren Analyse.

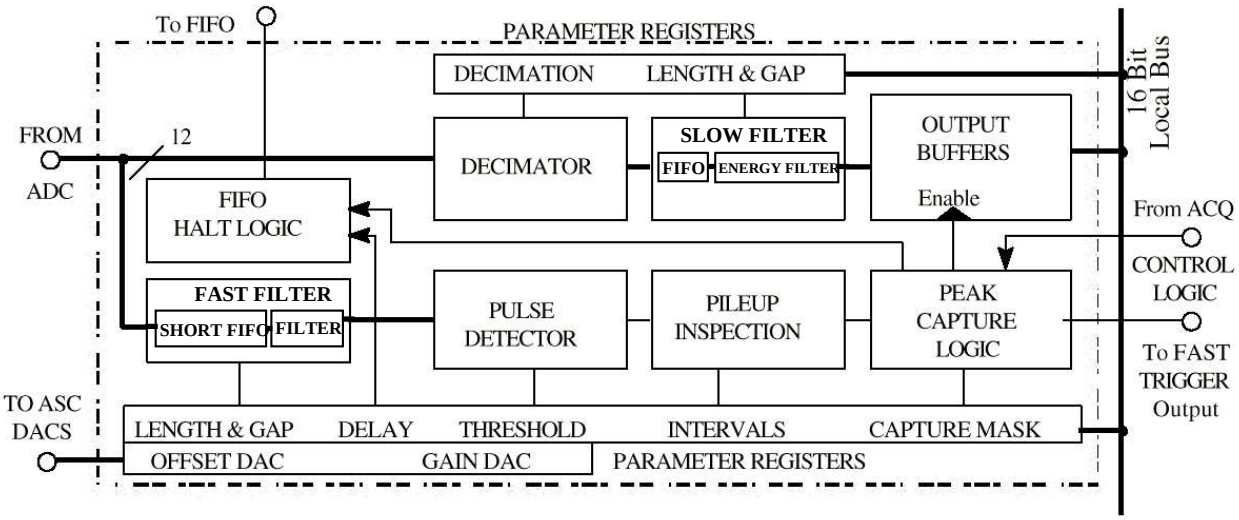

Abbildung 4.5: Detailzeichnung des FiPPI (implementiert in FPGA-Technologie)[18] . Der ADC-Datenstrom wird aufgespalten in den schnellen und langsamen Zweig. Der schnelle Zweig detektiert Ereignisse, überprüft auf Pileup und sorgt für die Auslese des Filterwertes zum gewünschten Zeitpunkt. Der langsame Kanal verbessert durch Mittelwertbildung das SNR der Energiemessung. Ihre Parameter erhalten die Baugruppen aus, durch den DSP programmierten, Registern.

4.2 Funktionsweise

Das Modul DGF-4C besteht, wie oben erwähnt, aus vier unabhängigen Kanälen. Bei einer Samplingfrequenz von 40 MHz ergibt sich bei einer ADC-Auflösung von 12 Bit ein Datenstrom von: $4 \cdot 40$ MHz $\cdot$ 12 Bit $= 4 \cdot 60$ Mbyte/s $= 240$ Mbyte/s. Aus diesen Daten gilt es die interessanten Ereignisse herauszufiltern. Das Prinzip ist in Abbildung 4.4 gezeigt. Der ADC-Datenstrom eines Kanals wird durch den FiPPI analysiert. Erfüllt ein Ereignis die Triggerbedingung und gab es kein Pileup, so wird ein Energiewert extrahiert. Die Datenreduktion erfolgt dabei auf zweierlei Weise: Erstens reduziert der FPGA die Häufigkeit der Aktionen des DSPs auf die Ereignisrate und zweitens die Menge an auszulesenden Daten auf die Filterwerte. Für höchste Eventraten genügen die Daten des FPGA-Filters ("fast list mode"). Nur wenn eine hohe Auflösung oder sonstige Pulsformanalysen (Energieinformation) benötigt werden, kann der DSP nachträglich die FIFO-Daten jedes Kanals auslesen und analysieren.

Die folgende Ausführungen sind vereinfachende Beschreibungen der Ablaufdiagramme aus dem Anhang. Normalerweise befindet sich der DSP im "Leerlauf". Seine Aufgabe besteht in der Extraktion von Baselinewerten und der regelmäßigen Kontrolle, ob im CAMAC **C**ontrol and **S**tatus **R**egister (CSR) das Start-Bit gesetzt wurde. Ist dies der Fall, so startet der DSP einen Run. Zuerst werden Buffer und Variablen initialisiert bzw. berechnet, wie z.B die Spektrenlänge und der Energie Cutoff, die Anzahl der ausgewählten Kanäle oder es wird dafür gesorgt, dass bestimmte Parameter negativ sind. Danach springt der Code in die eigentliche Eventschleife. In der Eventscheife angekommen, schaut der DSP nach, ob sich ein Event im Eingangsbuffer befindet und ob sich überhaupt noch ein freier Platz für die neuen Daten im Ausgabebuffer findet. Ist dies der Fall, so werden die Filterkorrekturen und Pulsformanalysen ausgeführt und die Energiewerte in die Histogramme geschrieben. Sind alle Kanäle abgearbeitet, wird der Eingangsbuffer auf neue zu verarbeitende Daten hin überprüft. Sind keine neue Daten angekommen, so wird der Run beendet. Der Leser wird sich vielleicht schon gewundert haben, wie es möglich ist, dass Daten vom DSP unbemerkt in den Eingangsbuffer gelangen können. Der DSP kann sich in zwei verschiedenen Zuständen befinden: Normaler und Interrupt Modus. Ist der FiPPI bereit zur Auslese, signalisiert er dies dem DSP durch ein Interruptsignal. Dadurch unterbricht der DSP sein laufendes Programm und springt in die passende Interrupt-Routine. Alle Berechnungen werden mit dem zweiten Registersatz ausgeführt. Ist die Interrupt-Routine beendet, fährt der DSP mit seinem ursprünglichen Programm fort. Das Hauptprogramm hat also keine Möglichkeit festzustellen, ob es während seiner Ausführung einen Interrupt gab, ausser über das Setzen von Variablen/Flags oder eben einer von Null verschiedener Differenz zwischen Lese- und Schreibadresse im Eingangsbuffer.

Das Auslesen der FiPPI- und FIFO-Daten im Interrupt Modus und die Verarbeitung der Bufferdaten im normalen Betrieb, sorgt für eine Entkopplung zwischen FPGA und DSP. Beide können relativ unabhängig voneinander arbeiten. Allerdings ist es dem DSP in der Zeit zwischen Interruptsignal und Auslese der FPGA-Daten durch den DSP (laut XIA) nicht erlaubt neue Events zu akzeptieren, weshalb dieser Zeitraum nicht in der LIVETIME des FPGAs enthalten ist. Dies ist vergleichbar mit dem bei Prozesoren üblichen Pipelining. Die Verarbeitungszeit eines Events bleibt unverändert, aber der Durchsatz wird erhöht.

[3] Positive Emitter Kollektor Logik

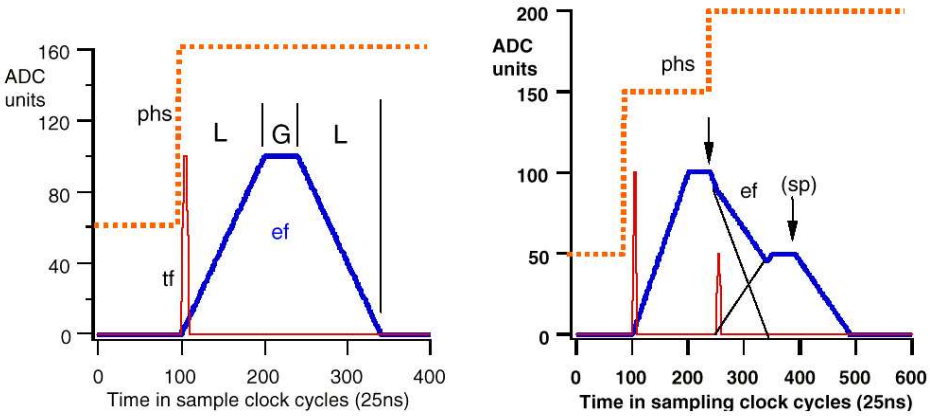

Abbildung 4.6: Minimaler Abstand zwischen zwei Ereignissen [18]. Im linken Bild ist die definierte Länge des FIR-Filters vorgeführt. Für den Energiefilter bedeutet dies beispielsweise, dass nach SL+SG+SL ist der Ausgang des Filter auf Null zurückgeht. Dies gilt exakt natürlich nur in diesem idealisiertem Beispiel. Fallen die Signale exponentiell ab, mißt der Filter im Anschluß an ein Ereignis die Steigung des Pulses, d.h. die Steilheit des Abfalls, was wiederum zur Korrektur der Filterwerte verwendet werden kann. Infolgedessen kann sich, wie auf der rechten Seite vorgeführt, ein neues Ereignis im Prinzip schon direkt nach der Auslese des Slow Filter Wertes ereignen. In diesem Fall sind beide Filterwerte gültig. Hätte sich der zweite Puls zum Zeitpunkt des Beginns der flachen Spitze des ersten Filterpeaks ereignet, so würde der erste Puls verworfen, aber der zweite Filterwert akzeptiert, da die Antwort des Filters auf den ersten Peak wieder den Nullpunkt erreicht hat, wenn der Filterwert des zweiten Ereignisses ausgelesen wird. Mit `tf` wird der Trigger Filter, mit `ef` der Energie Filter, `sp` der Zeitpunkt der Energiefilterauslese und mit `phs` der die Stufe des Eingangssignals bezeichnet.

Damit dies funktioniert muss es der Karte erlaubt sein innerhalb eines Runs (der durch setzten des Bit 0 im CAMAC CSR gestartet wird) mehrere Events aufzunehmen. Dies wird über die Variable MAXEVENTS gesteuert.

In der Detailzeichnung des FiPPIs – Abbildung 4.5 – sind die Register eingezeichnet, die ihre Werte durch den DSP einprogrammiert bekommen. Für jedes Register existiert deshalb eine korrespondierende Variable im DSP Speicher. Der Signalfluß durch die beiden Filter wird in den nächsten Abschnitten behandelt.

4.2.1 Digitale Filter

Der Eventtrigger und die Energiemessung sind durch digitale Filter realisiert worden. Es handelt sich in beiden Fällen um FIR-Filter, deren Koeffizienten die Werte -1,0,1 annehmen. Somit genügen zu ihrer Berechnung die Operationen Addition und Subtraktion. Sie gehorchen folgender Formel:

$$y(k) = -\sum_{i=k-2\cdot L-G+1}^{k-L-G} x(i) \quad + \quad \sum_{i=k-L+1}^{k} x(i) \quad . \tag{4.1}$$

Der Filter berechnet die Differenz zwischen L Werten, die jeweils den Abstand $L + G$ haben und summiert diese auf. Wie in obiger Formel erkennbar, läßt sich der aktuelle Filterwert durch folgende rekursive Formel (die vermutlich implementiert ist) aus den vorangegangenen Werten errechnen:

$$y(k) = y(k-1) - x(k-L) + x(k-2\cdot L-G) + x(k) - x(k-L-G) \quad . \tag{4.2}$$

Durch den 32 Samples langen FIFO (wohl eher ein Schieberegister) im FPGA, beschränkt sich die Filterlänge auf 31 Samples. Die Stufenantwort eines solchen Filters ist in Abbildung 4.6 gezeigt. Ein Trapez mit ansteigenden/abfallenden Kanten der Länge L und einem gleichbleibenden Abschnitt der Länge G.

Ereignisauswahl: kurzer (schneller) Filter

Der schneller Filter (FAST FILTER in Abbildung 4.5) dient zur Detektion eines Ereignisses im Germaniumdetektor. Überschreitet der Filterwert eine einstellbare Grenze für einen einstellbaren Minimalzeitraum (Pulse die kürzer als MINWIDTH=FL+FG (F von FAST, L und G wie oben) sind, werden als noise spikes angesehen und verworfen), so wird ein Eventtrigger erzeugt. Folglich läßt sich durch die Schwellehöhe eine untere Schranke auf die detektierbare Energie setzen. Der Filter arbeitet mit den Daten eines 32 Samples langen FIFOs, der seine Daten direkt vom ADC erhält. Aus der rekursiven Form leitet sich daher die Beschränkung FL+FG < 32 ab. Eine weitere Beschränkung liegt darin, dass FL < 5 gelten muss. Der Ausgang des schnellen Filters wird von dem "Pulse Detector" auf Schwellenübergänge hin überwacht. Verbleibt der schnelle Filter länger als eine, mittels der Variablen MAXWIDTH=FL+FG+Signalanstiegszeit, eingestellte Zeit über der Schwelle, so wird dies als Pileup im schnellen Kanal gedeutet und die Ereignisse verworfen. Akzeptierte Ereignisse werden von der darauffolgenden "Pileup Inspection"-Einheit auf ausreichende Separation geprüft. Die letzte Einheit, die "Peak Capture Logic" sorgt dafür, dass zum passenden Zeitpunkt (PEAKSAMPLE) – sofern kein Pileup, sowohl im schnellen (MAXWIDTH) als auch im langsamen Kanal (PEAKSEP), vorliegt – der Wert des Energiefilters in das Ausgaberegister (Outputbuffer) übernommen wird. Der schnelle Filter arbeitet, im Gegensatz zum Energiefilter, immer mit einer Zeitauflösung von 25 ns.

Energiemessung: langer (langsamer) Filter

Wird ein aufintegrierender Vorverstärker benutzt (reset amplifier), so leidet ein Trapezfilter nach Definition 3.2 nicht unter einem ballistischem Defizit, da er den ansteigenden Anteil des Pulses durch seine Lücke nicht miteinbezieht. Er misst den gleichen Wert, den er bei einem unendlich schnell ansteigenden Signal messen würde. Würde man den Anstieg miteinbeziehen, so wird nicht die volle Höhe gemessen. Setzt man die Lücke in Gedanken Null, so kann man die Größe des ballistischen Defizits etwa $\frac{Anstiegszeit}{Filterlänge}$ abschätzen, da durch den Pulsanstieg der Filter Signale aufsummiert, die noch nicht die volle Höhe erreicht haben. Werden stattdessen nun "resitiv feedback" Vorverstärker benutzt, so leidet der Trapezfilter unter ballistischen Defizit aufgrund der Tatsache, dass der Puls, durch den schon während der Sammelzeit erfolgenden exponentiellen Abfalls des Vorverstärkers, nicht seine maximale Höhe erreichen wird. Infolgedessen misst der Energiefilter (SLOW FILTER

in Abbildung 4.5) eine defizitäre Höhe, die durch Pulsformanalyse des ansteigenden Teils des Signals korrigiert wird. Auch hier ist nicht ein konstanter Verlust an Amplitude von Bedeutung, sondern die Variation der Amplitude, die durch die unterschiedlichen Anstiegszeiten des Detektorsignals verursacht wird und die ihrerseits wieder von der Position der Wechselwirkungen im Detektor abhängig sind.

In [49] ist ein Performance-Index, der den Vergleich der Pileup-Performance verschiedener Filter-Typen erlaubt, angegeben. Die "Resolving Time" λ ist definiert als, die Zeit nach der ein weiteres Ereignis erfolgen kann, ohne die Messung des vorhergehenden Events zu beeinträchtigen. Der Ausgang eines gaussförmigen Filters kann nach etwa $\lambda = 3\,\sigma$ vernachlässigt werden. Vergleicht man dies mit einem Trapezfilter dessen Lücke G halb so lang wie seine Anstiegszeit L ist, so folgt , dass bereits nach $\lambda = 1.5\,L$ ein weiterer Puls erfolgen kann. Gilt nun $L = \sigma$ so ergibt dies eine Verbesserung von einem Faktor zwei gegenüber einen Gauss-Shaper. In [49] ist ebenfalls ein Vergleich im Delta Noise Verhalten aufgeführt. Bei gleicher Resolving Time ($\lambda_{gauss} = \lambda_{trapez}$) unterbietet der Trapezfilter den Gaussfilter um den Faktor 1.2.

Das Modul DGF-4C erlaubt in vier Stufen die Einstellung verschiedener Filterlängen. Diese vier Stufen, die DECIMATION (kurz: DEC), sind durch die entsprechende Programmierung des FiPPIs festgelegt. Zur Zeit gibt es Programmierinformation (FPGA Code) für die DEC-Werte 0, 2, 4 und 6. Der Energiefilter arbeitet in allen Varianten mit einem FIFO der festen Länge 32, womit die Programmierung der Filterlänge einheitlich bleiben kann. Statt den FIFO zu verlängern, vergrößert man stattdessen die Wortbreite von 12 auf 12 + DEC Bits, wobei nun die Summe von 2^{DEC} Samplewerten im FIFO steht. Gleichzeitig arbeitet der langsame Kanal nur noch mit einer Zeitauflösung von $25 \cdot 2^{DEC}$ ns. Der DECIMATOR-Einheit in Abbildung 4.5 folgt der $12 + 2^{DEC}$ Bit breite Energiefilter-FIFO. Ignoriert man die unteren DEC Bits, so erhält man den Mittelwert (ohne Rundung). Ein Filter der Länge 6 μs (SL+SG, S wie Slow) beispielsweise, bräuchte 240 Speicherelemente der Wortbreite 12 Bit. Benutzt man eine DECIMATION von 4 so reduziert sich die Anzahl der Speicherelemente auf 15, die nun eine Breite von 16 Bit haben. Durch die DECIMATION reduziert sich entsprechend der Takt der Energiefiltereinheit, da der DECIMATOR alle 2^{DEC} Takte einen neuen Filterwert liefert. Der Ausgang des Energiefilters kann auf einen Output Buffer gegeben werden, der durch den DSP ausgelesen wird.

4.2.2 Pileup und Totzeit

Die Karte leidet unter zwei Arten von Pileup: Pileup im schnellen Kanal und Pileup im Energiefilterkanal. Der Eventtrigger (schneller Kanal) wird zwar durch einen kurzen Filter erzeugt (FL $\sim$ 0,1 μs), allerdings ist auch er von endlicher Länge und somit ebenfalls vom Pileup-Problem betroffen. Zur Identifikation eines Pileups im schnellen Kanal wird dessen Zeitraum des Verbleibs über der Triggerschwelle beobachtet. Überschreitet dieser einen einstellbaren Wert (empfohlen: MAXWIDTH=FL+FG+Signalanstiegzeit), so wird das Ereignis verworfen.

Abbildung 4.7 zeigt die verschiedenen Pileup Möglichkeiten. Kurz nachdem der Energiewert des ersten Events extrahiert wurde, erfolgt ein zweites Event, welches die erste Messung nicht mehr beeinflussen kann. Allerdings folgt ein drittes Event bevor der Filterwert ausgelesen wird. Da der Abstand der Pulse gemessen und überwacht wird, verwirft

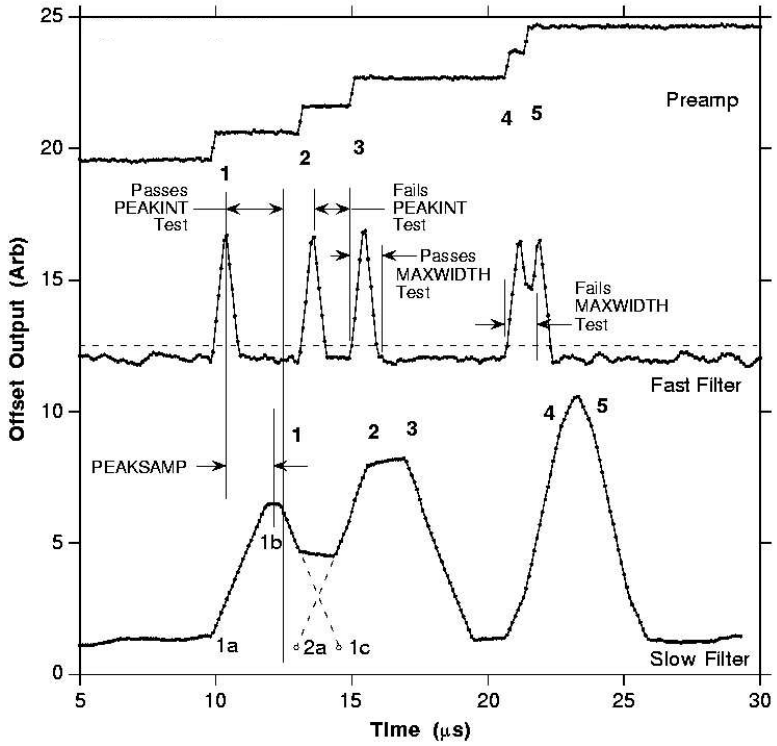

Abbildung 4.7: Pileup Detektion im schnellen Kanal (Eventtrigger) [18]. Im oberen Teil sind die Stufen des Vorverstärkers erkennbar. Jede Stufe wird durch den Fast Trigger erkannt, dessen Reaktion darunter abgebildetet ist. Man erkennt die beiden Arten von Pileup: mangelnde Separation der Fast Filter Impulse und Pileup im Fast Filter selbst. Unten im Bild ist die Reaktion des Slow Filters gezeigt. Nur das erste Event wird akzeptiert.

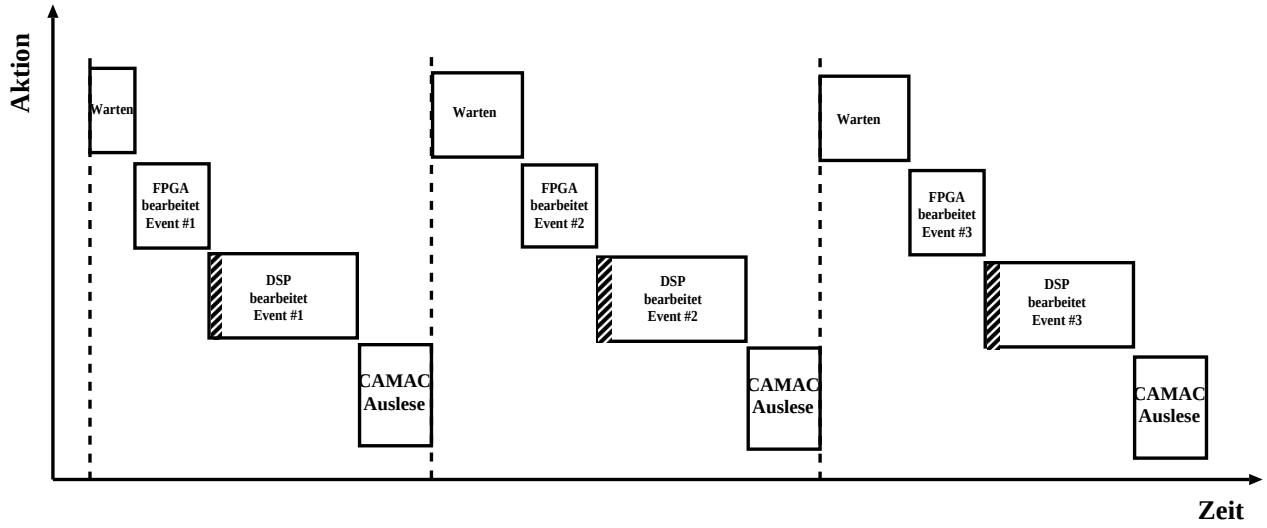

Abbildung 4.8: Verarbeitungsablauf bei MAXEVENTS=1. Die schraffierten Flächen zeigen die Zeit an, in der das FPGA durch den DSP ausgelesen wird. Die erneute Datenaufnahme ist solange blockiert, bis die durch den DSP zu bearbeiteten Daten vollständig ausgelesen wurden.

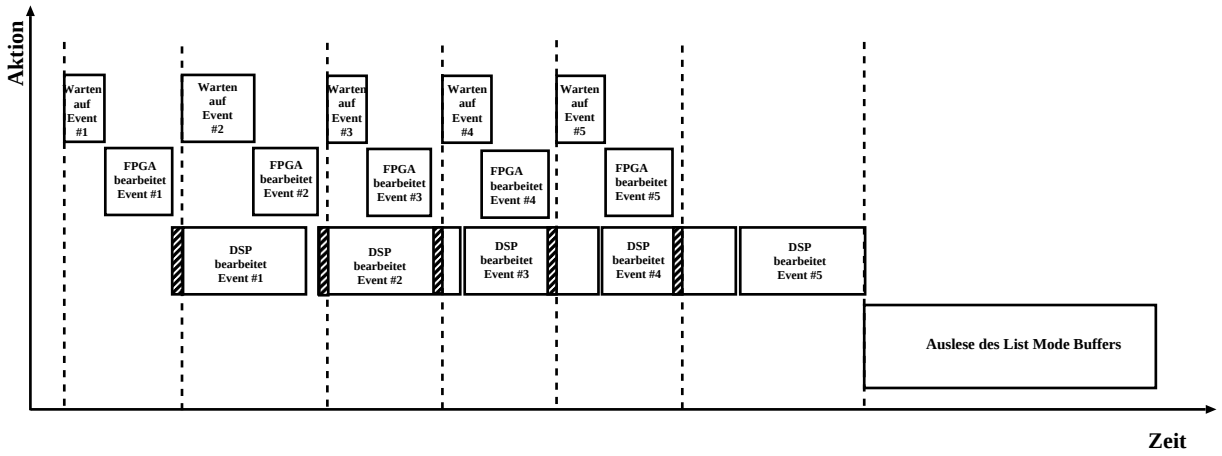

Abbildung 4.9: Verarbeitungsablauf bei MAXEVENTS=5. In diesem Fall, werden fünf Events aufgenommen und in den Buffer der XIA Karte geschrieben. Erst nachdem der DSP Event Nummer 5 bearbeitet hat, startet die Auslese. Im Unterschied zu oben ist hier die Datenaufnahme durch die FPGAs nur während der Auslese durch den DSP blockiert. Während der DSP die Eventdaten bearbeitet, können gleichzeitig neue Events aufgenommen werden.

der Pileup-Inspector dieses Event aufgrund mangelnder Separation. Die letzten beiden Ereignisse erfolgen in so kurzem Abstand, dass es zu Pileup im schnellen Kanal kommt. Dies wird durch Messung des Zeitraum des Verbleibs über der Triggerschwelle festgestellt.

Damit eine Energiemessung ungestört stattfinden kann muss der Filterwert eines vorangehenden Ereignisses vollständig verschwunden sein. Der Energiemessung findet ungefähr zum Zeitpunkt SL+SG nach dem Eventtrigger statt. Man erhält als minimalen Abstand die Filterlänge SL. Innerhalb des Zeitintervalls zwischen Eventtrigger und Energiemessung darf natürlich kein weiteres Ereignis auftreten. Deshalb muss nach hinten ein Abstand von SL+SG eingehalten werden. Man vergleiche mit Abbildung 4.6. Allerdings ist der empfohlene Wert konservativer: Es wird ein Abstand von mindestens SL+SG+2 zwischen zwei Pulsen verlangt. Soweit die Theorie.

Beide Filter sind paralysierbar, d.h. ihre Totzeit verlängert sich durch die Ankunft eines Events innerhalb ihrer Totzeit um ein weiteres Totzeitintervall (ab Zeitpunkt des neuen Events). Ihr Totzeitverhalten ist daher durch Formel 3.13 gegeben, wobei PEAKSEP=SL+SG+2 bzw. MAXWIDTH als Totzeit des langsamen bzw. schnellen Filters angenommen werden kann. Für den schnellen Kanal hängt die Totzeit auch von der Schwellenhöhe ab. Die Totzeit des schnellen Kanals läßt sich demnach nicht einfach berechnen, sondern sollte durch eine Messung bestimmt werden. Da DSP und FPGA parallel arbeiten können, gilt allerdings

$$\text{overall deadtime} \leq \text{FPGA deadtime} + \text{DSP deadtime} \; . \tag{4.3}$$

Für diesen Fall gibts es noch keine allgemeingültige Beschreibung, da sich die Totzeit des DSPs nicht einfach zur Filtertotzeit addieren lässt. Es ist für den FiPPI ohne weiteres möglich (und gewollt) weitere Ereignisse zu akzeptieren, während der DSP noch mit der Auswertung vorangegangener Events beschäftigt ist. Dies sorgt für eine Reduktion der Gesamttotzeit, da die DSP-Tozeit zur Aufnahme und ersten Verarbeitung des nächsten Events durch die FiPPI-Filter genutzt wird. Dies betrifft allerdings nur den Fall, in dem die Karte mehrere Events innerhalb eines Runs aufnehmen darf (MAXEVENTS>1). Die Abbildungen 4.8 und 4.9 stellen die Fälle MAXEVENTS=1 und MAXEVENTS=5 einander gegenüber. Sollte es mit einer zukünftigen Revision möglich sein, bearbeitete Eventdaten während eines Runs auszulesen so wird die DSP Totzeit für die Gesamttotzeit an Bedeutung verlieren.

4.2.3 Die Korrekturen durch den DSP

Im Gegensatz zu den bis hierher gezeigten Beispielen sind die Signale des MINIBALL Vorverstärkers nicht stufenförmig. Sie sind vielmehr durch einen langsamen exponentiellen Abfall gekennzeichnet ($\tau \sim 50\ \mu$s). Aus diesem Grund ist der Ausgang des Energiefilters in Abwesenheit eines Ereignisses nicht Null. Die Stärke des Abfalls ist proportional zur Amplitude des Signals. Aus diesem Grund muss der Energiewert korrigiert werden. Die zur Korrektur benötigten Werte werden, Mithilfe von sechs mitgelieferten Formeln (im Programmer's Manual), vorberechnet und in den Speicher des DSPs geschrieben. Über ihre Funktion kann, aufgrund fehlender Dokumentation, nur spekuliert werden.

In Abbildung 4.10 ist ein idealisierter Puls skizziert. Der Energiefilter misst die Differenz der Summen über SL Werte beginnend ab den Zeitpunkten t_A bzw. t_B. Wird der

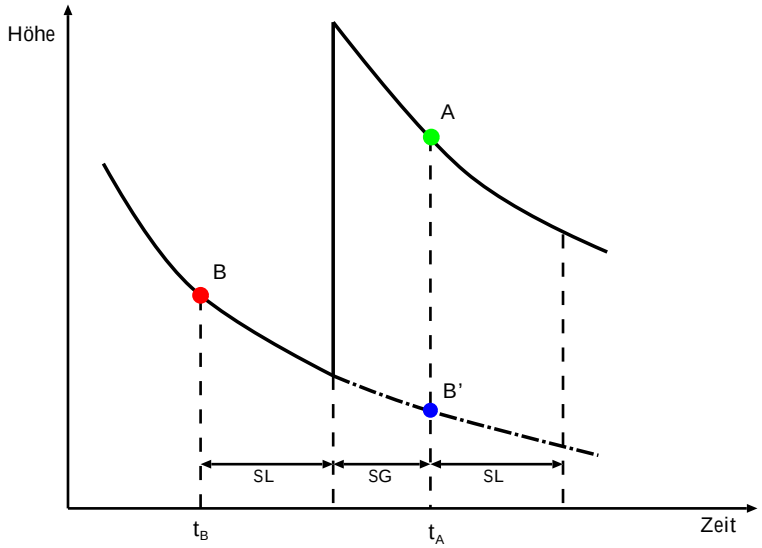

Abbildung 4.10: Skizze eines Vorverstärkerpulses unter Vernachlässigung der Anstiegszeit.

Einfluß der Exponentialfunktion aus dem Filter entfernen (endliche geometrische Reihe), so mißt dieser einen gemittelten Wert der Differenz der Amplituden A und B, wodurch das Signal zu Noise Verhältnis verbessert wird. Die Normierung geschieht durch Division durch die Filteranstiegszeit SL. Durch diese beiden Rechenschritte ist man bei einem normierten Filterwert angelangt, der die Höhendifferenz zwischen den Punkten A und B angibt. Wichtig ist die Einhaltung eines festen zeitlichen Abstandes zwischen Pulsbeginn und Zeitpunkt der Energiemessung (definierte Messzeit), da ansonsten die gemessenen Amplituden nicht eindeutig sind. Für FPGA-Code mit DEC > 0 muss deshalb noch der exakte Eventzeitpunkt eingerechnet werden, da der Energiefilter in diesem Fall mit (der DECIMATION entsprechend) verringerter Taktrate arbeitet.

In Abwesenheit eines Ereignisses misst die Karte nur den Wert der Baseline. Unter der Annahme eines perfekten exponentiellen Abfalls mißt man die selbe Kurve an den, durch $t = SL+SG$ getrennten, Zeitpunkten t_A und t_B und erhält die Werte B und B'. Der Wert der Baseline wird ständig gemessen und auf Wunsch in ein internes Baseline Histogramm geschrieben, welches zu Informationszwecken ausgelesen werden kann. Ausserdem wird der Mittelwert aus mehreren aufeinanderfolgenden Messungen bestimmt. Die Gewichtung einer einzelnen Messung kann über die Variable LOG2BWEIGHT gesteuert werden. Ist diese Null, so trägt die aktuelle Messung nicht bei. Die Formel für die Berechnung des Gewichts ist der Beschreibung der Variablen LOG2BWEIGHT zu entnehmen. Mithilfe einer weiteren Korrekturformel, läßt sich der Wert einer Größe die über einen Zeitraum von $SL + SG$ exponentiell abfällt berechnen. Damit ließe sich beispielsweise bei bekanntem B der Wert von B' berechnen, der im Falle eines Events nützlich sein könnte. Desweiteren hat der DSP im Prinzip Zugriff auf den aktuellen ADC Wert, da dieser Wert in ein Ausgaberegister des FPGAs geschrieben wird. Dieses Register wird benutzt, Pulse (ungetriggert) über einen längeren Zeitraum (TRACELENGTH $>$ FIFO-Länge, aber $\leq$ DSP Bufferlänge) aufzunehmen.

Durch Pulsformanalyse des durch den Energiefilter ausgelassenen Bereichs, kann die

Energieauflösung für Eventraten $\leq$ 20 kHz verbessert werden. Zu diesem Zweck liest der DSP die Pulsdaten aus den FIFOs. Der Beginn der Trace wird über die Variable USERDELAY gesteuert. Durch die Variablen PSAOFFSET und PSALENGTH wird der Startpunkt PSAOFFSET, nach Beginn der Trace festgelegt, ab dessen Position die in PSALENGTH angegebene Anzahl von Samples analysiert wird. Das erste Viertel dient zur Berechnung der Fläche unter der Baseline. Dieser Wert mit, 4 multipliziert, wird von der Summe über alle PSALENGTH Samples angezogen. Damit ergibt sich die Fläche unter dem Ladungspuls, die mit $\frac{1}{\tau}$ multipliziert, zur Korrektur des ballistischen Defizits verwendet werden kann.

Zwei Arten von CFD-Trigger sind im Modul DGF-4C implementiert: ein Hardware-CFD Trigger als Ersatz für den LE-Trigger (beide im FiPPI) und ein Software-CFD Algorithmus, ausgeführt durch den DSP. Während die Funktionsweise des Hardware CFD-Triggers von XIA beschrieben und getestet wurde, ist dies für den Software CFD-Trigger nicht der Fall. Seine Funktionsweise soll im folgenden erläutert werden. Zuerst wird die Höhe des Pulses bestimmt. Diese Höhe wird mit der Software-CFD Schwelle multipliziert, die demnach eine Zahl zwischen Null und Eins sein muss. Nun werden berechnete Pulshöhe und ADC-Wert solange verglichen bis der Puls diese Schwelle überschritten hat (t_0). Dann wird die Steigung aus der Differenz der ADC-Werten direkt unter $x(t_0 - 1)$ und über $x(t_0)$ der Schwelle bestimmt, sowie die Differenz zwischen Schwelle und dem ADC-Wert direkt darüber $x(t_0)$. Dann wird folgende Gleichung durch Division gelöst:

$$t_{\mathrm{CFD}} = \frac{x(t_0) - Schwelle}{x(t_0 - 1) - x(t_0)} \quad . \tag{4.4}$$

Sie liefert den Wert, um den die Position t_0 des ADC-Wertes direkt über der Schwelle erniedrigt werden muss. Somit erhält man die interpolierte Postion an der der Puls die CFD-Schwelle überschreitet.

4.3 Programmierung

4.3.1 Steuerung der Funktionalität

Für die Ansteuerung der Karte konnte auf vorhandene C-Routinen von Dirk Weisshaar vom IKP in Köln [12] zurückgegriffen werden. Diese Routinen bilden die Basis für ein C-Programm mit dessen Hilfe ein Parameterfile erstellt werden kann. Dieses Parameterfile wird von der eigentlichen Datenaufnahme eingelesen und in den DSP geschrieben. Alternativ kann auf ein mitgeliefertes Programm der Firma XIA zurückgegriffen werden, welches ebenfalls die Einstellung und Sicherung der Parameter erlaubt. Allerdings läuft dieses Programm nur auf Windows und Macintosh Platformen und unterstützt nur eine Sorte von CAMAC-Controllern. Gesteuert wird die Karte über Variablen, deren Werte sich innerhalb eines festen Speicherbereichs des DSPs befinden. Die Variablen werden eingeteilt in Eingabe- und Ausgabevariablen sowie nach ihrer Zugehörigkeit zu einen bestimmten Kanal bzw. ihrer Gültigkeit für das ganze Modul. Mithilfe der im Programmer's Manual angegebenen Formeln lassen sich für die meisten Variablen sinnvolle Werte berechen. Zur Bestimmung des Vorverstärker Offsets sind zwei Selbsttests implementiert, die einmal den Offset in Abhängigkeit der zugehörigen Variable messen und zweitens die ADC-Werte

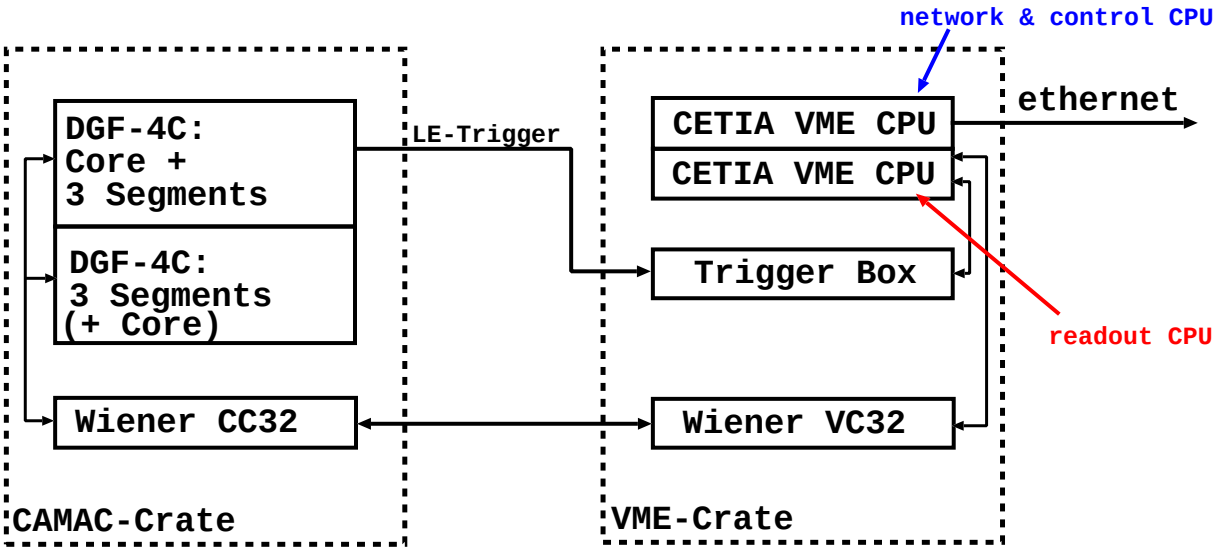

Abbildung 4.11: Prinzipieller Aufbau des DGF-4C Teststandes

ohne Event-Trigger auslesen. Weitere CONTROLTASKS dienen zur Programmierung der DACs[4] zur Einstellung der Verstärkung und des Offsets und der Programmierung der FPGAs. Insgesamt gibts es anscheinend 14 Controltasks. Allerdings sind davon nur vier dokumentiert worden.

4.3.2 Datenaufnahme

Der derzeitige Testaufbau, der für die vorliegende Arbeit aufgebaut wurde und der in Abbildung 4.11 schematisch gezeigt ist, besteht aus zwei VME CPUs, einer Triggerkarte, dem Wiener fast CAMAC-Controller CC32 [50] plus zugehöriger VME-Adapterkarte VC32 und zwei XIA DGF-4C Modulen (man Vergleiche mit Abbildung 4.11). Da die DGF-4C Karten den fast CAMAC Standard [51] unterstützen, sollte der dadurch entstehende Geschwindigkeitsvorteil nicht ungenutzt bleiben. Alle Zugriffe, mit Ausnahme der Steuerbefehle (Slow Control), machen von fast CAMAC gebrauch. Ein Triggersignal des DGF-4C Moduls wird auf eine Triggerbox gegeben, die Datenaufnahme bemerkt dies und beginnt mit der Auslese der Eventpufferspeicher nachdem das Run Finish Bit im CAMAC CSR der DGF-4C Karten (zu früh) gesetzt wurde. Die Wahl des Triggers ist unbedeutend, da sowohl FPGA- als auch DSP-Trigger bereits vor der vollständigen Verarbeitung eines Events stattfinden (sollen). In den Statusregistern der aktuellen Version gibt es kein Bit, welches das Ende eines Runs bzw. einen Buffer wirklich als auslesefertig kennzeichnet.

Für die Testmessungen konnte auf die vorhandene Datenaufnahme am MPI-K [52] zurückgegriffen werden. Die restliche Arbeit bestand darin, an geeigneter Stelle in dem Modul `vmesil.c` die Unterprogramme für die XIA Karten aufzurufen. Für die Testmessungen wurde ausserdem die Onlineanalyse an die Erfordernisse angepasst. Vom Eventbuffer der XIA Karten wird noch kein Gebrauch gemacht, da eine in naher Zukunft verfügbare Auslese per FireWire-Interface [53] zusammen mit einer zwei Puffer Architektur ohne größere Modifikationen mit der derzeitigen DAQ-Architektur betrieben werden kann. Die-

[4] **D**igital **A**nalog **C**onverter

ser FIFO-Modus der Datenaufnahme entspricht in gewisser Weise dem Pipelined Modus der XIA-Karten, da in diesem Fall zuerst die Datenaufnahme eines neuen Events gestartet wird und im Anchluß daran, die Auslese des vorangegangenen Events beginnt. Dazu wird allerdings ein gesplitteter Pufferspeicher benötigt, den die DGF-4C Module noch nicht haben. Wird der Synch-Eingang der DGF-4C Module auf Eins gehalten, beispielsweise kann ein zusatzlicher Ausgang der Triggerbox auf das ODER der Module gelegt werden, so startet die Datenaufnahme durch die Karten nicht, bevor die Datenaufnahmesoftware den Ausgang der Triggerbox auf Null setzt. Damit ist der Zeitnullpunkt festgelegt. Der Zeitpunkt eines Trigger (beliebige Detektoren und Elektronik) kann somit (in Software) durch die Datenaufnahme mittels der Triggerbox gemessen werden. Wird die XIA Elektronik nach MAXEVENTS Ereignissen ausgelesen, so müssen die Modulzeiten einfach der Reihe nach durch die von der Datenaufnahme gemessen XIA-Trigger Zeitpunkte ersetzt werden. Dieser Vorgang könnte auch durch ein noch zu entwickelndes Hardware-Modul ausgeführt werden, das mit beliebig vielen Triggersignalen umgehen kann. Problematisch wird es, wenn eine lokale Zeitmessung durch die XIA Module gegenüber einer globalen Zeitmessung durch die Datenaufnahme bevorzugt wird. Die nach einem Run aus dem Buffer der XIA Module ausgelesenen Daten und Eventzeiten müssen den Daten anderer Detektoren, die nicht die XIA Elekronik benutzen und deshalb keine eigene Eventzeit haben, zugeordnet werden. Benutzt das Experiment nur die XIA Elektronik so sollte es keine Probleme geben, denn zu diesem Zweck wurde die interne Zeitmessung entwickelt. Wie die Uhren der XIA Module mit externen Uhren, eventuell über künstlich erzeugte Events, zu synchronisieren sind ist noch nicht bekannt.

Nichtsdestotrotz ist die Funktionalität auch durch eine – konservative – Event-by-Event Auslese überprüfbar. Zu diesem Zweck können Daten in zwei verschiedenen Modi aufgenommen werden: list mode (RUNTASK=1) und fast list mode (RUNTASK=2).

Da der fast list mode (RUNMODE 3 in Tabelle 4.1) zur Erreichung eines maximalen Durchsatzes die FIFO Daten nicht ausliest, ist keine Pulsformanalyse durchführbar, weshalb dieser Modus im Moment keine Verwendung findet. Der list mode (RUNMODE 0-2) erlaubt dagegen die Auslese der FIFO Daten. Zur Offline Analyse der Pulsformen ist es durch Löschen des Bits 0 in MODCSRA möglich, die Pulsformen in den List Mode Buffer schreiben zu lassen, sie folgen dann direkt auf den Channelheader, dessen Länge um die Tracelänge erhöht wird. Ist Bit 0 in MODCSRA gesetzt, so werden die Pulsformen zur Analyse in den Event Buffer geschrieben. Allerdings werden nur die Analyseresultate (in die Header der Kanäle) und nicht die Pulsformen in den List Mode Buffer geschrieben.

Ein Problem liegt im (in der aktuellen DSP-Code Version) fest vorgegebenen Format des Eventbuffers, da dieses nicht die minimal mögliche Anzahl an Information – soweit es MINIBALL betrifft – enthält, vielmehr wurde der Header um zwei derzeit ungenutzte Datenwörter verlängert. In Zukunft sollte man sich unbedingt auf ein festes MINIBALL Format einigen. Damit ist auch eine feste Tracelänge gemeint, da man sich in diesem Fall die Längenangaben (die nicht der wirklichen Länge entsprechen, sondern schlicht umkopierte Variablenwerte sind) und die Formatinformationen sparen kann. Aus Simulationen [11] ist schon lange bekannt über welchen Zeitraum ein Puls des MINIBALL-Detektors interessante Informationen enthält. Für jedes zusätzliche Datenwort sollte es eine ausreichende Rechtfertigung geben.

Tabelle 4.2 zeigt den Aufbau des Setup-Files. Aus diesem File liest die Datenaufnah-

RUNMODE	Subevent	RUNTASK	MODCSRA Bit 0	Format	Bemerkung
0	666	1	gesetzt	variabel	einzelne Werte
1	667	1	ungesetzt	XIA	
2	668	1	gesetzt	XIA	Auslese der Header
3	669	2	gesetzt	variabel	keine FIFO Daten

Tabelle 4.1: Datenaufnahmemodi.

Position	Belegung
1	RUNMODE
2	ANZAHL DER XIA-KARTEN
3	PFAD FPGA-CODE
4	PFAD DSP-CODE
5	PFAD DSP-VARIABLEN-FILE

Tabelle 4.2: Derzeitiges Format der Initialisierungsfiles DGFSETUP.

me die Informationen wo der DSP- und FPGA-Code, sowie das Parameterfile zu finden sind, wieviel Karten gemeinsam betrieben werden sollen und letztendlich welcher Runmode gewählt wird. Derzeit gibt es deren vier. Sie sind in Tabelle 4.1, zusammen mit ihren zugehörigen Subeventnummer durch die sie durch die MPI-Datenaufnahmen identifiziert werden, aufgelistet. Darüber hinaus wurden noch die in Tabelle 4.3 aufgelisteten Subeventtypen eingeführt. Sie dienen zur Aufnahme des On-Board Spektren-Speichers, des Baseline Histogramms sowie einer von Zeit zu Zeit gewünschten Sicherung der Parameter.

4.3.3 Implementierung eigener Algorithmen

Zur Implementierung eigener Algorithmen wurde von der Firma XIA ein Interface (Version vom 11.05.00) bereitgestellt, welches aus dem vorcompiliertem XIA-Code (alle von XIA gelieferten Assemblermodule) dem Assembler Modul USER(.dsp) mit eigens dafür reserviertem Speicherbereich von 2048 Instruktionen (PM) und 1000 Datenwörtern, je 16 Ein- und Ausgabevariablen zur Kommunikation zwischen Host und User Code und eines Rückgabewertes (`UretVal`), der in die Kanalheader des Ausgabepuffers eingebunden wird, besteht. Der Assembler liefert ein Object (.OBJ), ein Code (.CDE), ein List (.LST) und ein Initialization (.INT) File.

Da alle Files (außer .LST) zum Linken benötigt werden, liegen alle XIA Module in diesen vier Varianten vor. Da im .LST File (zu Debug-Zwecken) der Quellcode enthalten ist, liegt dieses File natürlich nicht bei.

Eine Übersicht der Belegung der Ein- und Ausgabevariablen (USERIN und USE-

Subevent	Inhalt	Bemerkung
111	Modulparameter	416 Variablen pro Modul
222	On Board MCA Speicher Inhalt	8k Programmspeicher pro Modul
333	On Board Baseline Spektrum	2k Programmspeicher pro Modul

Tabelle 4.3: Weitere Subeventtypen.

```
************************************************************************
*   INTRFACE.INC
*   External declarations that give user code access to main code
*   variables.  This file is to be provided by XIA.
*
************************************************************************

.EXTERNAL  UserIn;        { 16 input data words }
.EXTERNAL  UserOut;       { 16 output data words }

.EXTERNAL  ATstart;       {  Address of current trace }
.EXTERNAL  TLen;          {  Length of current trace }
.EXTERNAL  Energy;        {  energy of current event }
.EXTERNAL  ChanNum;       {  Number of current channel }
.EXTERNAL  URetVal;       {  Return value from user routine }
```

Abbildung 4.12: Das Intrface.inc File

ROUT) befindet sich im Anhang. Der Zugriff auf XIA Code Variablen ist (war) auf die Startadresse und Länge der Trace (`ATstart, TLength`), die Kanalnummer (`ChanNum`), die Energie (`Energy`) und Rückgabewert (`URETVAL`) beschränkt. Durch Modifikation des Files `Intrface.int` (siehe Abbildung 4.12: `is to be provided by XIA`) lassen sich allerdings globale XIA Code Variablen hinzufügen.

Ebenfalls mitgeliefert wurden ein Makefile zur Erstellung einer Binär (`dspcode.bin`) und einer ASCII Version (`dspcode.exe`), das sog. Memory Image, des DSP-Codes. Das Makefile `makefile` wird durch das Programm `nmake.exe` ausgeführt und ist in Abbildung 4.13 gezeigt. Das Programm `mkd.exe` erzeugt das Binary File (`.bin`) und das Parameterfile `dgfcode.par`. Die Linker (`ld21`) Option `-i link` führt dazu, dass alle im File `link`, welches in Abbildung 4.14 gezeigt ist, angegebenen Objekte mit eingebunden werden. Die Option `-a dgf` verweist auf die Architekturbeschreibung 4.15, die Option `-e dgfcode` sorgt dafür, dass das Ausgabe File den Namen `dgfcode.exe` erhält (ansonsten `210X.exe`). Die Optionen `-g` und `-x` sorgen dafür, dass ein `.SYM` und ein `.MAP` File erstellt werden. Das `.MAP` File listet die Aufteilung des DSP Speichers auf ("dgfcode (dgfcode.exe) mapped according to DGF (dgf.ach)") und lässt ausserdem erkennen, dass die Anordnung des Speiches von XIA fest vorgegeben ist ("fixed program memory map"). Der Assembler (`asm21`) wird mit einer expliziten Angabe der Architektur (und damit des Befehlssatzes) aufgerufen: `-2181`. Eine Variable `ARCH` wurde gesetzt aber nicht benutzt. Wird die Endung des Filenamens des zu kompilierenden Codes nicht explizit angegeben (user, double), so wird automatisch die Endung `.dsp` angenommen.

Zur Kompilierung standen die ADSP-2100 Family Development Tools, Release 6.1 zur Verfügung. Programmiert wurde in Assembler. Zusätzlich wurde – aus reiner Neugier – die Fliesskommabibliothek von Analog Devices eingebunden. Allerdings machen weder die aktuelle XIA Code Version (im Gegensatz zu früheren Versionen) noch der User Code davon Gebrauch, sodass dies in Zukunft nicht mehr nötig ist und aus dem Makefile entfernt werden kann, sofern die dadurch gewonnene numerische Präzision nicht benötigt wird.

Da die User Routinen vom laufenden XIA Code aufgerufen werden, war der Gebrauch einiger Register verschiedenen Restriktionen unterworfen. Da beispielsweise der zweite Registersatz zur Interruptbehandlung benutzt wird, stand er nicht zur Verfügung. Ebenfalls eingeschränkt war der Gebrauch des ersten Registersatzes. Die Adressregister i5, i6 und i7

```
OPTIONS = -DDGF4C -DNCHANNELS=4
ARCH = -2181

dgfcode.bin :  dgfcode.exe
  mkd dgfcode
dgfcode.exe :  baseline.obj begin.obj double.obj interupt.obj \
               main.obj multipar.obj HardWare.obj  process.obj \
               testseg.obj user.obj
  ld21 -i link -a dgf -e dgfcode -g -x
double.obj : double.dsp
  asm21 double -2181
user.obj : user.dsp intrface.inc
  asm21 user -2181
```

Abbildung 4.13: Das makefile File

```
baseline
begin
double
hardware
interupt
main
multipar
process
testseg
user
```

Abbildung 4.14: Das link File

werden beispielsweise benutzt, um festzustellen, ob die Interruptroutine neue Daten in den Eingangspuffer geschrieben hat (Schreibadressse != Leseadresse). Diese Einschränkungen sind abhängig von der Position im User Code. Die genaue Beschreibung ist als Kommentar dem Interface durch die Firma XIA beigefügt worden.

Das Architekturfile (.ACH) File ist in Abbildung 4.15 gezeigt (Da leider das .SYS File nicht mitgeliefert wurde, muss vom .ACH auf das .SYS zurückgeschlossen werden). Der Name der Architekturbeschreibung ist in Zeile 1, die Zielarchitektur (der Prozessor) ist in Zeile 2 angegeben. Die Anweisung in Zeile 3 zeigt an, dass "boot loading" erlaubt ist, d.h. durch einen Reset wird das Laden des Codes beginnend mit Adresse 0x0000 ausgelöst. Mit den restlichen Anweisungen lässt sich auf die Aufteilung des DSP Speicher zurückschliessen. Da Programmspeicher sowohl Daten als auch Instruktionen speichern kann, tauchen hier zwei Möglichkeiten auf: pa**m** und pa**d**. Ausserdem muss noch angegeben werden, ob es sich um RAM oder ROM Speicher handelt. Dies zeigt der Buchstabe `a` für RAM in `pam` bzw.

```
1  DGF
2  ADSP2181
3  MMAP0
4  0000 0FFF pamPROGRAM t
5  1000 17FF pamUSERPROGRAM t
6  1800 3FFF padPMDATA t
7  0000 019F dadDGF_PARAMS t
8  01A0 3BF7 dadDMDATA t
9  3BF8 3FDF dadUSERDATA t
```

Abbildung 4.15: Das .ACH File

`pad` an.

Die Implementierung der R und Φ Algorithmen

Im folgenden wird nun der Aufbau des Moduls `User.dsp` (der vollständige Code befindet sich im Anhang) beschrieben, wobei gleichzeitig auf die Umsetzung der Algorithmen eingegangen wird. Damit der User Code ausgeführt wird muss Bit 0 im MODCSRB und für jeden benötigten Kanal Bit 0 im CHANCSRB gesetzt sein.

Das Modul USER (`.MODUL` Anweisung) beginnt mit der Dekalaration einiger Variablen und Buffer (Anweisung:`.VAR/DM/SEG=UserData`) und der Einbindung der `Intrface.INC` Datei mittels der `.include` Anweisung. Die Anweisung `.EXTERNAL` erlaubt den Zugriff auf die angegebene externe gobale Variable und die Anweisung `.ENTRY` ermöglicht Sprünge an Positionen (das Programm-Label wird angegeben) in externen Modulen. Das erste Programm-Label ist `UserBegin`. Dieses wird nur einmal – direkt nach einem Bootvorgang – ausgeführt. Aus diesem Grund werden in diesem Abschnitt nur die Run-Statistik Variablen initialisiert, d.h. die Anzahl der guten bzw. schlechten Events wird nur durch einen Reboot des DSPs null gesetzt. Da die folgenden Routinen auch für Resume Runs ausgeführt werden, gab es keine andere Möglichkeit eine Run Statistik zu führen. Die Instruktionen unter dem Label `UserRunInit` werden in der Initialiserungsphase eines jeden Runs ausgeführt. Hier befindet sich die Initialisierung einiger im weiteren Programmverlauf benötigter Variablen und die Berechnung von Konstanten. Der Haupteil findet sich unter dem Label `UserChannel`. Dieser wird für jeden eingeschalteten Kanal ausgeführt. Nachdem der Wert der Baseline des Signals aus einer mittels der in der USERIN00[5] Variable angebenen Anzahl von Samples ermittelt wurde, wird in Abhängigkeit von der Kanalnummer in die entsprechenden Unterroutinen gesprungen. Für die Kanalnummer 0 wird immer der Steepest Slope Algorithmus ausgeführt und für die Kanalnummern 1 bis 3 wird die induzierte Ladung bestimmt. Dadurch ist die Verkabelung der DGF-4C Module mit den Core- und Segmentsignalen festgelegt. Für Segmente die eine einstellbare Energieschranke (USERIN03) überschritten haben, wird die induzierte Ladung nicht über den betragsmäßig größten Differenz zwischen Baseline und Pulswert bestimmt [11], sondern über die Differenz zwischen Pulswert und einem ab Pulsbeginn linear ansteigenden Signal dessen Steigung durch $\frac{\text{Energie bzw. Pulshöhe}}{\text{Pulslänge}}$ gegeben ist [12]. Hier bieten sich mehrere Implementierungsmöglichkeiten an. Als Pulshöheninformation kann beispielsweise der Höhenunterschied zwischen Pulsanfang und Pulsende (XIA Software CFD) oder die Energieinformation der XIA Elektronik benutzt werden. Diese Informationen können mit einfach zu berechnenden Parametern skaliert werden, wie z.B. 7/8 oder 15/16. Die Pulslänge könnte, wie es für den XIA Code zur Korrektur des ballistischen Defizits geschieht, als Parameter an den Code übergeben oder aus der Pulsform berechnet werden. Zusammen mit der auf 7/8 skalierten Pulshöhe, könnte man so die T_{90} Zeit bestimmen. Ohne weiteres ist es auch möglich, vorberechnete Pulsformen in den Userbereich des DSP-Speicher zu laden, diese der Energie entsprechend skalieren zu lassen, um die so erhaltenen Pulsformen zur Analyse zu verwenden (Vergleiche).

Da die sinnvolle Information über die induzierte Ladung nur aus dem Anfang des Pul-

[5] Belegung im Anhang

ses bestimmt werden kann, ist die Analyselänge mittels der Variablen USERIN04 einstellbar. Unter `UserEventFinish` angegebene Instruktionen werden im Anschluss an die Verarbeitung eines Events ausgeführt. Da ein einzelner MINIBALL Detektor sieben Kanale benötigt, ein DGF-4C Modul allerdings nur vier Kanäle hat und es ausserdem nicht möglich ist, Daten zwischen den Karten auszutauschen, bleibt diese Aufgabe dem Host-PC überlassen. Die aktuelle Code Version erhöht an dieser Stelle die Zähler für die (nicht) erfolgreichen Events. Wird der Buffer-Modus der XIA Karte benutzt, d.h. es werden mehrere Events pro Run aufgenommen, so muss das kopieren der PSA Ergebnisse in die USEROUT-Variablen schon hier erfolgen. Zur Zeit findet dies in der Routine UserRunFinish statt, die am Ende eines Runs aufgerufen wird. Hier werden auch die Variablen zur Run Statistik übergeben, allerdings – mangels eines Labels, welches nur für New Runs aufgerufen wird – nicht beim nächsten neuen Run Start zurückgesetzt.

Der Endanwender hat letztendlich nur noch wenige Parameter einzustellen, dann kann der User Code benutzt werden. Der erste Parameter der gesetzt sein muss, ist die Länge der Baselineanalyse. Diese muss eine Potenz von 2 sein. Will man, dass die Baseline aus den ersten 16 Samples der Pulse bestimmt wird, so muss in diesem Fall der Wert 4 (also log(16)) in die Variable `USERIN00` geschrieben werden. Als nächstes muss angegeben werden, über wieviele Samples beginnend ab Ende der Baselineanalyse (in unserem Fall 17) nach dem Steepest Slope gesucht wird. Da die Berechnung lange Schleifen benutzt, sollte dieser Wert so klein wie möglich ausfallen. Die Analyselänge wird in die Variable `USERIN01` geschrieben. In der Variable `USERIN02` wird die Höhe der Schwelle zur Detektion des Pulses für den implementierten Startalgorithmus geschrieben. Die beiden letzten Variablen sind Eingaben an die Segmentanalyse. Der in die Variable `USERIN03` geschriebene Wert wird von der gemessenen Energie der XIA Karte subtrahiert. Erst wenn diese Differenz größer als Null wird, wird der Algorithmus für getroffene Segmente ausgeführt. Die Analyselänge für diesen Algorithmus wird in die Variable `USERIN04` geschrieben. Mithilfe der Variablen `USERIN08`, `09` und `10` können einzelne Programmteile ausgeschaltet werden.

Die Ausgabevariablen sind im Gegensatz zu den Eingabevariablen vollständig belegt. Alle Zeitangaben sind im `8.8` Format kodiert. Es wurde hier das gleiche Format gewählt, das der XIA Code zur Ausgabe der CFD-Zeit benutzt. Der Start des Pulses kann somit spätestens im 256ten Sample erfolgen, womit der größten Teil der Anwendungen abgedeckt sein sollte (FIFO Länge 1024 Samples). Gleichzeitig ist die kleinste Zeiteinheit ein 256tel eines Samples, also ~ 0.1 ns, was mehr als ausreichend ist. Das Format kann bei Bedarf abgeändert werden. Eine weitere Klasse an Ausgabevariablen sind die Bufferadressen und -längen. Mit diesen Angaben kann der Inhalt der User Buffer über den CAMAC Bus ausgelesen werden und beispielsweise mit den Ergebnissen einer Offline Analyse verglichen werden. Es soll darauf hingewiesen werden, dass der DSP Code die Buffer nicht nutzt, weshalb diese nicht initialisiert werden. Der DSP arbeitet direkt mit den berechneten Daten. Die Buffer dienen nur zur Information.

Erst nach Fertigstellung des User Codes wurde von der Firma XIA eine Testmöglichkeit für den User Code ermöglicht. Damit können künstliche Daten in den Event Buffer der XIA Karte geschrieben werden. Wird anschließend ein Run gestartet, so bemerkt die Karte die Daten im Eventbuffer und bearbeitet sie wie normale Eventdaten. Aber auch nur mit den 16 USEROUT Variablen und einiger Buffer kann ein DSP Programm erfolgreich debugged werden, was der vorliegende Code beweist.

Kapitel 5

Anwendung und Erprobung

'Trying is the first step to failure.'
Homer J. Simpson

Im folgenden Kapitel werden einige Erfahrung und Messungen zusammengestellt, die bei der Inbetriebnahme des Moduls DGF-4C der Firma XIA und den Tests des in der vorliegenden Arbeit entwickelten Usercodes erhalten wurden. Ein detaillierter Test der von der Firma XIA implementierten Algorithmen zur Energie- und Zeitbestimmung wurde von D.Weisshaar [12] durchgeführt.

5.1 Einfluß der Filterlänge und -lücke auf die Energieauflösung

Es ist von Interesse, ob das durch die Gleichungen 3.16 und 3.17 beschrieben Verhalten tatsächlich beobachtet werden kann. Dazu wurde bei ansonst konstanten Parametern (DECIMATION=4), die Energieauflösung für 1,3 MeV γ-Quanten bei varierenden Filterparametern gemessen. Eine Vergleichsmessung mit analoger Standardelektronik (3 μs Shapingzeit) ergab auf Anhieb eine Auflösung von 2,5 keV für den zentralen Kontakt. Abbildungen 5.1 zeigt die Energieauflösung des zentralen Kontakts in Abhängigkeit von der Slow Filter Lücke (SG). Wie erwartet verschlechtert sich mit zunehmender Lücke die Auflösung, da der Step Noise proportional zu dieser zunimmt. Weitere Gründe für die schlechter werdende Energieauflösung liegen möglicherweise auch darin, dass der Filterwert mit zunehmender Lücke durch die Ballistic Deficit Korrektur schlechter korrigiert wird und an der mit zunehmender Lücke abnehmenden Signal- bzw. Filteramplitude, wodurch sich das Signal zu Noise Verhältnis verschlechtert. Die Energiefilterlücke sollte deshalb nicht größer als unbedingt notwendig eingestellt werden. Für FPGA-Code mit DECIMATION=4 (ermöglicht die Einstellung der Filterzeiten in Einheiten von 0,4 μs) und MINIBALL Detektoren mit einer Anstiegszeit von $\leq$ 500 ns, sind 0,8 bzw. 1,2 μs Lücke gute Parameter. Für die folgenden Messungen wurde SG=1,2 μs gewählt.

In Abbildung 5.2 ist das Ergebnis einer Messung der Auflösung in Abhängigkeit von der Slow Filter Länge (SL) dargestellt. Die DECIMATION war 4, d.h. der Energiefilter arbeitet mit den Mittelwerten aus 16 Samples. Eine Vergrößerung der Filterlänge reduziert

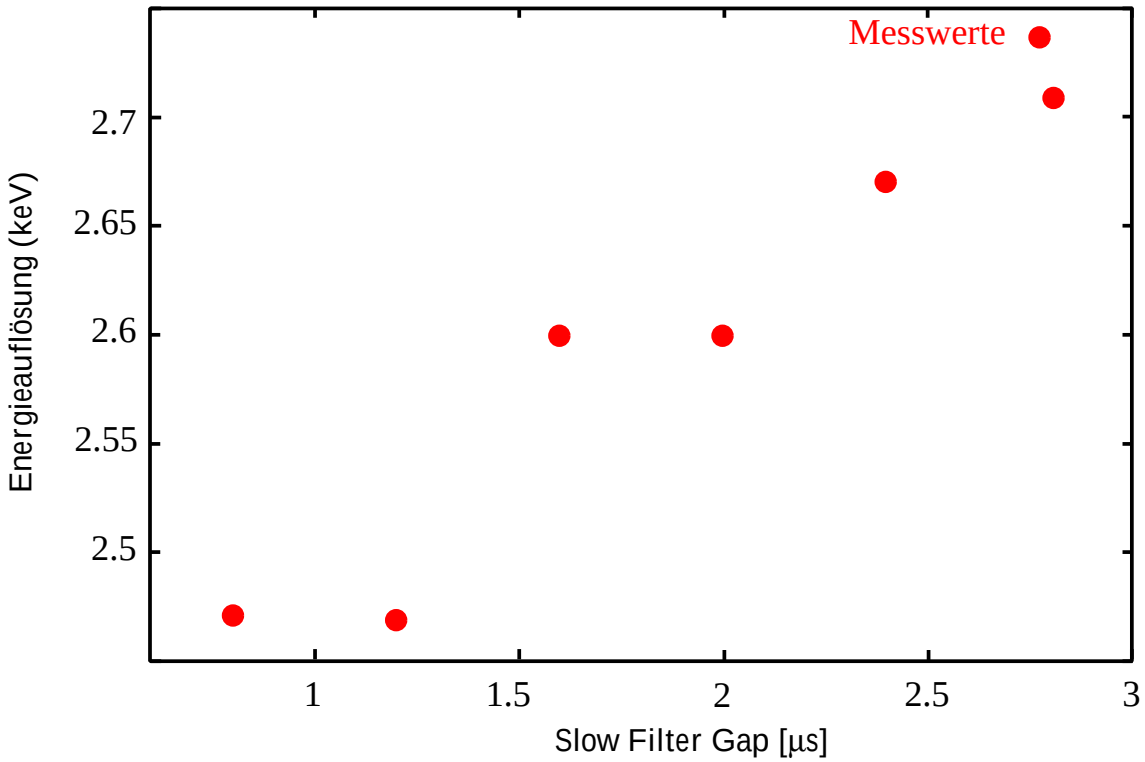

Abbildung 5.1: Einfluß der Filterlücke SG auf die Energieauflösung. Die Auflösung verschlechtert sich mit zunehmender Filterlücke. Die Werte aller anderen Parameter wurden nicht verändert.

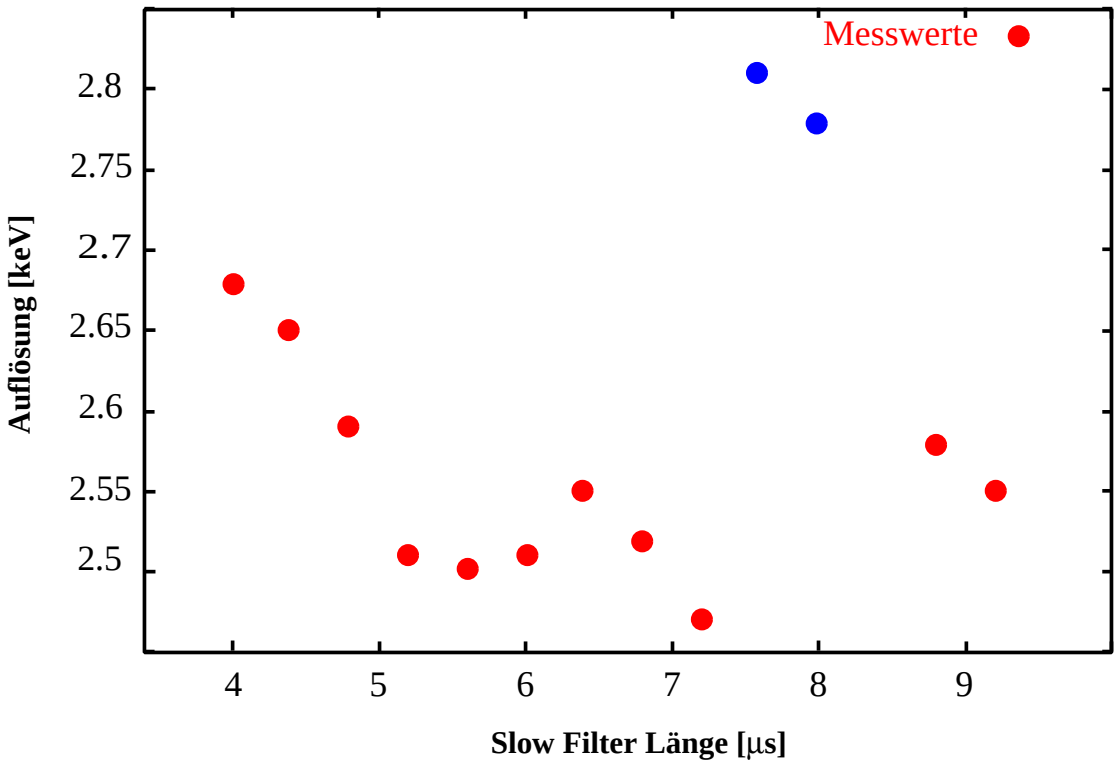

Abbildung 5.2: Einfluß der Filterlänge SL auf die Energieauflösung, alle anderen Variablenwerte blieben unverändert . Auffallend sind die beiden Ausreiser, die sich damit erklären lassen, dass der Energiefilterwert zu einem ungünstigen Zeitpunkt ausgelesen wird. Der Wert des Parameters PEAKSAMPLE, der diesen Zeitpunkt steuert, muss entsprechend angepasst werden. Ansonsten ist zu bemerken, dass eine Filterlänge von 5,2 μs nur eine geringfügig schlechtere Auflösung liefert als eine Filterlänge von 7,2 μs, womit sich aber 4 μs an der Gesamtfilterlänge ($2 \cdot \mathrm{SL} + \mathrm{SG}$) einsparen lassen.

den Anteil an Delta Noise und erhöht den Beitrag an Step Noise. Man sieht, dass sich mit zunehmender Filterlänge die Auflösung zunehmend verbessert, d.h. die Reduzierung des Delta Noise Beitrags überwiegt den Anstieg an Step Noise. Für SL $\geq$ 5 μs bleibt die Auflösung über einen größeren Bereich relativ konstant, d.h. die Beiträge gleichen sich in etwa aus. Allerdings fallen zwei Messwerte aus der Reihe (blau). Die Signatur (Verschlechterung der Auflösung um $\sim$ 10 %) ist laut XIA eindeutig einem schlecht gewählten Zeitpunkt der Energiemessung zuzurodnen. Infolgedessen muss der Wert der Variable, die den Zeitpunkt der Auslese des Energiefilters bestimmt (PEAKSAMPLE), so lange variiert werden ($\pm$ 1 oder 2) bis man eine zufriedenstellende Auflösung erreicht hat. Sinnvolle Energiefilterlänge liegen demnach im Bereich über 5 μs, sofern PEAKSAMPLE angepasst wird. Für die hier erwähnten Messungen wurde eine Filterlänge von SL=7,2 μs gewählt. Es soll bemerkt werden, dass die Energieauflösung immerhin leicht besser als beim analogen Äquivalent ist.

5.2 Totzeiten

Bearbeitungszeit des Usercodes

Es muss dafür gesorgt werden, dass die Ausführung des Usercodes zur Pulsformanalyse die Totzeit des Moduls nicht zu sehr erhöht. Die Prozessorzeit lässt sich, da es sich um einen DSP handelt, leicht abschätzen. Jede Zeile (Instruktionsende ist der Strichpunkt) des im Anhang aufgeführten DSP Codes benötigt 25 ns. Damit lässt sich der Aufwand der einzelnen Programmteile abschätzen. Für die Bestimmung der Baseline auf allen vier Kanälen wird (wenn die ersten acht Samples benutzt werden) eine Zeit von $\sim$ 2,5 μs veranschlagt. Die Bestimmung der induzierten Ladung eines nicht getroffenen Segments nimmt $\sim$ 1,5 μs in Anspruch. Dieser Wert erhöht sich für ein getroffenes Segment auf $\sim$ 2,5 μs. Die Bestimmung der Zeit zwischen Pulsanfang und Steepest Slope Zeitpunkt ist am aufwändigsten. Die derzeitige Methode benötigt $\sim$ 7,5 μs. Im schlimmsten Fall (drei getroffene Segmente) ergibt sich somit eine Laufzeit von $\leq$ 20 μs für die aktuelle Version.

Diese Überlegungen wurden schon für eine frühere Version des Usercodes angestellt. Damals ergab sich eine berechnete Laufzeit von rund 25 μs für ein Event mit nur einem getroffenen Segment. Dieser Wert wurde durch eine Messung mittels Oszilloskop bestätigt. Dabei wurde die Zeit zwischen einem 0$\rightarrow$1 Übergang (Run Stop) und einem 1$\rightarrow$0 Übergang (Run Start) gemessen. Da das Run Stop Signal nach der Auslese der FPGAs und FIFOs und vor der Pulsformanalyse erfolgt, d.h. die Verarbeitungszeit der Pulsformanalysen ist in dieser Zeit enthalten, sollte sich aus der Differenz der Laufzeiten die Länge des Usercodes bestimmen lassen, sofern zusätzlicher Overhead vermieden wird. Dies geschah durch Entfernung der Auslese der List Mode Daten über den CAMAC Bus aus dem Code der Datenaufnahme. Infolgedessen wurde direkt im Anschluß an einen erfolgreichen Run die Datenaufnahme erneut gestartet. Die damalige Messung ergab einen Wert von 60 μs bei ausgeschalteten Usercode (Dieser Wert ist natürlich davon abhängig, für wieviele Kanäle der XIA Code Pulsformanalysen (Ballistic Deficit, CFD) ausführen muss) und einen Wert von 85 μs bei eingeschaltetem Usercode. Die Differenz bestätigt die Abschätzung von 25 μs für die frühere Version. Es wurde ebenfalls der Overhead durch die Auslese per fast CAMAC Adapter gemessen. Dank des Usercodes kann auf eine Auslese der Tracedaten

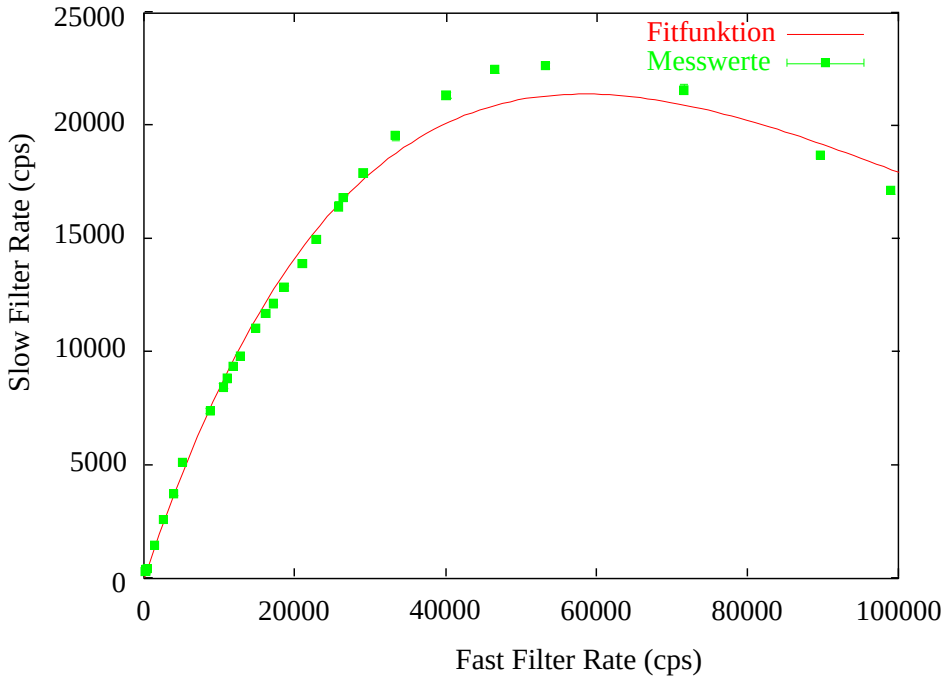

Abbildung 5.3: Totzeit für den paralysierbaren Energiefilter. Aufgetragen ist die Slow Filter Rate gegen die Fast Filter Rate. Da die Slow Filter Ereignisse nicht – wie die Fast Filter Ereignisse – gezählt werden, wird angenommen, dass die Anzahl der DSP Ereignisse, denen der Slow Filter Ereignisse entspricht. Die Totzeit des Fast Filters wurde vernachlässigt. Der Fit mit der Funktion 3.13 führt auf eine Totzeit von $\sim$ 17 μs. Erwartet wurde ein dem Wert, der dem des Parameters PEAKSEP=9.2 μs entspricht. Mögliche Erklärungen finden sich im Text.

verzichtet werden. Stattdessen wurden nur die Header Daten jedes Kanals ausgelesen. Es ergab sich hieraus eine Verbesserung von $\sim$ 600 μs auf $\sim$ 120 μs, was der um einen Faktor fünf reduzierten Datenmenge entsprach.

Versuche die Totzeit des Moduls DGF-4C zu bestimmen

Ein größeres Problem stellt die Messung der Totzeit des Moduls DGF-4C dar, da erstens die wahren Ereignisraten nicht bekannt sind (die sich allerdings mit Kenntnis der Totzeit des schnellen Filter berechnen lassen), zweitens die von der Karte gemessenen Zeiten und Ereigniszahlen nicht in dem gewünschten Maße dokumentiert sind und drittens es kein “echtes“ Busy-Signal gibt. Der letzte Punkt ist verständlich, ist die Karte doch in der Lage, während der DSP Busy Zeit durch den FPGA Daten aufnehmen zu lassen. Das vorhandene Busy-Out Signal wird zusammen mit dem Synch-Eingang der Karte zur Synchronisation des Datenaufnahmestarts verwendet.

Der schnelle Triggerfilter ist paralysierbar. Seine Filteränge FL beträgt im vorliegenden Fall 0.1 μs, die Lücke FG wurde Null gesetzt. Wird als Totzeit des schnellen Filter der Wert der Variable MAXWIDTH=Filterlänge + Pulsanstiegszeit (rund 500 ns) (Totzeit insgesamt wohl rund 1 μs) angenommen, so ergibt sich aus Gleichung 3.13 eine maximale Ausgaberate von ca. $6 \cdot 10^5$ Events pro Sekunde. Für Ereignisraten weit unterhalb ($\sim$ 10

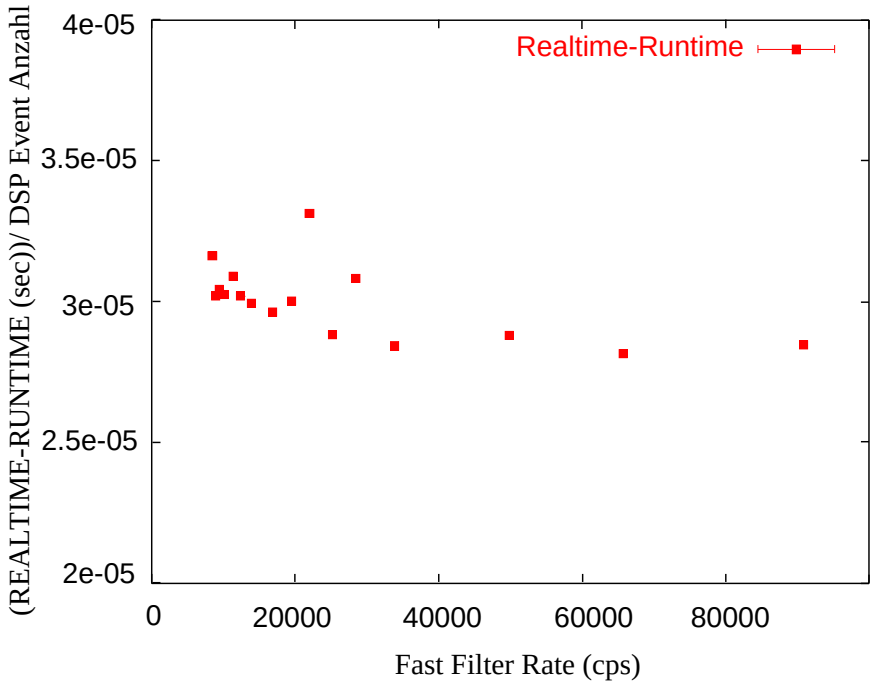

Abbildung 5.4: Die Differenz zwischen REALTIME und RUNTIME ist nahezu konstant und beträgt $\sim$ 30 μs, obwohl der DAQ-Overhead durch Entfernung der CAMAC Auslese minimiert wurde.

kHz) dieses Grenzwertes sollte die schnelle Filterrate der wahren Ereignisrate entsprechen, bei höheren Raten müssen die Fast Filter Raten wegen der Totzeit des schnellen Filters korrigiert werden. Dies ist im Prinzip leicht möglich, da dieser eine (reine) paralysierbare Totzeit haben sollte. Als Totzeit kann der Wert der Variable MAXWIDTH angenommen werden, da eine Überschreitung dieses Wertes als Pileup im schnellen Kanal gedeutet wird. Im folgenden wurde die Totzeit des schnellen Filters vernachlässigt, da keine Messungen zur exakten Bestimmung der Totzeit des Fast Filters durchgeführt wurden. Der Energiefilter des Moduls DGF-4C ist ebenfalls paralysierbar. Die Ankunft eines Events vor der Auslese des Filterwerts, erhöht die Totzeit erneut um den in der Variable PEAKSEP angegeben Wert (SL+SG+2). Damit ergibt sich hier eine maximale Slow Filter Rate von beinahe 40 kHz bei einer Fast Filter Rate von 63,3 kHz. Zieht man die gesamte Filterlänge in Betracht, so folgt daraus eine maximale Slow Filter Rate von fast 22,5 kHz bei einer Fast Filter Rate von ca. 36,600 kHz.

Das Modul DGF-4C ist, im Gegensatz zu seinem Vorgängern DXP-4C, in der Lage die Pulsformen durch den DSP analysieren zu lassen. Das Modul DXP-4C arbeitet nur mit den Energiefilterwerten. Für dieses Modul ist die Korrektur bzw. Messung der Totzeit dokumentiert. Dieses Modul liefert eine LIVETIME der FPGAs und die Anzahl an Fast und Slow Filter Ereignissen. Diese Methode sollte, trotz der Unterschiede, im Prinzip auch auf das Modul DGF-4C anwendbar sein. Auf eine Sache soll allerdings hingewiesen werden. Keine der im folgenden aufgeführten Zeiten (LIVE-, RUN-, REALTIME) ist exakt dokumentiert, d.h. es ist nicht genau bekannt welche Zeiträume damit gemessen werden. Die Messung der LIVETIME geschieht durch den FPGA, weshalb hierzu nichts bekannt

ist. Die REALTIME ist die durch den DSP-Timer gemessene Zeit. Während diesen beiden Zeiten einigermaßen vertraut werden kann, bestehen Zweifel an der Glaubwürdigkeit der RUNTIME.

Die Fast Filter Rate erhält man durch Division der Fast Filter Ereignisse eines Kanals durch die zugehörige LIVETIME, also

$$\text{Fast Filter Rate} = \frac{\text{FASTPEAKS}}{\text{LIVETIME}} \quad . \tag{5.1}$$

Da die Anzahl der Slow Filter Ereignisse im Modul DGF-4C nicht gezählt wird, wurde angenommen, dass die Anzahl an DSP Ereignissen der Anzahl an Slow Filter Ereignissen entspricht. Dies ist sinnvoll, da die LIVETIME die Zeiten, in denen das FPGA keine Slow Filter Events an den DSP schicken kann, ausschliesst. Ausserdem ist kein Grund bekannt weshalb der DSP ein akzeptiertes Slow Filter Ereignis nachträglich verwerfen sollte. Die Slow Filter Rate wurde demnach durch Division der Anzahl an DSP Ereignissen (NUMEVENTS) durch die LIVETIME des Eventtriggerkanals erhalten:

$$\text{Slow Filter Rate} = \frac{\text{NUMEVENTS}}{\text{LIVETIME}} \tag{5.2}$$

In Abbildung 5.3 ist die Slow Filter Rate gegen die Fast Filter Rate aufgetragen. Fittet man an diese Messungen die Funktion des paralysierbaren Totzeitverhaltens 3.13, so ergibt sich eine Energiefilter Totzeit von 17,2±0,2 μs. Da die Totzeit durch die Variable PEAKSEP (=SL+SG+2=9,2 μs) bestimmt sein sollte, wurde ungefähr die Hälfe erwartet. Stattdessen liegt der Wert sogar über der gesamten Filterlänge von 15.6 μs. Eine mögliche Ursache könnte die Auslese eines Baselinewertes direkt im Anschluß an ein Event (wenn der Filterausgang auf "Null" zurückgegangen ist) oder die Auslese eines ADC Wertes sein, sodass bis zum vollständigen Abklingen des Filters kein weiteres Event erfolgen darf. Es darf ebenfalls nicht vergessen werden, dass die FIFOs zur Auslese der Tracedaten angehalten werden (Ausnahme: fast list mode). Ist die FIFO-Auslese in der LIVETIME enthalten, so vergrößert sich die Totzeit bei einer Tracelänge von 40 Samples (wie dies der Fall war) um ~ 1 μs pro Kanal.

Es wurde versucht, diese Methode auf den DSP anzuwenden, um so zu einer funktionalen Beschreibung des Totzeitverhaltens des Moduls DGF-4C zu gelangen. Durch die Event-by-Event Auslese kann die XIA-Elektronik ihre Pipeline Fähigkeiten nicht anwenden, d.h. ist ein Event durch den Pileup-Inspector akzeptiert worden, so wird dieses (unausweichlich) durch den DSP weiterverarbeitet. Der FPGA nimmt kein weiteres Event auf, da die eigentliche Datenaufnahme abgeschlossen ist (MAXEVENTS=1). Die Eventverarbeitung durch den DSP ist deshalb nicht paralysierbar. Die Rate des DSPs y sollte demnach eine Funktion g der Slow Filter Rate f sein, die ihrerseits wiederum eine Funktion der Fast Filter Rate x ist, was einen funktionalen Zusammenhang der Art $y = g(f(x))$ ergibt:

$$y = \frac{x \cdot e^{-x \cdot \tau_1}}{1 + (x \cdot e^{-x \cdot \tau_1}) \cdot \tau_2} \quad , \tag{5.3}$$

wobei x die Fastfilterrate, $x \cdot e^{-x \cdot \tau_1}$ die Slowfilterrate und y die DSP Eventtrate bezeichnet. Für die Totzeiten des Slowfilters und DSPs wurden die beiden Werte τ_1 und τ_2 eingefügt.

Der Durchsatz des gesamten Systems sollte sich Mithilfe der REALTIME, die die Zeit zwischen Start und Ende eines Runs mißt, bestimmen lassen, also:

$$\text{System Rate} = \frac{\text{NUMEVENTS}}{\text{REALTIME}} \quad . \tag{5.4}$$

Die DSP Eventrate ihrerseits, müßte Mithilfe der RUNTIME, die die Zeit mißt in der die Datenaufnahmne der Karte aktiv ist, bestimmen lassen, also:

$$\text{DSP Rate} = \frac{\text{NUMEVENTS}}{\text{RUNTIME}} \quad . \tag{5.5}$$

Laut Anleitung (Programmer's Manual) erlaubt ein Vergleich zwischen REALTIME und RUNTIME eine Abschätzung des Overheads, der aus der Datenaufnahme (z.B. Auslese, erste Auswertung, Neustart) resultiert. Da das Modul nicht das Ende eines Runs anzeigen kann, wurde vermutet, dass auch die RUNTIME nicht die wahre Dauer eines Runs mißt. Deshalb wurde eine Messung durchgeführt, bei der wieder einmal die CAMAC Auslese aus dem Code der Datenaufnahme entfernt wurde. Das Ergebnis ist in Abbildung 5.4 gezeigt. Statt der erwarteten Differenz von Null, denn die Auslese wurde entfernt, besteht ein nahezu konstanter Unterschied von ~ 30 μs zwischen der REALTIME und der RUNTIME. Innerhalb dieser Zeit kann die VME CPU 3000 Instruktionen ausführen. Die DSP Code Schleife, die das gemeinsame Starten steuert, ist bekanntlich nur 0.1 μs lang. Es wird deshalb vermutet, dass die RUNTIME zur Bestimmung der DSP Totzeit ungeeignet ist. Diese Vermutung wird dadurch Unterstützt, dass das Modul DGF-4C, wie erwähnt, nicht in der Lage ist das wirkliche Ende eines Runs anzuzeigen (Bit setzen im CAMAC CSR), sondern im Gegenteil zu einem zu frühen Zeitpunkt das Run Ende anzeigt. Die Ursache liegt darin, dass das Bit, welches das Ende eines Runs anzeigen sollte, zu Synchronisation der Module (Busy-Synch-Loop) benutzt wird und deshalb gar nicht das wirkliche Ende (Daten im List Mode Buffer sind bereit zum Auslesen) anzeigen soll. Es soll betont werden, dass die Zeitmessung Mithilfe des DSP-Timers prinzipiell auf 25 ns exakt ist und deshalb hierdurch kein Fehler entstehen kann. Wäre dies nicht der Fall, so könnte auch der Eventzeitpunkt (TIMESTAMP) nicht exakt bestimmt werden. Ein wesentliches Feature der XIA Elektronik würde nicht funktionieren. Der Fehler liegt vermutlich darin, dass die falschen Zeitintervalle gemessen werden, weshalb diese auch nicht dokumentiert sind.

Trotzdem wurde die DSP Rate bzw. die System Rate gegen die Fast Filter Rate versuchsweise aufgetragen. Auffallend ist ein deutlicher Knick in dieser Auftragung bei einer Fast Filter Rate von $\sim$ 25 kHz. Die Ursache ist unbekannt und nicht verstanden. Allerdings ist hiermit klar, dass das einfache Modell 5.3 nicht zutrifft. Um die Rechenzeit des Usercodes zu bestimmen wurde trotzdem die Formel 5.3 an die Messpunkte gefittet. Dieser Fit führte auf die Werte $\tau_1 = 27.69 \pm 0.65$ μs und $\tau_2 = 71.15 \pm 0.50$ μs für ausgeschalteten Usercode und $\tau_1 = 27.21 \pm 0.65$ μs und $\tau_2 = 88.46 \pm 0.62$ μs ür eingschalteten Usercode. Die FPGA Totzeit τ_1 bleibt wie erwartet nahezu konstant, allerdings liegt ihr Wert sogar über der doppelten Filterlänge (15.6 μs), wird also stark überschätzt. Die DSP Totzeit τ_2 erhöht sich durch den Usercode um rund 17 μs, was dem aus der Codelänge berechneten Wert erstaunlich gut entspricht. Der DSP Totzeit ohne User Code stehen die am Oszilloskop gemessenen 60 μs gegenüber, der DSP Totzeit mit Usercode dementsprechend die 85 μs. Da die Fitfunktion 5.3 die Messungen im Bereich des Knicks nur sehr schlecht beschreibt,

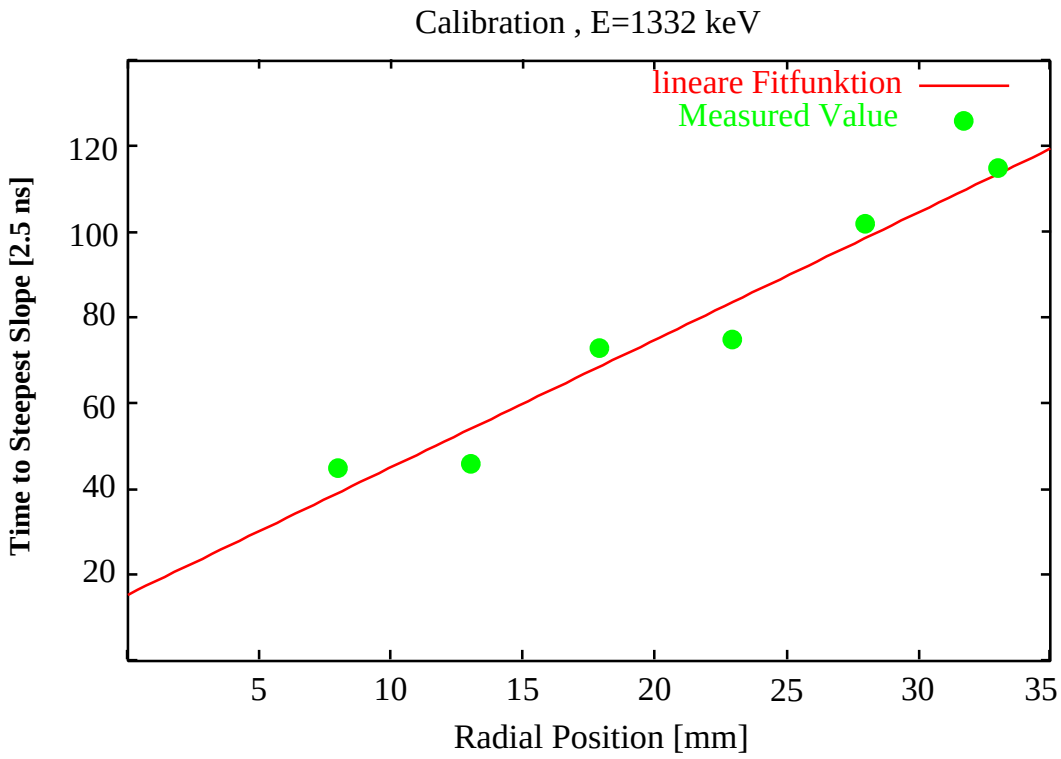

Abbildung 5.5: Eichkurve zur Umrechnung der Steepest Slope Zeit in den Radius R. Eine lineare Eichung hat sich als ausreichend herausgestellt. Der Fit liefert die Umrechnung: R [mm] = 0,337 · T_{slope} [2,5 ns] - 4,41. Die damit erhaltene Eichkurve wurde anschließend zur Umrechnung der Zeiten bis zum Steepest Slope T_{slope} in Radien R benutzt.

erscheint diese Methode nicht sinnvoll zur exakten Bestimmung der DSP Totzeit. Es konnte also keine einfache Beschreibung des Totzeitverhaltens des Moduls DGF-4C hergeleitet werden.

5.3 Messungen mit kollimierter γ-Quelle

Um Funktionalität des User DSP Codes zur R und Φ Bestimmung mittels Pulsformanalyse zu überprüfen, wurden mehrere Messung mit einer kollimierten ^{60}Co-Quellen durchgeführt. Dazu konnte der bereits vorhandene Aufbau [10] benutzt werden. Die Quelle wurden mittels des gestuften Bleikollimators auf die gewünschte radiale Position ausgerichtet. Sollten, bei konstantem Radius, verschiedene Winkelpositionen gemessen werden, so wurden diese durch eine entsprechende Rotation des hängend montierten Detektors (mit Kryostat) bewerkstelligt. Im Unterschied zu früheren Messungen [10, 11] wurden die Detektorpulse mit der XIA Elektronik onboard und in Echtzeit analysiert. Dadurch konnte auf die Auslese der Tracedaten verzichtet werden, d.h. es wurden nur die Headerdaten aus dem List Mode Buffer der DGF-4C Module gelesen. Ausgewertet wurden nur Ereignisse mit einer Energie von 1,3 MeV. Innerhalb eines Runs wurde nur ein Ereignis aufgenommen (MAXEVENTS=1). Die Schwelle zur Bestimmung des Startzeitpunkts war auf 20 ADC-Werte eingestellt. Da die Zeitauflösung diese Algorithmus bisher noch nicht gemessen wurde, ist

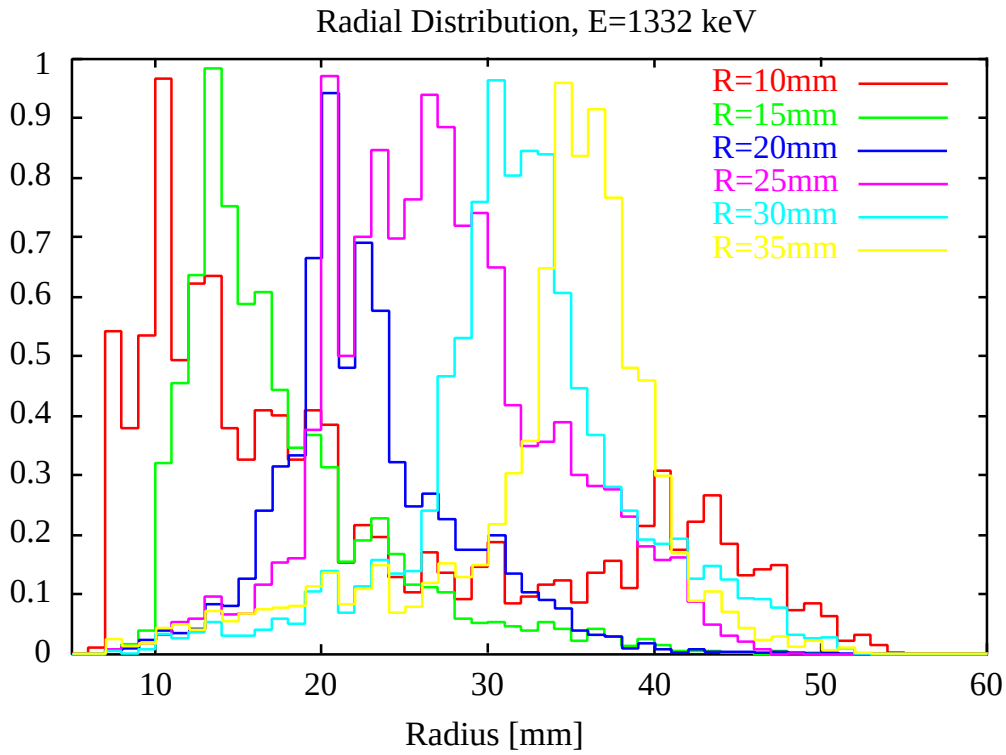

Abbildung 5.6: Radiale Verteilung, erhalten durch Messungen mit einer kollimierten ^{60}Co-Quelle bei senkrechtem Einschuss. Es wurden nur FEP-Ereignisse mit einer Energie von 1,3 MeV bearbeitet. Es wurden bei dieser Energie keine Messungen mit Radien kleiner als 10 mm durchgeführt.

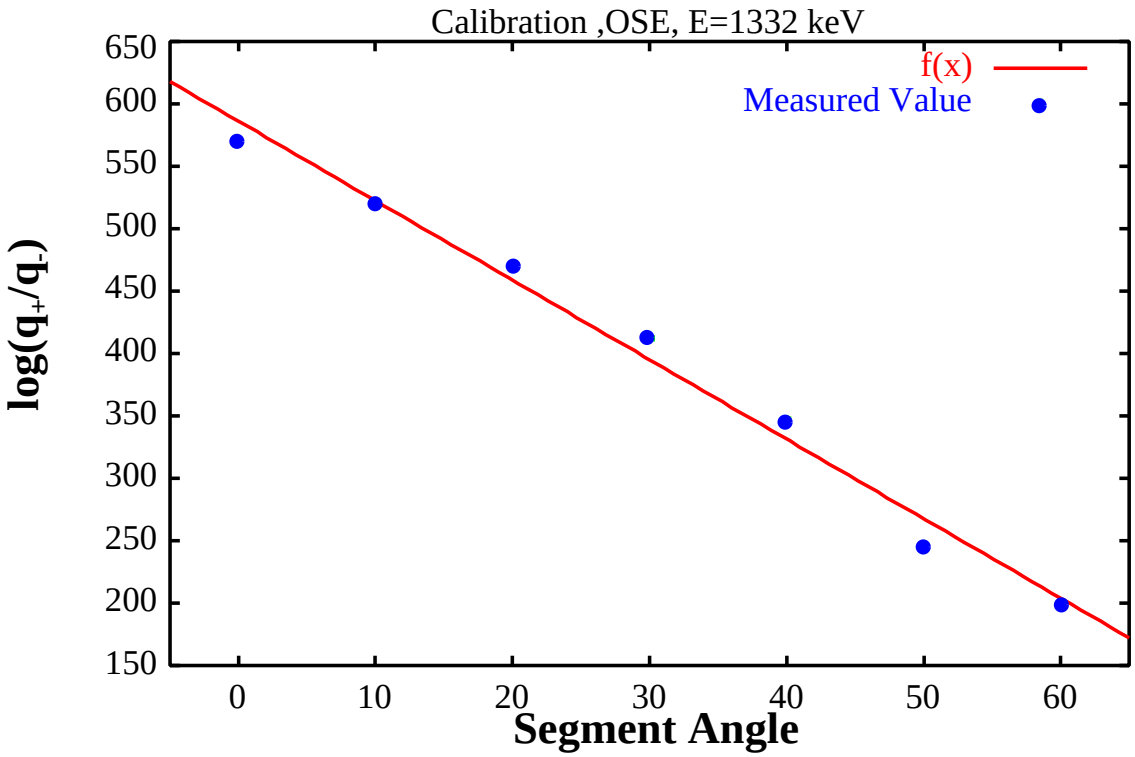

Abbildung 5.7: Lineare Eichkurve zur Berechnung des Segmentwinkels für OSEs. Wie durch Simulationen gezeigt, ergibt sich aus der Form des Weighting Feldes ein linearer Zusammenhang zwischen Segmentwinkel und $log(\frac{q+}{q-})$.

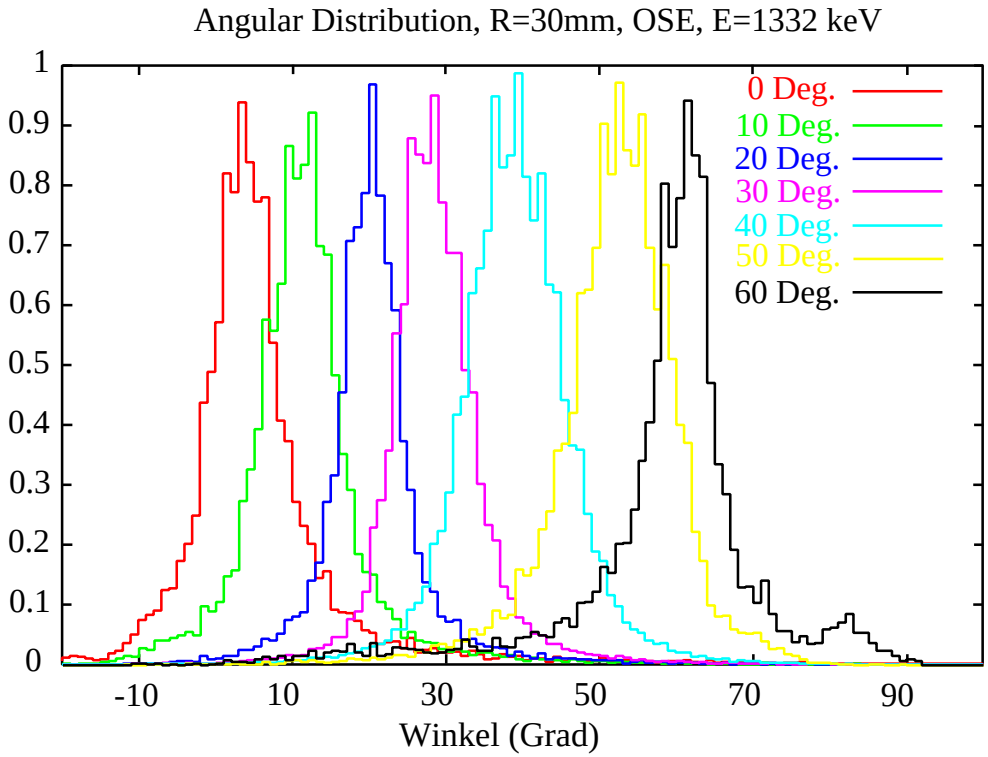

Abbildung 5.8: Winkelverteilung für OSEs, erhalten durch Messungen mit kollimierter ^{60}Co-Quelle bei senkrechtem Einschuss. Die radiale Position wurde für diese Messungen konstant auf R = 30 mm gehalten. Ausserdem wurden nur FEP-Ereignisse mit einer Energie von 1,3 MeV bearbeitet. Der Algorithmus liefert eine ausgezeichnete Separation der mit Winkeldifferenzen von 10 Grad aufgenommenen Verteilungen. Man erkennt ausserdem, dass einige Verteilung nicht den eingestellten Winkeln zugeordnet werden. Dies wurde vermutlich durch eine unpräzise Einstellung der Meßpositionen verursacht. Da der Detektor von Hand gedreht wurde, kann es hierbei leicht zu einer Fehlpositionierung kommen.

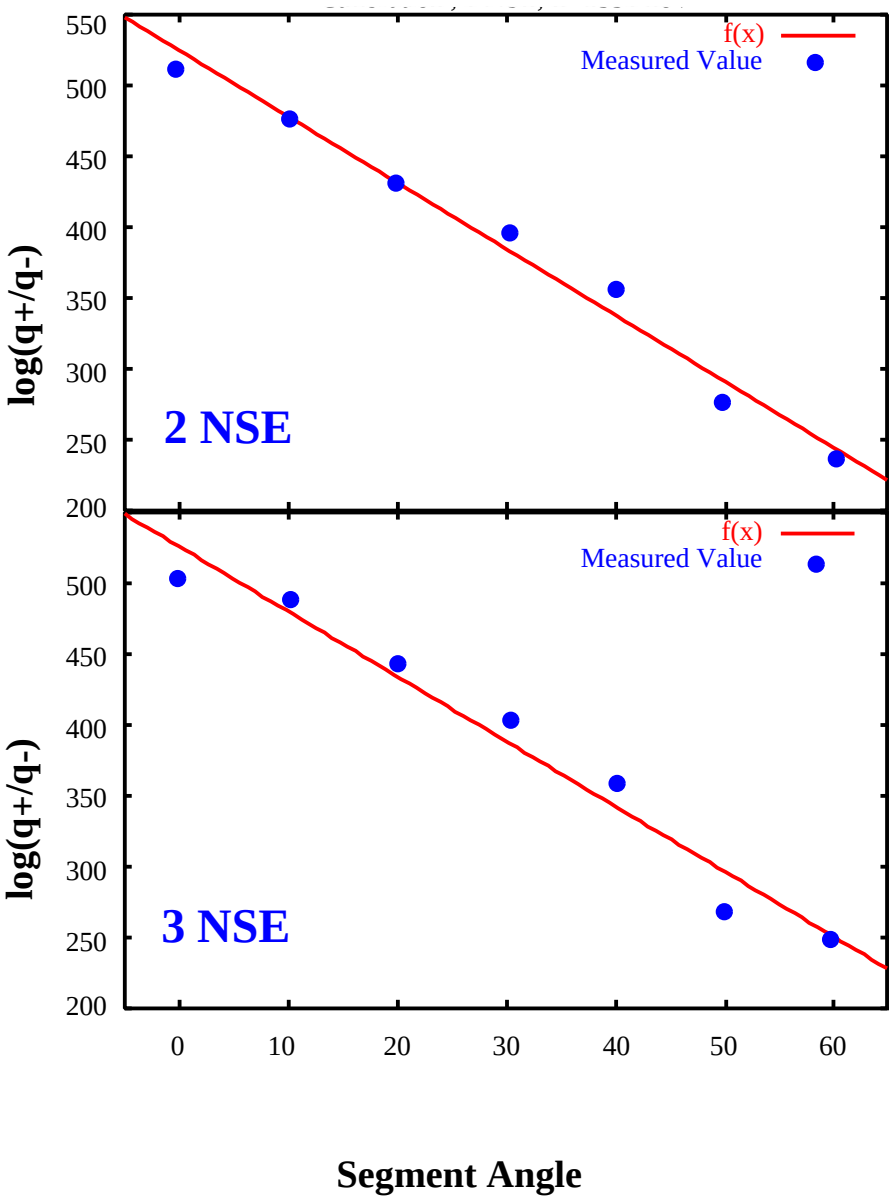

Abbildung 5.9: Eichkurven zur Berechnung des Segmentwinkels für NSEs. Man beachte, dass auch in diesem Falle der lineare Zusammenhang zwischen (internem) Segmentwinkel und den vom Algorithmus gelieferten Werten gilt, allerdings ergibt sich eine von OSE-Fall verschiedene Eichkurve.

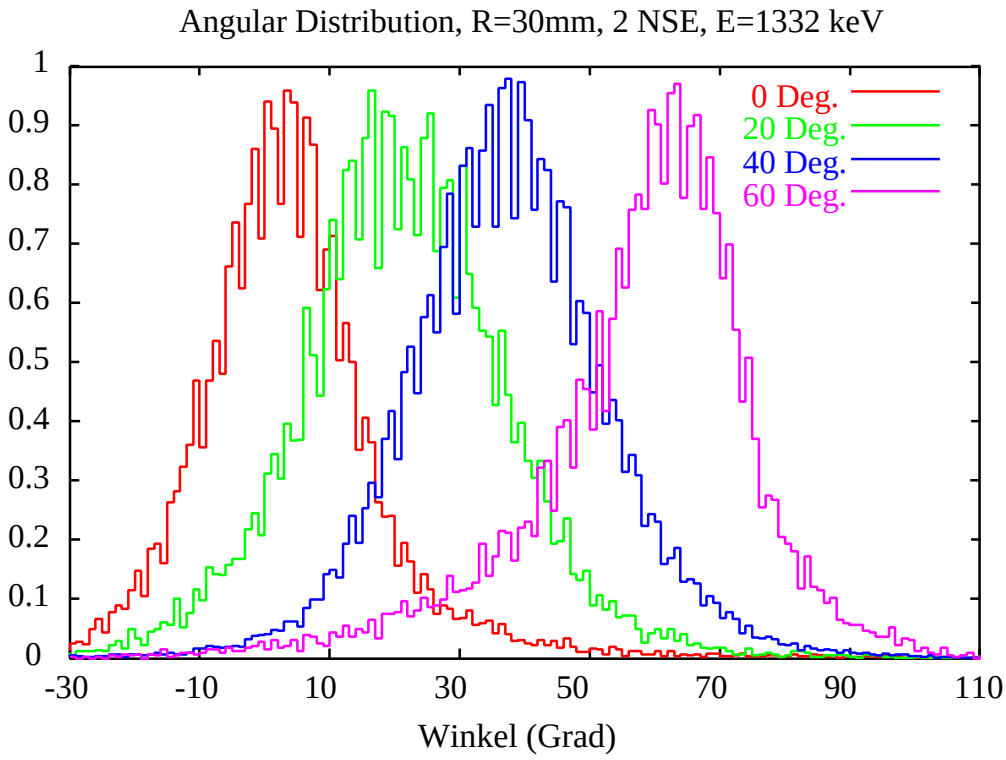

Abbildung 5.10: Winkelverteilung für 2 NSEs, erhalten durch Messungen mit kollimierter ^{60}Co-Quelle bei senkrechtem Einschuss. Der Radius war fest auf 30 mm eingestellt und der Algorithmus wurde nur auf Ereignisse mit einer Energie von 1,3 MeV angewendet. Die Verteilungen, die für Segmentwinkel in Abständen von 20 Grad aufgenommen wurden, sind gut separiert. Allerdings werden die Ereignisse bei einem Segmentwinkel von 60 Grad, durch den Algorithmus einem Winkel von $\sim$ 70 Grad zugeordnet. Dies wurde vermutlich durch eine fehlerhafte Positionierung verursacht.

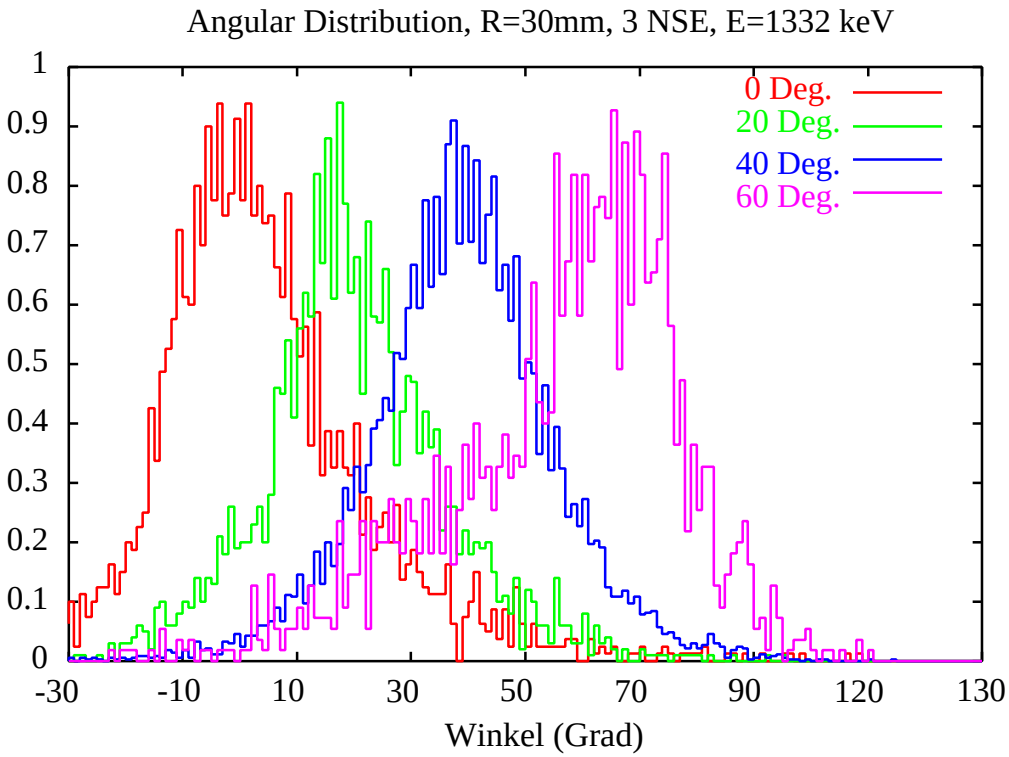

Abbildung 5.11: Winkelverteilung für 3 NSEs, erhalten durch Messungen mit einer kollimierte^{60}Co-Quelle bei senkrechtem Einschuss. Die Verteilung entspricht in etwa der für 2 NSEs. Die Einschusspositionen im Abstand von 20 Grad sind zu unterscheiden. Hier werden die Ereignisse bei einem Einschuss unter einem Segmentwinkel von 60 Grad, im Gegensatz zu den 2 NSEs, auch einem Winkel von 60 Grad zugeordnet. Allerdings werden dafür die Ereignisse für die 20 Grad Position, durch den Algorithmus einem Segmentwinkel von 15 Grad zugeordnet. Auch hier wird vermutet, dass die Ursache eine Fehlpositionierung der kollimierten Quelle war.

es möglich, dass diese Einstellung nicht die optimalsten Resultate liefert. Da der MINIBALL Detektor insgesamt sieben Kanäle hat, wurden zwei DGF-4C Module zur Analyse benötigt. Da die verwendete Vorverstärkerelektronik nur ein einziges Core Signal lieferte, fehlte einer Karte eine korrekte Timing Information. Die mit dieser Karte verbundenen Segmentsignale wurden deshalb nicht zum Test der Winkelauflösung verwendet. Stattdessen wurden alle Messungen mit dem Segment Nummer 2 durchgeführt, da sowohl beide Nachbarn (Segment 1 und 3) als auch das Core Signal, welches die Timing Information liefert, in dem gleichen Modul verarbeitet werden. Die Anbindung der XIA Elektonik an die Datenaufnahme wurde schon in Zusammenhang mit Abbildung 4.11 beschrieben.

In einer ersten Meßreihe sollte die Auflösung des implementierten Algorithmus zur Radiusbestimmung getestet werden. Deshalb wurde für verschiedene Radien, bei senkrechtem Einschuss mit einer kollimierten ^{60}Co-Quelle, die Verteilung der Zeit bis zum Steepest Slope T_{slope} gemessen. Hierbei zeigte sich, dass durch die Interpolationen der Einfluß, der durch den ADC vorgegebenen, Zeitauflösung von 25 ns nicht vollständig entfernt werden kann. Das Maximum dieser Verteilung wurde als die zu diesem Radius gehörende Zeit T_{slope} angenommen. Diese Zeit wurde gegen die gemessenen radialen Positionen aufgetragen. Danach wurde durch einen linearen Fit ein funktionaler Zusammenhang erstellt, der in den darauffolgenden Messungen zur Umrechnung der Steepest Slope Zeit T_{slope} in die radiale Position der Hauptwechselwirkung diente. Abbildung 5.5 zeigt die Auftragung zusammen mit der durch den Fit erhaltenen Funktion T_{slope} [2,5ns] =2,97 $\cdot$ R [mm] + 13,1. In Abbildung 5.6 ist die damit erhaltenen radiale Verteilung dargestellt. Die radiale Position der kollimierten Quelle, wurde dabei jeweils um 5 mm verändert, es wurden allerdings keine Messungen mit Radien kleiner als 10 mm aufgenommen, da in keiner der zuvor durchgeführten Analysen bei diesen kleinen Radien ein signifikanter Unterschied in den Verteilungen zu erkennen war. Man erkennt, dass die Verteilungen der verschiedenen radialen Positionen separiert sind. Auffallend ist, dass bei einem Einschuss im Abstand von 25 mm, eine vermehrte Anzahl von Ereignissen zu niederen Radien eingeordnet werden. Weiterhin ist bemerkenswert, dass bei einem Einschuss im radialen Abstand von 10 mm, eine hohe Anzahl an Events zu Radien größer als 30 mm zugeordnet werden. Dies ist für die Einschusspositionen von R=15 mm nicht mehr der Fall. Die Ursache ist noch nicht bekannt und muss durch erneute Messungen bei kleinen Radien gesucht werden. Trotzdem kann davon ausgegangen werden, dass die derzeitige Implementierung des Echzeit Steepest Slope Algorithmus korrekt und voll funktionsfähig ist, da die Verteilungen in etwa denen entsprechen die durch offline Analysen erhalten wurde [17, 10, 12]. Man verliert also nichts an radialer Positionsauflösung, wenn die Pulsformen, die das DGF-4C Modul liefert, in Echtzeit analysiert werden, stattdessen erhöht sich aber der Durchsatz der Datenaufnahme. Es sollte in diesem Zusammenhang nochmal auf die durch die Elektronik vorgegeben Zeitauflösung von 25 ns hingewiesen werden. Die durch den Fit abgeleitete Funktion liefert für einen Radius von 10 mm eine Zeit T_{slope} von rund 87 μs, zwischen Pulsbeginn und Zeitpunkt des Auftreffen der Elektronen der Hauptwechselwirkung auf den zentralen Kontakt. Das Modul DGF-4C kann innnerhalb diese Zeitraums, aufgrund der Samplingrate von 40 MHz, drei Samples liefern. Die zweite Ableitung des Ladungssignales muß innerhalb dieser Zeit ihren Steepest Rise (Start des Pulses, Erzeugung aller Ladungen), einen Nulldurchgang und ihren Steepest Slope haben. Daraus folgt, dass drei bzw. minimal zwei Samples, d.h. der Nulldurchgang erfolgt zwischen den gemessenen Zeitpunkten, die untere Grenze

für eine sinnvolle Analyse – mittels des Steepest Slope Algorithmus und des Moduls DGF-4C – darstellen. Infolgedessen sollte ein neuer Algorithmus zur exakten Lokalisierung der Hauptwechselwirkung bei kleinen Radien entwickelt werden.

In einer weiteren Meßreihe wurde die Auflösung der implementierten Winkelalgorithmen getestet. Dabei wurde das Segment Nummer 2, bei einer festen radialen Position von 30 mm, mit einer kollimierten ^{60}Co Quelle bei senkrechtem Einschuss für verschiedene (interne) Segmentwinkel Φ_2 bestrahlt. Ausserdem wurde eine Fallunterscheidung eingeführt, da für Ereignisse ohne getroffene Nachbarn (OSE: One Segment Event) und Ereignisse mit einem (2 NSE) oder zwei (3 NSE) getroffenen Nachbarn (Neighbouring Segment Events), wie zuvor erläutert, verschiedene Algorithmen ausgeführt werden. Diese Fallunterscheidung sollte sich bei der Aufnahme der Eichmessungen, die in den Abbildungen 5.7 und 5.9 gezeigt sind, als nützlich erweisen. Zuerst wurde festgestellt, dass sich, wie nach den Simulationen [11] erwartet, ein linearer Zusammenhang zwischen (internem) Segmentwinkel Φ_2 und dem Logarithmus aus dem Verhältnis der, in den direkt benachbarten Segmenten, induzierten Ladung ergibt. Deshalb konnte wiederum eine lineare Fitfunktion benutzt werden. Hierbei stellte sich heraus, dass der funktionale Zusammenhang zwischen Segmentwinkel Φ_2 und dem Algorithmuswert sich für OSEs und NSE aufgrund der unterschiedlichen Methoden, unterscheidet. Für OSEs ergab der lineare Fit Φ_2^{OSE} [Grad] = $-0.16 \cdot$ x - 91,4, wobei x die Ausgabe des Algorithmus und Φ_2 den (internen) Segmentwinkel in Grad angibt. Für NSEs ergab sich aus dem linearen Fit die Funktion Φ_2^{NSE} [Grad] = $-0.22 \cdot$ x - 113,9, wobei die gleichen Bezeichnungen wie zuvor verwendet werden. Danach wurde das Segment Nummer 2 in 10 Grad Abstände mit der kollimierten ^{60}Co Quelle bei senkrechtem Einschuss und konstanter radialer Position bestrahlt. Die Abbildungen 5.8 zeigt die damit erhaltenen Verteilungen für die verschiedenen Einschusspositionen für OSEs, die Abbildungen 5.10 und 5.11 die Verteilungen für 2 NSE bzw. 3 NSE. Auch hier entsprechen die Verteilungen dem, was man nach Simulationen und Offline Analysen erwartet hat. Für OSEs sind die Verteilungen im Abstand von 10 Grad separiert, für NSEs ist dies für eine Winkeldifferenz von 20 Grad der Fall. Demnach kann hier festgestellt werden, dass die derzeitige Umsetzung der Algorithmen zur Winkelbestimmung funktionieren und äquivalente Ergebnisse wie vergleichbare offline Analysen liefert. Ob und wie gut die implementierten Algorithmen bei niedrigeren Energien funktionieren, wird die noch zu erfolgende Auswertung der restlichen Daten zeigen.

Kapitel 6

Zusammenfassender Ausblick

> 'If something is too hard, give it up. The moral my boy is to never try anything.'
> Homer J. Simpson

In der vorliegenden Arbeit wurden Algorithmen zur Pulsformanalyse auf einer neuartigen Elektronik, dem CAMAC Modul XIA DGF-4C, implementiert, die das Vorverstärkersignal direkt digitalisiert. Die digitalen Pulsformen akzeptierter Ereignisse, können durch einen digitalen Signalprozessor, der auch zur Implementierung der eigenen Algorithmen verwendet wird, analysiert werden. Zur Implementierung der Algorithmen stehen aber im Prinzip zwei Bausteine zur Verfügung: FPGA und DSP. Die Firma XIA hat ein Interface zur Programmierung des DSP bereitgestellt, welches in der vorliegenden Arbeit benutzt wurde. Darüber hinaus hat die Firma XIA einer Kooperation zwecks Implementierung einfacher Algorithmen in FPGA Technologie zugestimmt. Ein erster Test einer Hardware Implementierung des Steepest Slope Algorithmus hat deshalb stattgefunden. Er ist im Anhang beschrieben.

Das Modul DGF-4C der Firma XIA wurde erfolgreich in die MPI Datenaufnahme eingebunden und es konnten damit Testmessung durchgeführt werden. Derzeit wird dem Modul erlaubt ein Event pro Run aufzunehmen (MAXEVENTS=1). Die programmierten Datenaufnahmeroutinen sind jedoch nicht auf diesen Modus festgelegt, sondern im Prinzip auch auf Runs, in denen mehrere Ereignisse aufgenommen werden (MAXEVENTS >1), vorbereitet. Allerdings wurde dieser Modus noch nicht getestet, da die Firma XIA eine parallel zu Datenaufnahme stattfindende Auslese ermöglichen wird. Dies wird mit den Modulen der nächsten Revision möglich sein, welche über einen Firewire-Anschluß ausgelesen werden können. Diese Karten wurden kurz vor Fertigstellung dieser Diplomarbeit geliefert, allerdings ohne Interface zur Einbindung des User DSP Codes, weshalb diese Karten nicht mehr getestet werden konnten. Die implementierten Algorithmen zur Bestimmung der Position der Hauptwechselwirkung innerhalb eines einzelnen MINIBALL Moduls benötigen ≤ 20 μs Rechenzeit. Gleichzeitig erlaubt es die Echtzeitanalyse mittels der XIA Elektronik, auf die Auslese der Signaldaten zu verzichten, womit die anfallende Datenmenge um etwa einen Faktor fünf reduziert wird. Die Ergebnisse der Echtzeitimplementierung des Steepest Slope Algorithmus sind vielversprechend. Die Radiusauflösung wird durch die onboard Pulsformanalyse kaum beeinträchtigt. Die Messungen bei kleineren γ-Energien wurden noch nicht ausgewertet. Erst wenn dies der Fall ist, kann behauptet werden, dass die Implementierung vollständig gelungen ist. Dies kann ebenfalls von den derzeit implementierten Algorithmen

zur Bestimmung des Segmentwinkels behauptet werden. Auch hier erreicht die Echtzeitanalyse bei einer Energie von 1,3 MeV die Qualität vorangegangener offline Analysen. Für die Messungen wurde ein MINIBALL Modul mit einer kollimierten ^{60}Co Quelle bestrahlt. Die radiale Position des Einschuss wurde auf 30 mm fixiert. Für One Segment Events, die die Statistik dominieren, wird unter diesen Bedingungen ungefähr die doppelte Auflösung wie für Neighbouring Segment Events erreicht. Damit ist man dem Ziel, die Granularität eines einzelnen MINIBALL-Moduls von 6 auf 100 zu erhöhen, sehr nahe gekommen.

Vom Gebrauch der Zeitparameter (RUNTIME, REALTIME) zur Bestimmung der Totzeit des Moduls DGF-4C ist vorerst abzuraten, da deren Messintervalle noch nicht dokumentiert sind. Da diese in Software mittels des DSP-Timers gemessen werden und der DSP vermutlich nicht über den Status des Host-Interfaces (CAMAC-FPGA) informiert ist, ist anzunehmen, dass die Totzeit des Datenaufnahmesystems mit diesen Parametern nicht bestimmmt werden kann, sofern die unteren 16 Bit der REALTIME nicht den Wert des DSP-Timers spiegeln. Für ein einzelnes Modul und unter Auslassung der CAMAC-Auslese, könnte eventuell mittels der Realtime eine DSP-Totzeit bestimmt werden, was allerdings noch nicht verifiziert werden konnte. Durch Analyse des XIA Codes konnte die genaue Bedeutung der Zeitvariablen noch nicht in Erfahrung gebracht werden. Der Autor ist jedoch davon überzeugt, dass eine Zeitmessung in Hardware der gegenwärtigen Lösung, trotz DSP-Timer Interrupts, vorzuziehen wäre, so wie dies auch für die LIVETIME der Fall ist.

Es ist festzustellen, dass der Einzug digitaler Methoden der Hochenergiephysik in die Kernphysik mit Erfolg vollzogen wurde. Dies ist umso bedeutender, liegen doch die Ansprüche der Germaniumdetektoren an die Signalqualität um ein Vielfaches höher als beispielsweise die eines Kalorimeters, dessen Energieauflösung ($\frac{\sigma}{E} \sim \frac{1}{\sqrt{E}}$) im GeV Bereich liegen kann. Außerdem liegen Rauschniveau (ENC) konstruktions- wie physikalisch (Detektorkapazität) bedingt um Welten auseinander. Im Vergleich zu den früher verwendeten analogen Aufbauten verspricht die XIA-Lösung, aufgrund der definierten Länge eines FIR-Filters und der Pipelining Möglichkeit, einen höheren Durchsatz. Ausserdem ist es ohne weiteres – mittels des User Moduls – möglich die DSP-Software um eigene Funktionalitäten zu erweitern. Zur Zeit kann das Modul DGF-4C allerdings noch nicht vollständig ausgereizt werden, da dieses gemeinsam mit älteren Experimentaufbauten verwendet wird. Dadurch wird die Pipelining Fähigkeit vermutlich nicht zum Einsatz kommen.

Anhang A

Steuerung des User-DSP-Codes

'Das einzige Problem am Nichtstun ist, daß man nie weiß, wann man fertig ist.'

Die Firma XIA hat, wie im Hauptteil beschrieben, ein Interface bereitgestellt, welches zum Originalcode hinzugelinkt werden kann. Dem Usercode stehen zweimal 16 Variablen zur Verfügung, die zur Ein- und Ausgabe von Parametern genutzt werden können.

Die Formatangaben sind folgendermaßen zu verstehen: $m.n$ bedeutet, dass m Bits zur Darstellung des ganzzahligen Anteils und n Bits zur Darstellung des gebrochenzahligen Anteils verwendet werden. Am Beipiel des Format 8.8 ohne Vorzeichenbit sei dies kurz erläutert.Die Zahl 0000101010110000 im 8.8 Format entspricht dem Wert $0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} + 0 \cdot 2^{-5} + 0 \cdot 2^{-6} + 0 \cdot 2^{-7} + 0 \cdot 2^{-8} = 10.6875$ Ähnlich wird für beliebige Formate $m.n$ verfahren.

Zur Darstellung negativer Zahlen im Binärformat sind zwei Konventionen üblich: 1er und 2er Komplement. Den negativen Wert einer positiven Zahl erhält man im 1er Komplement, indem man die postive Zahl invertiert. Somit zeigt ein gesetztes MSB (Most significant Bit) an, dass es sich um eine negative Zahl handelt. Am Beispiel des 4 Bit Wort 0101 wird dies kurz vorgeführt. Die Binärzahl 0101 entspricht dem Dezimalwert 5 (keine Nachkommmastellen). Die Darstellung der Dezimalzahl -5 im 1er Komplement erfolgt durch Invertierung und man erhält 1010. Damit diese Zahl nicht als +10 interpretiert wird, muss das Zahlenformat angegeben werden. Der verfügbare Raum an Binärzahlen wird auf die positiven und negativen Zahlen verteilt, ausserdem ist die Null zweimal vorhanden, nämlich als 0000 und als 1111.

Das 2er Komplement erhält man durch Invertierung und anschliessende Addition von Eins, d.h. zum 1er Komplement wird Eins addiert. Dadurch ist die Null nicht mehr doppelt vorhanden. Die Dezimalzahl -5 aus dem obigen Beispiel berechnet sich wie folgt: 0101 $\xrightarrow{1er}$ 1010 $\xrightarrow{+1}$ 1011.

Die aktuelle Belegung der Variablen durch den Usercode ist in den Tabellen A.1 und A.2 gezeigt. Eine Beschreibung befindet sich im Hauptteil.

Variablenname	Belegung	Format	2er Komplement
USERIN00	ln(Baselinelänge)	16.0	nein
USERIN01	Analyselänge für Core	16.0	nein
USERIN02	Sigma auf Baseline	16.0	ja
USERIN03	Energie cutoff für Segmente	16.0	ja
USERIN04	Analyselänge für Segmente	16.0	nein
USERIN05	unbenutzt	-	-
USERIN06	unbenutzt	-	-
USERIN07	unbenutzt	-	-
USERIN08	Kölnalgorithmus Ein-/Ausschalter	16.0	nein
USERIN09	Segmentanalyse Ein-/Ausschalter	16.0	nein
USERIN10	Coreanalyse Ein-/Ausschalter	16.0	nein
USERIN11	unbenutzt	-	-
USERIN12	unbenutzt	-	-
USERIN13	unbenutzt	-	-
USERIN14	unbenutzt	-	-
USERIN15	unbenutzt	-	-

Tabelle A.1: Belegung der Eingabevariablen des User Moduls

Variablenname	Belegung	Format	2er Komplement
USEROUT00	USERIN08	16.0	nein
USEROUT01	Startzeit des Pulses	8.8	nein
USEROUT02	Ende des Pulses	8.8	nein
USEROUT03	Position des Steepest Slope	8.8	nein
USEROUT04	Position des Steepest Slope	16.0	nein
USEROUT05	Adresse des Steepest Slope Buffers	16.0	nein
USEROUT06	Länge des Steepest Slope Buffers	16.0	nein
USEROUT07	Adresse des Buffers für Kanal 1	16.0	nein
USEROUT08	Länge des Buffers für Kanal 1	16.0	nein
USEROUT09	Adresse des Buffers fuer Kanal 2	16.0	nein
USEROUT10	Länge des Buffers für Kanal 2	16.0	nein
USEROUT11	Adresse des Buffers für Kanal 3	16.0	nein
USEROUT12	Länge des Buffers für Kanal 3	16.0	nein
USEROUT13	Bit 0 gesetzt, bedeutet gutes Event	16.0	nein
USEROUT14	Anzahl der guten Events	16.0	nein
USEROUT15	Anzahl der schlechte Events	16.0	nein

Tabelle A.2: Belegung der 16 Ausgabevariablen des User Moduls

Anhang B

Programmablaufdiagramme

> 'My instinct was to win, eliminate anyone who is in competition, destroy my enemy, and move on without any kind of hesitation at all.'
> Arnold Schwarzenegger

Die folgenden Programmablaufdiagramme wurden durch reverse engineering gewonnen. Da die mit den DGF-4C Modulen mitgelieferten Beschreibungen dürftig sind, können einige Diagramme eventuell von Nutzen sein und den Einstieg in eigene Analysen des XIA Codes erleichtern. Bevor nicht der gesamte Aufbau des Moduls bekannt ist (Hiermit meine ich vor allem den FPGA Code. Hierzu muß erst herausgefunden werden, was die einzelnen Programmierinformationen für den XILINX Spartan bedeuten.), kann leider keine vollständige Analyse stattfinden. Es soll hier darauf hingewiesen werden, dass der genaue Aufbau des Moduls DGF-4C unrelevant wäre, wenn exakt beschrieben wäre wie das Modul im Prinzip funktioniert, welche Signale es wann liefert und wie man es einsetzt. Schließlich interessiert sich niemand für den exakten Aufbau des MINIBALL Detektors, hauptsache er funktioniert. Letzteres kann leider von dem Modul DGF-4C nicht immer behauptet werden.

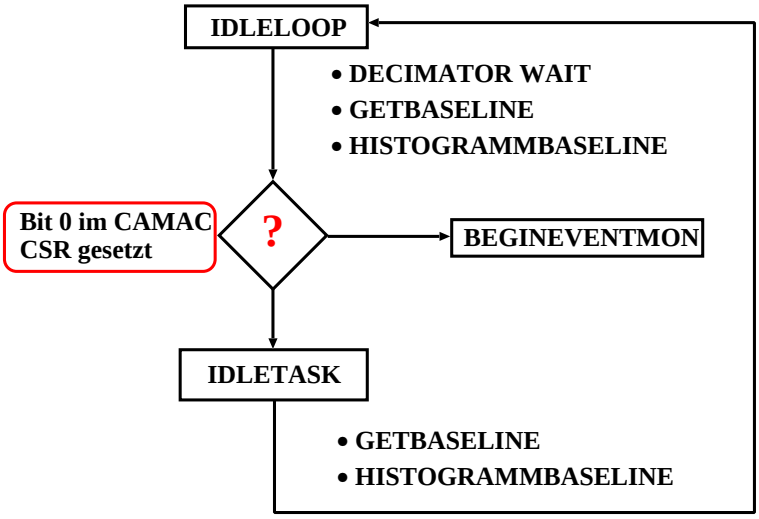

Abbildung B.1: Die Überwachungsschleife. Solange kein Run gestartet wurde, durchläuft der DSP Code diese Schleife. Die Aufgabe diese Codes ist die ständige Messung von Baselinewerten und die Abfrage des CAMAC Control und Status Registers. Ist in diesem das Bit 0 gesetzt, so wurde über den CAMAC Bus der Befehl zum Start eines Runs gegeben. In diesem Falle verläßt der Code die Idle-Schleife.

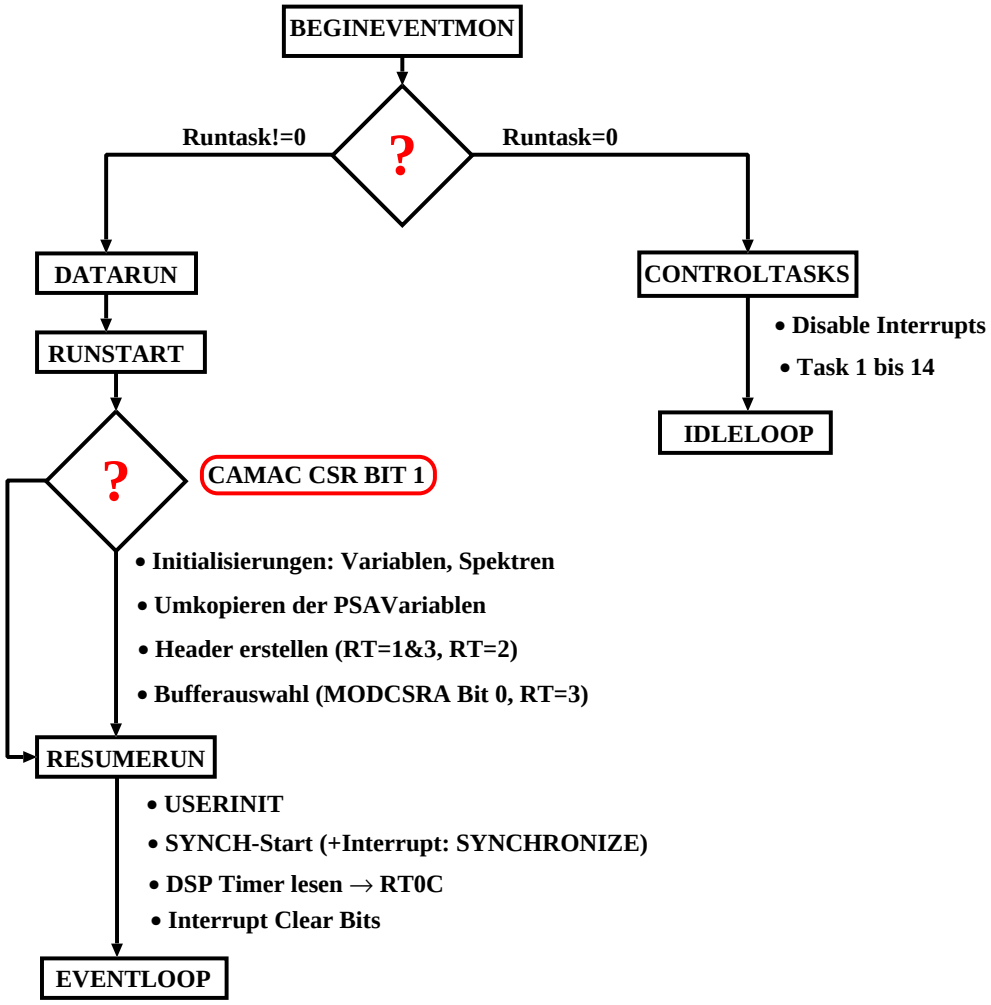

Abbildung B.2: Der Neustart eines Runs. Bevor der DSP die eigentliche Datenaufnahme startet, muss er zuerst feststellen, um welchen Runtyp es sich handelt. Handelt es sich um einen Controltask, so wird der entsprechende ausgeführt und anschließend wieder in den Idle Zustand zurückgekehrt. Sollen andererseits Daten aufgenommen werden, so muss überprüft werden, ob es sich um einen Neustart oder die Fortsetzung (Resume) eines Runs handelt. Handelt es sich um einen Resume Run (CAMAC CSR Bit 1 nicht mehr gesetzt) so kann der Code die Initialisierungen umgehen.

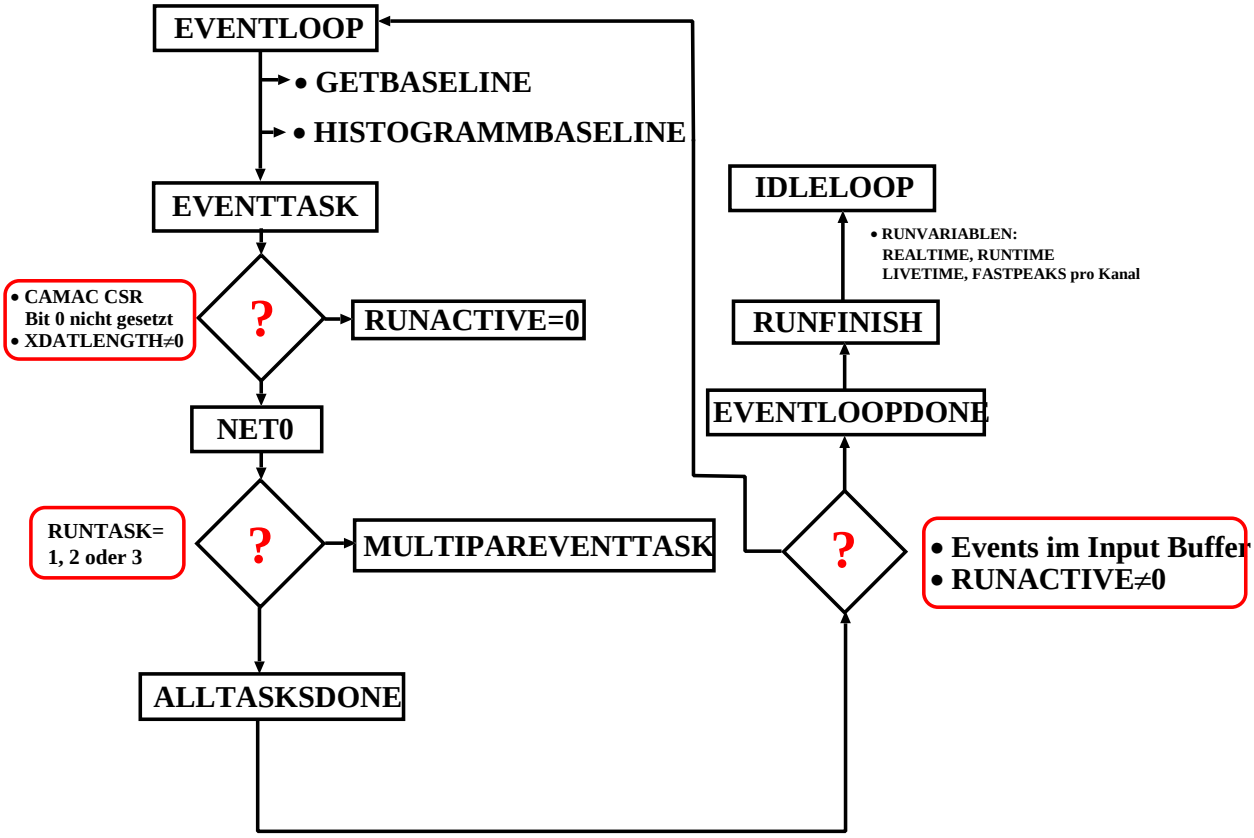

Abbildung B.3: Die Ereignisschleife. Interessanterweise ist die erste Aufgabe der Eventschleife, die Messung eines Baselinewertes. Falls kein Run gestartet wurde (CAMAC CSR Bit 0) und auch keine Userdaten im Eingangsbuffer stehen (XDATLENGTH) so wird der Run für beendet erklärt. Ansonsten wird nachgeschaut, um welche Runtyp es sich handelt und direkt in die Routine zur Verarbeitung der Eventdaten gesprungen. Trifft dies nicht zu, beginnt der Prozeß von vorne, falls sich Daten im Eventbuffer befindet oder ein Run gestartet wurde. Ansonsten wird der Run beendet und das Modul geht in den Stand-By Modus.

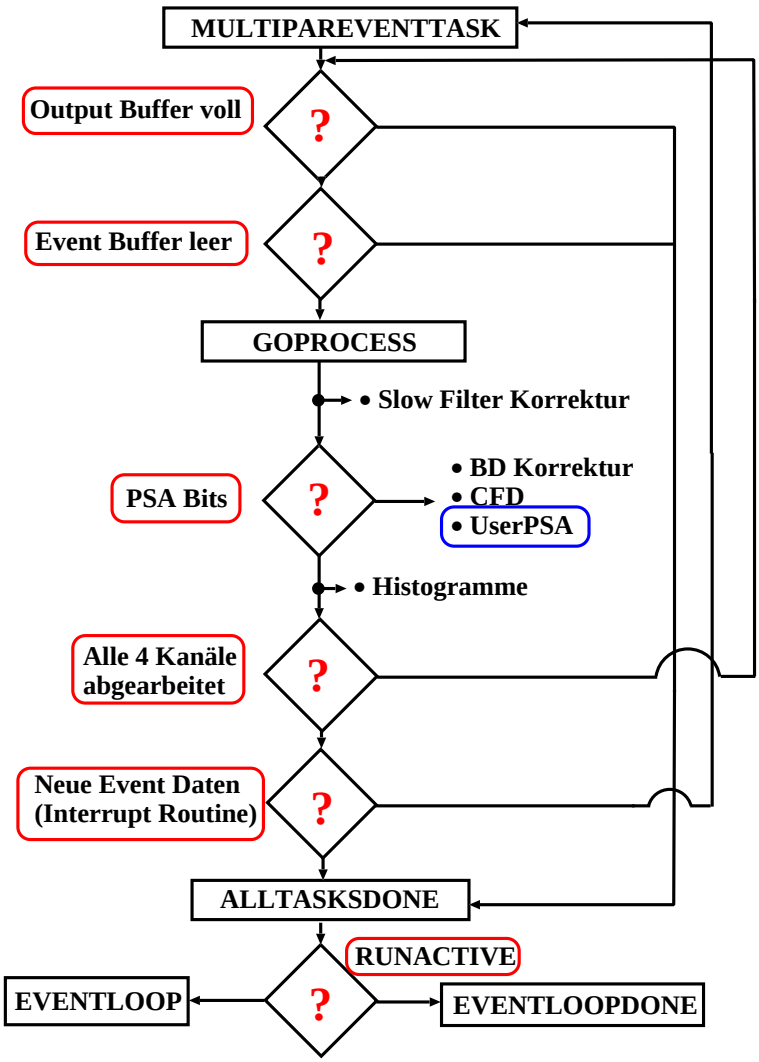

Abbildung B.4: Die Verarbeitung der Eventdaten. Zuerst prüft der Code, ob überhaupt genug Platz für die Eventdaten vorhanden ist. Ist dies der Fall und befinden sich Eventdaten im Eingangsbuffer, so beginnt der Code mit der Korrektur der Filterwerte. Anschliessend werden die gewünschten Pulsformanalysen ausgeführt. An dieser Stelle verzweigt der XIA Code in den Hauptteil des Usercodes: USERCHANNEL. Dies wird solange wiederholt bis alle Kanäle abgearbeitet sind und sich ausserdem keine Eventdaten mehr im Eingangsbuffer befinden. Sind alle Daten bearbeitet, verzweigt der Code in die Eventschleife, falls die Datenaufnahmen immer noch läuft, ansonsten beendet der Code den Run.

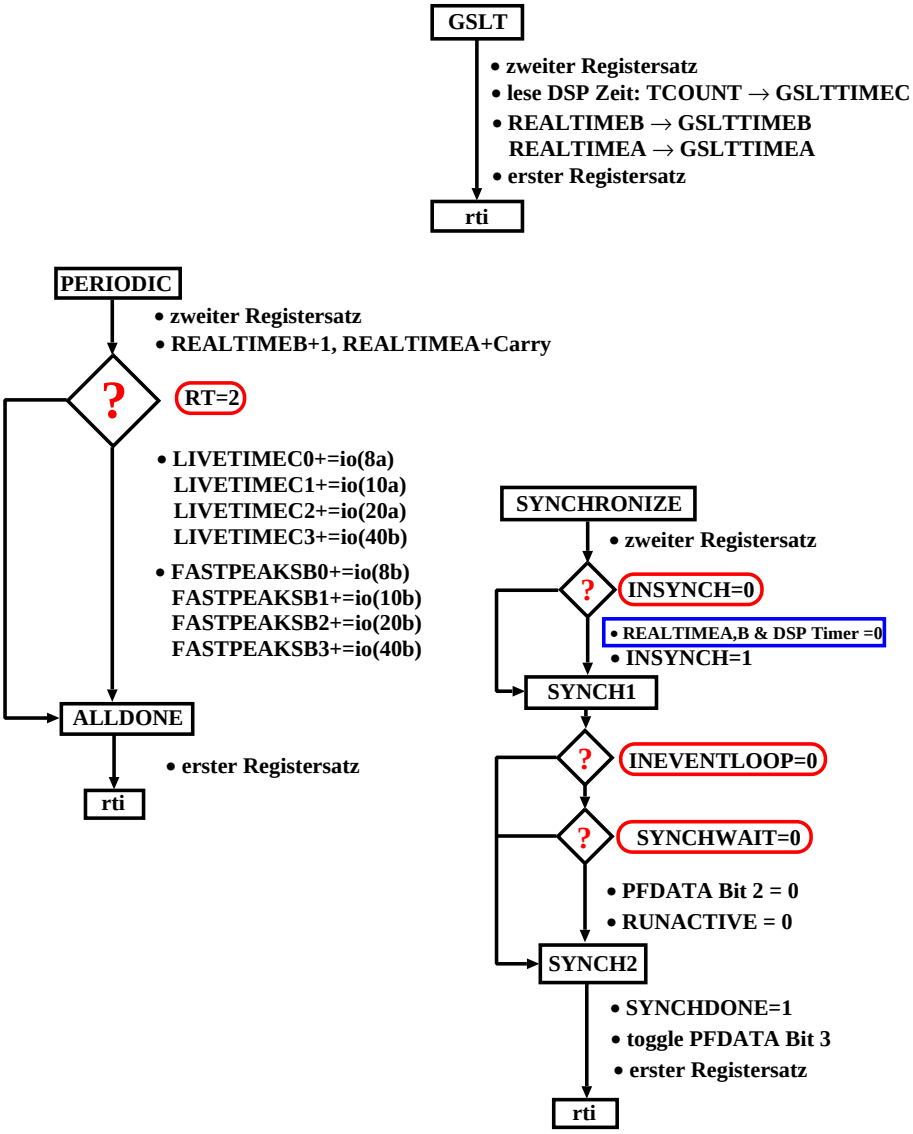

Abbildung B.5: Die Interrupt-Routinen, Teil 1. Kennzeichned für eine Interrupt-Routine ist die Verwendung des zweiten Registersatzes. Die GSLT Routine liest den DSP Timer und benutzt diesen zusammen mit den oberen 32 Bit der Echtzeit (REALTIME), um die Zeit des Second Level Triggers festzulegen. Der DSP Timer kann ebenfalls einen Interrupt erzeugen. Alle 1.6384 ms ($2^{16} \cdot 0.025\ \mu$s) wird die Routine PERIODIC aufgerufen. Die oberen 32 Bit der Echtzeit werden um Eins erhöht, da der DSP Timer einmal übergelaufen ist. Mit Ausnahme des Fast List Mode, wird für alle Datenaufnahmenmodi die Livetime und die Anzahl der Fast Trigger Events aus den FPGAs gelesen. Die letzte Interruptroutine wird benutzt, um das gemeinsame Starten der Karten zu steuern. Sie setzt beispielsweise die REALTIME und den DSP Timer zurück. In der Routine BEGINEVENTMON hat der DSP SYNCHDONE auf Null gesetzt. Danach wartet er in einer Schleife darauf, dass die Interruptroutine die Variable SYNCHDONE auf Eins setzt (sofern SYNCHWAIT != 0). Die Interruptroutine SYNCHRONIZE, die bei jedem 1 → 0 Übergang am Synch-Eingang des DGF-4C Moduls aufgerufen wird, setzt als eine der letzten Instruktionen SYNCHWAIT auf Eins. Kehrt der DSP danach zum normalen Programm zurück, so setzt er die Datenaufnahme fort.

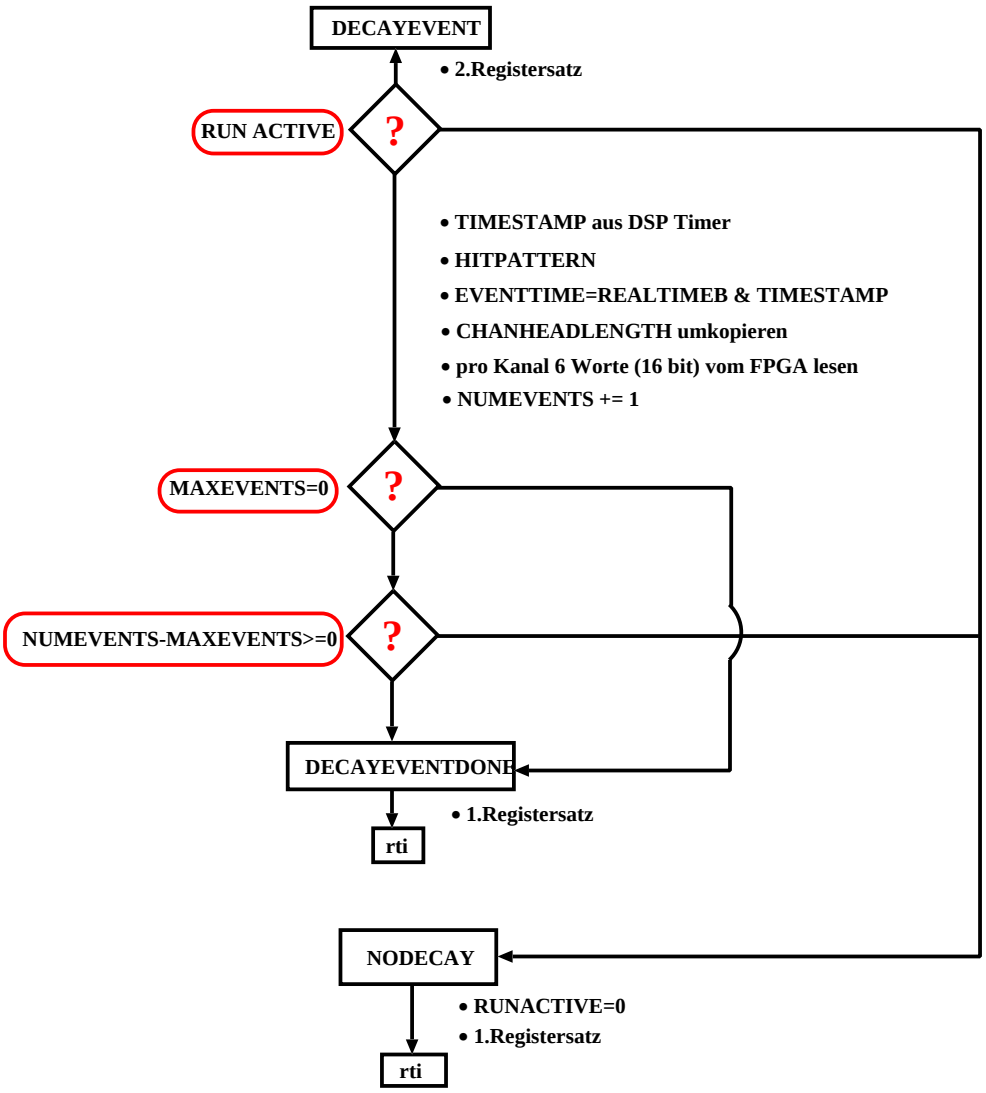

Abbildung B.6: Die Interrupt-Routinen, Teil 2. Kennzeichned für eine Interrupt-Routine ist die Verwendung des zweiten Registersatzes. Da diese Routine nur die FPGA Daten (6 Worte) ausliest und keine FIFO Daten, wird sie vermutlich für den fast list mode benutzt. Die unteren 16 Bit des Eventzeitpunkts werden aus dem DSP Timer ausgelesen. Die oberen 16 Bit sind durch die REALTIME gegeben. Ebenfalls ausgelesen wird, welche Kanäle getriggert haben. Ist die erforderliche Anzahl von Ereignissen aufgenommen worden (der Ereigniszähler wird jedesmal erhöht), so wird der Run beendet.

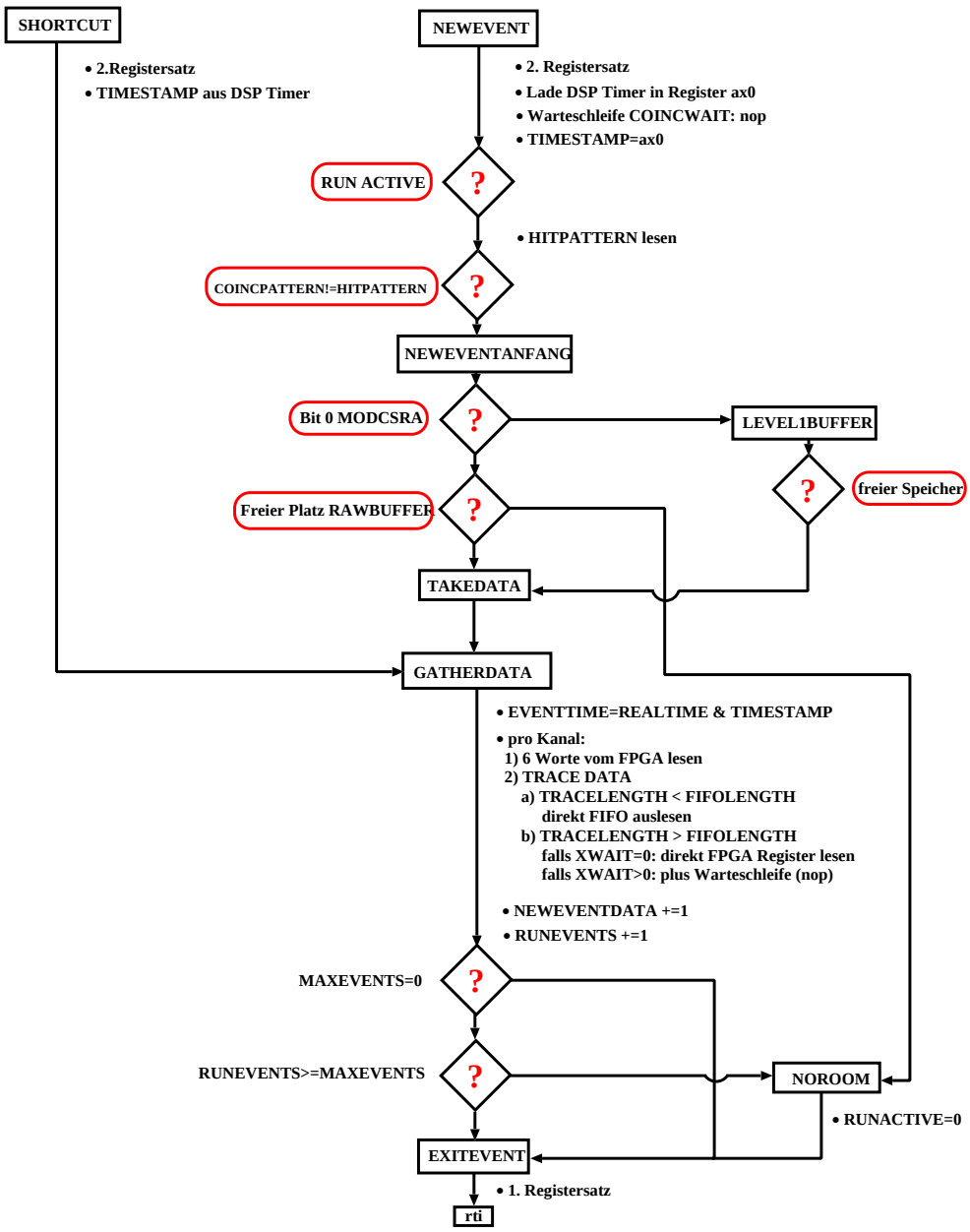

Abbildung B.7: Die Interrupt-Routinen, Teil 3. Kennzeichned für eine Interrupt-Routine ist die Verwendung des zweiten Registersatzes. Hiermit findet vermutlich die Datenaufnahme im list mode statt. Bevor der DSP mittels der COINCWAIT-Schleife auf weitere Event Trigger wartet, bestimmt er den Event Zeitpunkt mittels des DSP Timers. Nach der Warteschleife wird das Hitpattern ausgelesen. Danach wird bestimmt, wohin die Eventdaten kopiert werden sollen und ob überhaupt genug freier Speicherplatz vorhanden ist. Nachdem die Eventzeit mittels der aus dem DSP Timer gelesenen Zeit und der REALTIME bestimmt wurde, werden die Eventdaten ausgelesen. Dazu werden pro Kanal 6 Worte aus den FPGAs gelesen und die Tracedaten aus den FIFOs (TRACELENGTH < FIFOLENGTH) bzw. mittels eines FPGA Registers (TRACELENGTH > FIFOLENGTH) erhalten. Ist die benötigte Anzahl an Events aufgenommen worden, so wird der Run beendet.

Anhang C

Der User Quellcode

```
/*****************************************************************
* Register usage:
*   The user routines may use all computational registers without
*   having to restore them. However, the secondary register set
*   cannot be used, because the XIA interrupt routines use these.
*   The usage of the address registers I0..I7 and the associate
*   registers M0..M7, and L0..L7 is subject to restrictions.
*   These are listuser.dsped below for the various routines.
*   The associate registers L,M are preset and guaranteed as follows:
*   L0..L7 = 0
*   M0 = 0; M1 = 1; M2 = -1;
*   M4 = 0; M5 = 1; M6 = -1;
*   M3 and M7 have no guaranteed values.
*   UserBegin, UserRunInit, and UserRunFinish:
*   No further restrictions, but user code must leave the associated
*   registers listed above in exactly this state when exiting.
*   UserChannel:
*     I5,I6,I7  may not even temporarily be overwritten, because
*     L5,L6     there are interrupt functions which depend on the
*     M0,M1,M2  content of these registers.
*     M4,M5,M6
*     I0,I1,I3  These may be altered, but must be restored on exit.
*     I4
*     L0,L1,L2,L3
*     L4,L7
*     I2        May be altered and need not be restored
*     M3,M7
*   UserEvent:
*     I5,I6,I7  may not even temporarily be overwritten, because
*     L5,L6     there are interrupt functions which depend on the
*     M0,M1,M2  content of these registers.
*     M4,M5,M6
*     I3        These may be altered, but must be restored on exit.
*     L0,L1,L2,L3
*     L4,L7
*     I0,I1,I2,I3 May be altered and need not be restored
*     I4
*     M3,M7
*****************************************************************/

.MODULE/RAM/SEG=UserProgram user;

.ENTRY UserBegin;
.ENTRY UserRunInit;
.ENTRY UserChannel;
.ENTRY UserEvent;
.ENTRY UserRunFinish;

.EXTERNAL UserBeginReturn;
.EXTERNAL UserRunInitReturn;
.EXTERNAL UserChannelReturn;
.EXTERNAL UserEventReturn;
.EXTERNAL UserRunFinishReturn;

.INCLUDE <INTRFACE.INC>; /* interface provided by XIA */

.VAR/DM/SEG=UserData saveI0, saveI1, saveI3, saveI4;

.VAR/DM/SEG=UserData Baseline16;          /* Variablen */
.VAR/DM/SEG=UserData Baseline12;
.VAR/DM/SEG=UserData BaseShift16;
.Var/DM/SEG=UserData BaseShift12;
.VAR/DM/SEG=UserData BaseLen;
.VAR/DM/SEG=UserData AnaLen;
.VAR/DM/SEG=UserData SegAnaLen;
.VAR/DM/SEG=UserData Sigma;
.VAR/DM/SEG=UserData EnCutOff;
```

```
.VAR/DM/SEG=UserData Start;              /* startzeit aus core, falls kein coresignal -> segment mit treffer */
.Var/DM/SEG=UserData RawStart;
.VAR/DM/SEG=UserData Ende;

.VAR/DM/SEG=UserData MaxSteepPos0;       /* ergebnis core psa */
.VAR/DM/SEG=UserData RawSlopePos;
.VAR/DM/SEG=USerData RawSlopeAdr;

.VAR/DM/SEG=UserData MaxPeakVal1;        /* ergebnisse segment psa */
.VAR/DM/SEG=UserData MaxPeakVal2;
.VAR/DM/SEG=UserData MaxPeakVal3;

.VAR/DM/SEG=UserData GundKoeln;
.VAR/DM/SEG=UserData SegAnaOn;
.VAR/DM/SEG=UserData CoreAnaOn;

.VAR/DM/SEG=UserData GoodEventNum;        /* Run/Debug-Info */
.VAR/DM/SEG=UserData BadEventNum;
.VAR/DM/SEG=UserData GoodBadSwitch;       /* 0-> good event, 1=bad event */


.VAR/DM/SEG=UserData SteSloBuffer[64];    /* Buffer fuer zweite Ableitung */
.VAR/DM/SEG=UserData SegBuffer1[64];
.VAR/DM/SEG=UserData SegBuffer2[64];
.VAR/DM/SEG=UserData SegBuffer3[64];

.VAR/DM/SEG=UserData MSTAT_RMBR;
.VAR/DM/SEG=UserData mr0_RMBR;
.VAR/DM/SEG=UserData mr1_RMBR;


/*-----------------------------------------------------------------------
+ UserBegin is called once after power up or reboot, after all
+ memory and variables have been initialized, but before
+ the trigger/filter FPGAs and the DACs have been programmed.
-----------------------------------------------------------------------*/
UserBegin:
   ax0=0;
   dm(GoodEventNum)=ax0;
   dm(BadEventNum)=ax0;
JUMP UserBeginReturn;

/*-----------------------------------------------------------------------
+ UserRunInit is called once during the run initialization phase
-----------------------------------------------------------------------*/
UserRunInit:
   i0=^UserIn;             /* 1. nuetzliche Koeff. berechen */
   ay0=dm(i0,m1);          /* 1.1 Baseline Messung */
   ax0=4;
   ar=ax0-ay0;
   dm(BaseShift12)=ay0;
   dm(BaseShift16)=ar;     /* Korrekturshift der Baselinesummation */
   ar=1;
   se=ay0;
   sr=lshift ar (lo);
   dm(BaseLen)=sr0;        /* einstellen der Baselineanalyse -> [0, BaseLen], in potenzen von 2 angeben */
   ax0=dm(i0,m1);          /* 1.2 Analyselaenge fuer Core */
   dm(AnaLen)=ax0;         /* AnalyseLaenge der Trace umsetzten z.B. 500ns->20  (*25ns) */
   ax0=dm(i0,m1);          /* 1.3 Sigma fuer Startfinder */
   dm(Sigma)=ax0;          /* Sigma fuer Baseline */
   ax0=dm(i0,m1);          /* 1.4 EnergieCutOff fuer Segmente */
   dm(EnCutOff)=ax0;
   ax0=dm(i0,m1);          /* 1.5 Analyselaenge fuer Segmente */
   dm(SegAnaLen)=ax0;
   ax0=dm(i0,m1);          /* 1.6 leer */
   ax0=dm(i0,m1);          /* 1.7 leer */
   ax0=dm(i0,m1);          /* 1.8 leer */
                           /* 2. IN-Variablen umkopieren zwecks Ablaufsteuerung */
   ax0=dm(i0,m1);          /* 2.1 Schalter fuer Reine Gund-Analyse bzw. mit Koeln-Algo */
   dm(GundKoeln)=ax0;      /* Umschalten zwischen CG-Methode, d.h. Hitpattern abhaengig, oder Dirks 1.Ordnung abschaetzung */
   ax0=dm(i0,m1);          /* 2.2 Segment-Analyse einschalten */
   dm(SegAnaOn)=ax0;       /* an/aus der segment psa */
   ax0=dm(i0,m1);          /* 2.3 Core Analyse ein */
   dm(CoreAnaOn)=ax0;      /* an/aus der core psa */
                           /* rest ist leer */
/*------------------------------ INIT ------------------------------------------------*/
   ax0=0;
   dm(Baseline16)=ax0;
   dm(Baseline12)=ax0;
   dm(Ende)=ax0;
   dm(MaxSteepPos0)=ax0;
   dm(MaxPeakVal1)=ax0;
   dm(MaxPeakVal2)=ax0;
   dm(MaxPeakVal3)=ax0;
   dm(RawSlopePos)=ax0;
   dm(RawSlopeAdr)=ax0;
   dm(GoodBadSwitch)=ax0;
   dm(Start)=ax0;       /* Init des Startpunktes fuer Module ohne CoreSig. und Seg.Hit */
   dm(RawStart)=ax0;
JUMP UserRunInitReturn;
```

```
/*--------------------------------------------------------------------
+  UserChannel is called for every event, and for every channel
+  with an entry in the read/hit pattern, and for which bit 0
+  in its ChanCSRB has been set.
--------------------------------------------------------------------*/
UserChannel:
   DM (saveI0)   = I0;
   DM (saveI1)   = I1;
   DM (saveI3)   = I3;
   DM (saveI4)   = I4;

   call Base;                  /* Baseline fuer Core/Seg. bestimmen */
   ax0 = dm(ChanNum);
   ar = pass ax0;
   if eq jump SteepestSlope1;  /* Kanal 0 -> RadiusPSA */
   ar=ar-1;
   if eq jump Winkelanalyse1;  /* Kanal 1 -> SegmentPSA */
   ar=ar-1;
   if eq jump Winkelanalyse2;  /* Kanal 2 -> SegmentPSA */
   ar=ar-1;
   if eq jump Winkelanalyse3;  /* Kanal 3 -> SegmentPSA */
JUMP UserChannelReturn;        /* zur Sicherheit- > ENDE*/


/*****************************************************************************/
/* Ermittele Baseline aus den ersten 2^x Samples -> leichte Division durch Shift*/
/* 12bit ADC -> Format: 12.4 moeglich ohne Verluste mit 16 Bit -> max. Stellen  */
/*****************************************************************************/
Base:
   i0=dm(ATstart);            /* Startadresse in i0 laden */
   cntr=dm(BaseLen);          /* Counter fuer Schleife setzen */
   af=pass 0, ax1=dm(i0,m1);  /* feedback register=null fuer akkumulation, lade ersten ADC-Wert*/
   do addloop until ce;                /* die ersten xx Samples addieren */
addloop:   af=ax1+af, ax1=dm(i0,m1);   /* add and load */
   ar=pass af;                /* Ergebnis uebergeben */
   se=dm(BaseShift16);        /* lese baseshift ein */
   sr=lshift ar (lo);         /* shifte ergebnis so, dass aus xx.x (16bit) ein 12.4 Format wird d.h. die */
                              /* wichtigen Daten 12 high bits und kommastellen low 4 bits */
   dm(Baseline16)=sr0;        /* Ergebnis sichern (FORMAT 12.4) */
   ar=dm(BaseShift12);
   ar=-ar;                    /* diesmal in die andere richtung... */
   se=ar;
   ar=pass af;                /* ergebnis wieder uebergeben */
   ar=ar+8;                   /* Runden: +0.5 */
   sr=lshift ar (lo);         /* Runden: */
   dm(Baseline12)=sr0;        /* Ergebnis sichern, Format 12.0 (mit runden) */
rts;

/*************************************/
/* Bestimme Start-/Endzeitpunkt      */
/* vorzugsweise aus Coresignal       */
/* Finde SteepestSlope-Position (CORE) */
/*************************************/
StartFinder:
   cntr=dm(AnaLen);            /* Loopanzahl = Analyse-Laenge */
   i0=dm(ATstart);             /* Startadresse laden */
   m3=dm(BaseLen);             /* Baseline-Laenge einlesen */
   modify(i0,m3);              /* Startadresse=Tracestart + BaselineLaenge */
   m3=-2;
   modify(i0,m3);              /* Startadresse=Tracestart + BaselineLaenge -2 */
   my0=dm(i0,m1);              /* lade ADC(n-2) -> SteepestSlope von ADC(n) */
   sr0=1;                      /* Coeff */
   sr1=2;                      /* Coeff */
   i1=^SteSloBuffer;           /* i1 zeigt auf SteepestSlopeBuffer*/
   ay1=0;                      /* init ay1 */
   mr=sr0*my0 (uu), my0=dm(i0,m1);               /* multipliziere ADC(n-2), lade ADC(n-1) , modify auf ADC(n)*/
   do differloop until ce;                       /* 2.Ableitung bilden und SteepestSlope suchen (irgendein Punkt mitten drin) */
            mr=mr-sr1*my0 (uu), my0=dm(i0,m2);   /* Accumulate, lade ADC(n), modify auf ADC(n-1) -> neuer ADC(n-2) !!! */
            mr=mr+sr0*my0 (uu), my0=dm(i0,m1);   /* Accumulate, lade neuen ADC(n-2), modify auf ADC(n-1) */
            ar=mr0-ay1, dm(i1,m1)=mr0;           /* ar=2.abl - akt minimum, Sichere Ergebnis im SteSloBuffer */
            if lt call negativeslope;            /* wenn neue 2.Abl. < alte 2.Abl. -> merken neues Min. */
differloop:   mr=sr0*my0 (uu), my0=dm(i0,m1);    /* multipliziere ADC(n-2), lade neuen ADC(n-1), modify auf ADC(n) */
   ay0=dm(BaseLen);            /* addiere Baselinelaenge */
   ax1=dm(RawSlopeAdr);        /* Adresse im SteSloBuffer [+1 (modify)] */
   ar=ax1+ay0;                 /* korrigiere Position -> match charge position */
   i2=^SteSloBuffer;           /* lade Startadresse */
   ay1=i2;
   ar=ar-ay1;                  /* abs.Adr -> rel.Adrresse: insge. SteSloAdr(+1)-SteSloBufferAdr+Baselen */
   i0=dm(ATstart);             /* lade PulsStartAdr. */
   dm(RawSlopePos)=ar;         /* speichere rel Adr. , korrekt wegen modify !! */
   m3=ar;
   modify(i0,m3);              /* mache rel.Adr. zu Adr im Ladungspuls,MaxPos uebergebe -> Pos=(i1-^SteSloBuffer) + BaseLen + ATstartAddr */
   ay0=dm(i0,m1);              /* puls(n), modify auf n+1 */
   ax0=dm(i0,m1);              /* puls(n+1), modify auf n+2 */

   do suchenull until le;      /* Suche ENDE, nach vorne (m1=1), im LADUNGS-Puls, Abbbruch:puls(n+1)<=puls(n), KEIN ANSTIEG MEHR */
            ar=ax0-ay0, ay0=ax0;        /* bilde: ADC(n+1)-ADC(n), ADC(n+1)->ADC(n) */
suchenull:   ax0=dm(i0,m1);             /* lade puls(n+2), modify auf puls(n+3) */
   ar=i0;                      /* uebergebe Zeiger auf naechste ADC-Adresse */
   ay0=dm(ATstart);            /* lade abs.Startadresse */
   ar=ar-ay0;                  /* relative Position = Position - Addr.Offset - 3 (wegen modify usw.) */
   ar=ar-3;                    /* dito */
```

```
   sr=lshift ar by 8 (lo);     /* bringe ins passende 8.8 Format (allerdings ohne nachkommastellen zu haben */
   dm(Ende)=sr0;               /* hat (zur Zeit) nicht viel Sinn hier die Aufloesung zu optimieren */
   ax0=dm(RawSlopePos);        /* lade rel. Maxstromposition */
   ar=ax0+ay0;                 /* Ausgangsposition berechnet, ay0=ATstart */
   i0=ar;
   ar=dm(Baseline12);          /* lade Baselinewert */
   ay0=dm(Sigma);
   ar=ar+ay0;                  /* Sigma addieren ... */
   ay0=ar;
   ax0=dm(i0,m2);
   do suchenull1 until lt;                 /* Suche START: nach hinten (m2=-1), LADUNGS-Puls, Abbruch: ADC<=Baseline */
suchenull1:   ar=ax0-ay0, ax0=dm(i0,m2);   /*ar=ADC-Baseline, lade neuen ADC-Wert */
/* abbruch: testet abbruch gleichzeitig mit berechnung, wirkt erst einen Takt spaeter !! wenn ergebis fertig */
/* bsp: adc(17)- baseline < ,lade 16 modigy auf 15, kein abbruch da zu spaet; adc(16)-baseline<, lade 15 zeige auf 14 -> jetzt erst abbruch wegen < bei adc(17) !!*/
   call interloop;            /* lin. Interpolieren */
   ax0=dm(Ende);
   ay0=dm(Start);
   ar=pass ay0;
   if le call error;
   ar=ax0-ay0;
   if le call error;         /* Ende vor Start ??? */
rts;

negativeslope:
   ay1=mr0;                    /* sichere Wert von Steepest Slope als neuen Vergleichswert */
   dm(RawSlopeAdr)=i1;         /* sichere Adresse */
rts;


rts;


/*********************************************/
/* lineare Interpolation des Startzeitpunkts */
/*********************************************/
interloop:
   m3=4;
   modify(i0,m3);           /* Zeiger auf ADC > Baseline ,d.h. der ADC-Wert der zur Interpolation nach Baseline benutzt wird */
   dm(Start)=i0;            /* merke dir die GROBE StartZeit */
   ay0=dm(i0,m1);           /* ADC-Wert>Baseline (Modify rueckgaengig und eins zurueck )-> Steigung */
   ax0=dm(i0,m1);           /* naechst groesseren ADC-Wert einlesen */
   ar=ax0-ay0;              /* Steigung bestimmen ( *1/25ns faellt weg), sollte Positiv sein-> 16.0 UNSIGNED */
   if lt ar=abs ar;
   ax0=ar;                  /* sichere STEIGUNG -> NENNER in ax0 -> FORMAT:16.0 UNSIGNED */
   ax1=dm(Baseline12);      /* Baselinewert auslesen -> FORMAT: 16.0 */
   ar=ax1-ay0;              /* ZAEHLER=ar=Baseline (16.0)-ADC(n) (16.0), Ergebnis: SIGNED (13bit) */
   ar=abs ar;               /* ergebnis:  unsigned 12 bit */
   sr=lshift ar by 15 (lo); /* Zaehlerformat: 17.15 (32 bit) */
   ay0=sr0, af=pass sr1;    /* uebergebe Zaehler , aq=0; y=ax+b -> x=(y-b)/a */
/* UNSIGNED DIVISION -> evtl noch Korrekturen einfuegen */
   astat=0;
   divq ax0; divq ax0;      /* 17.15/16.0 = 2.14 */
   divq ax0; divq ax0;
   divq ax0; divq ax0;
   divq ax0; divq ax0;
   divq ax0; divq ax0;      /* bis hierhin: FORMAT: 2.8 unsigned -> sollte gen"ugen */
   ax0=1023;
   ar=ax0 AND ay0;          /* sorge fuer 8.8 FORMAT */
   ay0=ar;
   ay1=dm(ATstart);         /* lade Startadresse */
   ax0=dm(Start);           /* lade GROBE Adresse des ADC-Samples von dem aus interpoliert wurde */
   ar=ax0-ay1;              /* position(in samples) = akt&grobe Pos - StartPos */
   dm(RawStart)=ar;         /* ohne Nachkomma Start */
   sr=lshift ar by 8 (lo);  /* bringe SamplePosition in 8.8 Format -> max. Tracelen 256 !!! (worst case) */
   ar=sr0-ay0;              /* Pos(exakt)=pos(ADC(n))-interpol_pos (gemeinsames FORMAT: 8.8) */
/* wegen unsigned division und absolutwert bildung -> subtrahieren der Korrektur */
   dm(Start)=ar;            /* speichere EXAKTE StartZeit */
rts;


/*********************************************/
/* CORE-PSA: TimeToSteepestSlope bestimmen   */
/* 1. Startzeit -> 2. SteepestSlopeZeitpunkt */
/*********************************************/
SteepestSlope1:
   ax0=dm(CoreAna0n);          /* spaetes abbrechen, falls keine CoreAnalyse gewuenscht */
   ar=pass ax0;
   if eq JUMP UserChannelReturn;
   call StartFinder;           /* bestimme Startzeit/Endzeit und finde SteepestSlopePosition */
jump SteepestSlope2;


/*******************************/
/* Ermittele SteepestSlope     */
/*******************************/
SteepestSlope2:
   ar=dm(RawSlopePos);     /* Position von SteSlo-ADC(n+1) [in Samples] */
   i2=dm(RawSlopeAdr);          /* Position von SteSo [Adresse] */
   call quadinterpol;      /* Quadratisch interpolieren -> Rueckgabe im Format 8.8 */
   ax0=ay0;                /* uebernehme ergebnis */
   dm(MaxSteepPos0)=ay0;
   ay0=dm(Start);          /* FORMAT: 8.8 */
```

```
   ar=ax0-ay0;             /* SteepPos-Start */
   if lt call quaderr;
   dm(UretVal)=ar;         /* Rueckgabe: TiToSteSlo */
   I0 = DM (saveI0);
   I1 = DM (saveI1);
   I3 = DM (saveI3);
   I4 = DM (saveI4);
JUMP UserChannelReturn;


quaderr:
   ax0=0;
   ar=ax0;
   ax0=1;
   dm(GoodBadSwitch)=ax0;
rts;


/*****************************************/
/*                                       */
quadinterpol:
/* in i2 die Adresse von ADC(n+1)         */
/* in ar n+1, also die "relative adresse" */
/* FORMAT des Ergebnisses: 8.8 in ay0     */
/*****************************************/
   mx0=2;
   ay1=dm(RawSlopePos);             /* merke (n+1) in ax1 fuer Nenner */
   ar=ar+ay1, my1=dm(i2,m2);        /* ar=n+1+n+1 , my1=ADC(n+1), modify auf n */
   ar=ar-3; my0=dm(i2,m2);          /* ar=n+n-1 , my0=ADC(n) ; modify auf ADC(n-1) */
   mr=ar*my1 (ss), ax1=my1;         /* -> mr=(n-1+n)*ADC(n+1), merke ADC(n+1) in ax1 fuer Nenner */
   if av jump error;                /* KORREKT bis HIER ! */
   ar=ar+1;     /* ar=n+n  */
   dm(mr0_RMBR)=mr0;
   dm(mr1_RMBR)=mr1;
   mr=mx0*my0 (ss);                 /* mr=2*ADC(n) (KORREKT) */
   my1=mr0;
   mr0=dm(mr0_RMBR);
   mr1=dm(mr1_RMBR);
   mr=mr-ar*my1 (ss), my1=dm(i2,m2); /* -> mr=mr-(n+n)*2*ADC(n), my1=ADC(n-1), modify auf n-2 */
   if av jump error;                 /* KORREKT!!!!!!!!!! -> Probleme mit mf !!!! */
   ar=ar+1;                         /* ar=n+n+1 */
   mr=mr+ar*my1 (ss), ay1=my1 ;     /* -> ZAEHLER=mr=mr+ADC(n-1)*(n+n+1), lade ADC(n-1) in ay1 */
   if av jump error;
jump shifting;                      /* shifte ins passende format */


/* Bringe Zaehler ins FORMAT 23.9 !! (MIT VZ -> signed division) */
shifting:
   sr=ashift mr1 by 9 (hi);          /* shifting a double precision number */
   sr=sr or ashift mr0 by 9 (lo);
   ay0=sr0;
   af=pass sr1;
jump divis;


error:                               /* es gab alu-overflow */
   ax0=1;
   dm(GoodBadSwitch)=ax0;
   ay0=0;
   af=pass ay0;
rts;


abbruch:                            /* Nenner = Null -> abbruch, ergebnis soll null sein*/
   ar=1;
   dm(GoodBadSwitch)=ar;
   ay0=0;
   af=pass ay0;
rts;


/* die eigentlich division */
divis:
/* uebergebe Nenner */
   ar=ax1+ay1;                      /* ar=ADC(n+1)+ADC(n-1) */
   mr=mx0*my0 (ss), my1=mx0;        /* mr=2*ADC(n), lade die 2 in my1 <- mf uebernehmen ??? */
   if av jump error;
   ay1=mr0;                         /* uebergebe 2*ADC(n) an ay1 */
   ar=ar-ay1;                       /* ar=ADC(n+1)+ADC(n-1)-2*ADC(n)=NENNER */
   if eq jump abbruch;
   mr=ar*my1 (ss);                  /* mr=Nenner(mit VZ)*2 */
   if av jump error;
   ar=mr0;                          /* uebergebe Ergebnis (max 16 bit ) an ar */
/* SIGNED DIVISON -> evtl. noch Korrekturen einfuegen */
   divs af,ar ; divq ar;            /* berechne VZ-> XOR */
   divq ar; divq ar;                /* 23.9 / 16.0 = 8.8 */
   divq ar; divq ar;
   divq ar; divq ar;
   divq ar; divq ar;
   divq ar; divq ar;
   divq ar; divq ar;
   divq ar; divq ar;
```

```
rts;


/*****************************************************************************/
/* Fuer Segmente ohne Energiedepos.    : Finde absolutes Maximum in der gesamten Trace  */
/* Fuer Segmente mit Energiedeposition : Finde Maximum ueber linear interpoliertem Puls */
/*****************************************************************************/
Winkelanalyse1:
   ax0=dm(SegAna0n);
   ar=pass ax0;
   if eq jump UserChannelReturn;    /* falls doch keine Segmentanalyse gewuenscht -> ende */
   ax0=dm(GundKoeln);
   ar=pass ax0;
   if ne jump Koelnanalyse1;         /* 1->benutze den Koeln-Methode, 0->benutze CG-Methode */
JUMP Gundanalyse1;



/*  CG/Koeln(OhneTreffer): Segment -> finde |KuhMax| */
Gundanalyse1:
   ay1=0;                   /* initilisiere alten Wert */
   i0=dm(ATstart);
   i1=^SegBuffer1;
   m3=dm(BaseLen);
   modify(i0,m3);           /* Startadresse nach i0 */
   ay0=dm(Baseline12);      /* lade Baseline ADC-Wert nach ay0 -> FORMAT 12.0 */
   ax0=dm(i0,m1);           /* lade 1.ADC-Wert nach sr0 */
   cntr=dm(AnaLen);         /* Counter=Analyse-Laenge */
   ar=ax0-ay0, ax0=dm(i0,m1);
   do cmploop1 until ce;
          if lt ar=abs ar;              /* bilde Absolutwert */
          af=ar-ay1, dm(i1,m1)=ar;      /* ar=neuer Wert - alter Wert(alles im 12.0 Format), erstelle buffer*/
          if ge call newmax;            /* ar>0, d.h. neuer Wert>alterWert (absolut)*/
cmploop1:  ar=ax0-ay0, ax0=dm(i0,m1);    /* ar=ADC(n)-Baseline , lade ADC(n+1) in ax0 */
   dm(MaxPeakVal1)=ay1;     /* Maximale Ladungspulshoehe sichern 16.0 !!!*/
   dm(UretVal)=ay1;         /* Rueckgabe: Ladungspulshoehe FORMAT: 16.0 !!!*/

   I0 = DM (saveI0);
   I1 = DM (saveI1);
   I3 = DM (saveI3);
   I4 = DM (saveI4);
JUMP UserChannelReturn;


/* Segment MIT Treffer:  |KuhMax| ueber Gerade */
Koelnanalyse1:
   ax0=dm(Energy);             /* FORMAT: 13.3, benutze als PeakHoehe */
   ay0=dm(EnCut0ff);
   ar=ax0-ay0;
   if le jump Gundanalyse1;    /* falls Segment nicht getroffen wurde -> |Kuhmax| */
   ay0=dm(Start);
   ar=pass ay0;                /* generate alu status */
   if eq call StartFinder;     /* falls CoreSignal fehlt -> gewinne Start/StopZeit aus Segmentsignal */
   ay0=dm(Start);
   ax1=dm(Ende);               /* 8.8 */
   ar=ax1-ay0;                 /* laenge=ende-anfang 8.8 */
   sr=lshift ar by -8 (lo);    /* 8.8 -> Differenz im 8.0 (16.0) FORMAT, OHNE runden ! */
   ax1=sr0;                    /* copy fuer division */
   i0=dm(ATstart);             /* Startadresse einladen */
   m3=dm(RawStart);
   modify(i0,m3);
   mr0=dm(i0,m1);              /* ADC(Anfang) einladen, Zeiger auf ADC(n+1) */
   ar=dm(Energy);              /* Steigung ausrechnen: PeakHoehe(=Energie/8)/ende-anfang */
   sr=lshift ar by 2 (lo);     /* unsigned Integerdivision: 27.5/16.0=12.4 */
   ay0=sr0;
   af=pass sr1;
   astat=0;                         /* unsigned division */
   divq ax1; divq ax1;
   divq ax1; divq ax1;
   divq ax1; divq ax1;
   divq ax1; divq ax1;
   divq ax1; divq ax1;
   divq ax1; divq ax1;
   divq ax1; divq ax1;               /* 12 bit berechnet -> keine Nachkommastellen */
   ax0=4095;                         /* FORMAT korrektur */
   ar=ax0 AND ay0;                   /* FORMAT 16.0 */
   i1=^SegBuffer1;
   my0=1;                            /* 1 wegen reinem addieren */
   mr1=0;                            /* mr0 = startwert (s.o.), mr1 wir gel"oscht */
   mx0=ar;                           /* init der Steigung: Ergebnis der Division */
   ay1=0;                            /* init des Vergeichswerts */
   cntr=dm(SegAnalen);
   mr=mr+mx0*my0 (uu), ay0=dm(i0,m1); /* berechne ADC, lade ADC(n+1) */
   do cmplooop1 until ce;                       /* Vergleiche ADC(n+m) mit ADC(n) + m*Steigung */
            ar=mr0-ay0;                         /* bilde Differenz: ADC(n)-Gerade */
            if lt ar=abs ar;                    /* absolutwert */
            af=ar-ay1, dm(i1,m1)=ar;            /* vergleiche Pulshoehe mit vorherigen Werten , speichere Wert im Segbuffer*/
            if gt call newmax;
cmplooop1:  mr=mr+mx0*my0 (uu), ay0=dm(i0,m1);  /* mr=mr+steigung*1= ADC(0) + n* Steigung*1 , lade neuen ADC(n) */
   dm(UretVal)=ay1;                     /* ay1, Koeln Format 16.0, keine Nachkommastellen */
   dm(MaxPeakVal1)=ay1;

   I0 = DM (saveI0);
```

```
   I1 = DM (saveI1);
   I3 = DM (saveI3);
   I4 = DM (saveI4);
JUMP UserChannelReturn;


/*-------------------------------------------------------------------------------------*/

Winkelanalyse2:
   ax0=dm(SegAna0n);
   ar=pass ax0;
   if eq jump UserChannelReturn;    /* falls doch keine Segmentanalyse gewuenscht -> ende */
   ax0=dm(GundKoeln);
   ar=pass ax0;
   if ne jump Koelnanalyse2;         /* 1->benutze den Koeln-Methode, 0->benutze CG-Methode */
JUMP Gundanalyse2;


/*  CG/Koeln(OhneTreffer): Segment -> finde |KuhMax| */
Gundanalyse2:
   ay1=0;                   /* initilisiere alten Wert */
   i0=dm(ATstart);
   i1=^SegBuffer2;
   m3=dm(BaseLen);
   modify(i0,m3);           /* Startadresse nach i0 */
   ay0=dm(Baseline12);      /* lade Baseline ADC-Wert nach ay0 -> FORMAT 12.0 */
   ax0=dm(i0,m1);           /* lade 1.ADC-Wert nach sr0 */
   cntr=dm(AnaLen);         /* Counter=Analyse-Laenge */
   ar=ax0-ay0, ax0=dm(i0,m1);
   do cmploop2 until ce;
          if lt ar=abs ar;              /* bilde Absolutwert */
          af=ar-ay1, dm(i1,m1)=ar;      /* ar=neuer Wert - alter Wert(alles im 12.0 Format), erstelle buffer*/
          if ge call newmax;            /* ar>0, d.h. neuer Wert>alterWert (absolut)*/
cmploop2:  ar=ax0-ay0, ax0=dm(i0,m1);    /* ar=ADC(n)-Baseline , lade ADC(n+1) in ax0 */
   dm(MaxPeakVal2)=ay1;     /* Maximale Ladungspulshoehe sichern 16.0 !!!*/
   dm(UretVal)=ay1;         /* Rueckgabe: Ladungspulshoehe FORMAT: 16.0 !!!*/


   I0 = DM (saveI0);
   I1 = DM (saveI1);
   I3 = DM (saveI3);
   I4 = DM (saveI4);

JUMP UserChannelReturn;


/* Segment MIT Treffer:  |KuhMax| ueber Gerade */
Koelnanalyse2:
   ax0=dm(Energy);             /* FORMAT: 13.3, benutze als PeakHoehe */
   ay0=dm(EnCut0ff);
   ar=ax0-ay0;
   if le jump Gundanalyse2;    /* falls Segment nicht getroffen wurde -> |Kuhmax| */
   ay0=dm(Start);
   ar=pass ay0;                /* generate alu status */
   if eq call StartFinder;     /* falls CoreSignal fehlt -> gewinne Start/StopZeit aus Segmentsignal */
   ay0=dm(Start);
   ax1=dm(Ende);               /* 8.8 */
   ar=ax1-ay0;                 /* laenge=ende-anfang 8.8 */
   sr=lshift ar by -8 (lo);    /* 8.8 -> Differenz im 8.0 (16.0) FORMAT, OHNE runden ! */
   ax1=sr0;                    /* copy fuer division */
   i0=dm(ATstart);             /* Startadresse einladen */
   m3=dm(RawStart);
   modify(i0,m3);
   mr0=dm(i0,m1);              /* ADC(Anfang) einladen, Zeiger auf ADC(n+1) */
   ar=dm(Energy);              /* Steigung ausrechnen: PeakHoehe(=Energie/8)/ende-anfang */
   sr=lshift ar by 2 (lo);     /* unsigned Integerdivision: 27.5/16.0=12.4 */
   ay0=sr0;
   af=pass sr1;
   astat=0;                         /* unsigned division */
   divq ax1; divq ax1;
   divq ax1; divq ax1;
   divq ax1; divq ax1;
   divq ax1; divq ax1;
   divq ax1; divq ax1;
   divq ax1; divq ax1;
   divq ax1; divq ax1;               /* 12 bit berechnet -> keine Nachkommastellen */
   ax0=4095;                         /* FORMAT korrektur */
   ar=ax0 AND ay0;                   /* FORMAT 16.0 */
   i1=^SegBuffer2;
   my0=1;                            /* 1 wegen reinem addieren */
   mr1=0;                            /* mr0 = startwert (s.o.), mr1 wir gel"oscht */
   mx0=ar;                           /* init der Steigung: Ergebnis der Division */
   ay1=0;                            /* init des Vergeichswerts */
   cntr=dm(SegAnalen);
   mr=mr+mx0*my0 (uu), ay0=dm(i0,m1); /* berechne ADC, lade ADC(n+1) */
   do cmplooop2 until ce;                     /* Vergleiche ADC(n+m) mit ADC(n) + m*Steigung */
            ar=mr0-ay0;                       /* bilde Differenz: ADC(n)-Gerade */
            if lt ar=abs ar;                  /* absolutwert */
            af=ar-ay1, dm(i1,m1)=ar;          /* vergleiche Pulshoehe mit vorherigen Werten , speichere Wert im Segbuffer*/
            if gt call newmax;
cmplooop2:  mr=mr+mx0*my0 (uu), ay0=dm(i0,m1);  /* mr=mr+steigung*1= ADC(0) + n* Steigung*1 , lade neuen ADC(n) */
   dm(UretVal)=ay1;                  /* ay1, Koeln Format 16.0, keine Nachkommastellen */
   dm(MaxPeakVal2)=ay1;
```

```
   I0 = DM (saveI0);
   I1 = DM (saveI1);
   I3 = DM (saveI3);
   I4 = DM (saveI4);
JUMP UserChannelReturn;


/*-------------------------------------------------------------------------------------*/


Winkelanalyse3:
   ax0=dm(SegAna0n);
   ar=pass ax0;
   if eq jump UserChannelReturn;    /* falls doch keine Segmentanalyse gewuenscht -> ende */
   ax0=dm(GundKoeln);
   ar=pass ax0;
   if ne jump Koelnanalyse3;         /* 1->benutze den Koeln-Methode, 0->benutze CG-Methode */
JUMP Gundanalyse3;


/*  CG/Koeln(OhneTreffer): Segment -> finde |KuhMax| */
Gundanalyse3:
   ay1=0;                   /* initilisiere alten Wert */
   i0=dm(ATstart);
   i1=^SegBuffer3;
   m3=dm(BaseLen);
   modify(i0,m3);           /* Startadresse nach i0 */
   ay0=dm(Baseline12);      /* lade Baseline ADC-Wert nach ay0 -> FORMAT 12.0 */
   ax0=dm(i0,m1);           /* lade 1.ADC-Wert nach sr0 */
   cntr=dm(AnaLen);         /* Counter=Analyse-Laenge */
   ar=ax0-ay0, ax0=dm(i0,m1);
   do cmploop3 until ce;
           if lt ar=abs ar;              /* bilde Absolutwert */
           af=ar-ay1, dm(i1,m1)=ar;      /* ar=neuer Wert - alter Wert(alles im 12.0 Format), erstelle buffer*/
           if ge call newmax;            /* ar>0, d.h. neuer Wert>alterWert (absolut)*/
cmploop3:  ar=ax0-ay0, ax0=dm(i0,m1);    /* ar=ADC(n)-Baseline , lade ADC(n+1) in ax0 */
   dm(MaxPeakVal3)=ay1;     /* Maximale Ladungspulshoehe sichern 16.0 !!!*/
   dm(UretVal)=ay1;         /* Rueckgabe: Ladungspulshoehe FORMAT: 16.0 !!!*/

   I0 = DM (saveI0);
   I1 = DM (saveI1);
   I3 = DM (saveI3);
   I4 = DM (saveI4);
JUMP UserChannelReturn;


/* Segment MIT Treffer:  |KuhMax| ueber Gerade */
Koelnanalyse3:
   ax0=dm(Energy);             /* FORMAT: 13.3, benutze als PeakHoehe */
   ay0=dm(EnCut0ff);
   ar=ax0-ay0;
   if le jump Gundanalyse3;    /* falls Segment nicht getroffen wurde -> |Kuhmax| */
   ay0=dm(Start);
   ar=pass ay0;                /* generate alu status */
   if eq call StartFinder;     /* falls CoreSignal fehlt -> gewinne Start/StopZeit aus Segmentsignal */
   ay0=dm(Start);
   ax1=dm(Ende);               /* 8.8 */
   ar=ax1-ay0;                 /* laenge=ende-anfang 8.8 */
   sr=lshift ar by -8 (lo);    /* 8.8 -> Differenz im 8.0 (16.0) FORMAT, OHNE runden ! */
   ax1=sr0;                    /* copy fuer division */
   i0=dm(ATstart);             /* Startadresse einladen */
   m3=dm(RawStart);
   modify(i0,m3);
   mr0=dm(i0,m1);              /* ADC(Anfang) einladen, Zeiger auf ADC(n+1) */
   ar=dm(Energy);              /* Steigung ausrechnen: PeakHoehe(=Energie/8)/ende-anfang */
   sr=lshift ar by 2 (lo);     /* unsigned Integerdivision: 27.5/16.0=12.4 */
   ay0=sr0;
   af=pass sr1;
   astat=0;                         /* unsigned division */
   divq ax1; divq ax1;
   divq ax1; divq ax1;
   divq ax1; divq ax1;
   divq ax1; divq ax1;
   divq ax1; divq ax1;
   divq ax1; divq ax1;               /* 12 bit berechnet -> keine Nachkommastellen */
   ax0=4095;                         /* FORMAT korrektur */
   ar=ax0 AND ay0;                   /* FORMAT 16.0 */
   i1=^SegBuffer3;
   my0=1;                            /* 1 wegen reinem addieren */
   mr1=0;                            /* mr0 = startwert (s.o.), mr1 wir gel"oscht */
   mx0=ar;                           /* init der Steigung: Ergebnis der Division */
   ay1=0;                            /* init des Vergeichswerts */
   cntr=dm(SegAnalen);
   mr=mr+mx0*my0 (uu), ay0=dm(i0,m1); /* berechne ADC, lade ADC(n+1) */
   do cmplooop3 until ce;                      /* Vergleiche ADC(n+m) mit ADC(n) + m*Steigung */
           ar=mr0-ay0;                         /* bilde Differenz: ADC(n)-Gerade */
           if lt ar=abs ar;                    /* absolutwert */
           af=ar-ay1, dm(i1,m1)=ar;            /* vergleiche Pulshoehe mit vorherigen Werten , speichere Wert im Segbuffer*/
           if gt call newmax;
cmplooop3:  mr=mr+mx0*my0 (uu), ay0=dm(i0,m1);  /* mr=mr+steigung*1= ADC(0) + n* Steigung*1 , lade neuen ADC(n) */
```

```
   dm(UretVal)=ay1;                    /* ay1, Koeln Format 16.0, keine Nachkommastellen */
   dm(MaxPeakVal3)=ay1;

   I0 = DM (saveI0);
   I1 = DM (saveI1);
   I3 = DM (saveI3);
   I4 = DM (saveI4);

JUMP UserChannelReturn;


/*-------------------------------------------------------------------------------------------------------------------------------------------*/



newmax:
   ay1=ar;                     /* sichere neuen PeakWert (12.4) */
rts;



/*--------------------------------------------------------------------
+ UserEvent is called after the regular event processing has finished,
+ and only if for at least one channel bit 0 of ChanCSRB was set.
--------------------------------------------------------------------*/
UserEvent:
   ax0=dm(GoodBadSwitch);
   ar=pass ax0;
   if ne jump badevents;
   ar=dm(GoodEventNum);
   ar=ar+1;
   dm(GoodEventNum)=ar;
JUMP UserEventReturn;

badevents:
   ar=dm(BadEventNum);
   ar=ar+1;
   dm(BadEventNum)=ar;
JUMP UserEventReturn;




/*--------------------------------------------------------------------
+ UserRunFinish is called after the run has ended, but before the host
+ is notified,but only if for at least one channel bit 0 of ChanCSRB
+ was set.
--------------------------------------------------------------------*/
UserRunFinish:
   m3=1;
   i0=^UserOut;
   ax0=dm(GundKoeln);          /* gebe analyse typ zurueck */
   dm(i0,m3)=ax0;
   ax0=dm(Start);              /* DEBUG-Info */
   dm(i0,m3)=ax0;
   ax0=dm(Ende);
   dm(i0,m3)=ax0;
   ax0=dm(MaxSteepPos0);
   dm(i0,m3)=ax0;
   ax0=dm(RawSlopePos);
   dm(i0,m3)=ax0;
   ax0=^SteSloBuffer;
   dm(i0,m3)=ax0;
   ax0=%SteSloBuffer;
   dm(i0,m3)=ax0;
   ax0=^SegBuffer1;
   dm(i0,m3)=ax0;
   ax0=%SegBuffer1;
   dm(i0,m3)=ax0;
   ax0=^SegBuffer2;
   dm(i0,m3)=ax0;
   ax0=%SegBuffer2;
   dm(i0,m3)=ax0;
   ax0=^SegBuffer3;
   dm(i0,m3)=ax0;
   ax0=%SegBuffer3;
   dm(i0,m3)=ax0;
   ax0=dm(GoodBadSwitch);
   dm(i0,m3)=ax0;
   ax0=dm(GoodEventNum);
   dm(i0,m3)=ax0;
   ax0=dm(BadEventNum);
   dm(i0,m3)=ax0;
JUMP UserRunFinishReturn;
.EndMod;
```

Anhang D

Der Steepest Slope Algorithmus in Hardware

'Milk is for babies. When you grow up you have to drink beer.'
Arnold Schwarzenegger

Der Ansatz der Firma XIA wurde im Hauptteil erklärt: möglichst einfache Algorithmen im FPGA, der direkt auf dem ADC Datenstrom operiert, implementieren und komplexe Berechnung durch den DSP ausführen lassen. Dieser Ansatz kann auch für die MINIBALL Algorithmen von Nutzen sein. Der Steepest Slope Algorithmus benötigt die zweite Ableitung. Dies läßt sich als ein drei Tap langer FIR Filter mit den Koeffzienten 1 und -2 darstellen. Aus diesen Daten muss dann nur noch die Position des Steepest Slope in den ADC Daten bestimmt werden. Diese ist dann auf 25 ns genau bekannt. Der DSP kann diese Position auslesen und damit direkt die interpolierte Position berechnen. Der DSP muss derzeit die Position des Steepest Slope nachträglich bestimmen und hierfür mehrmals längere Schleifen ausführen. Der Rechenaufwand für die eigentliche Interpolation ist dagegen vernachlässigbar. Die Hardware-Implementierung würde die Steepest Slope Position "nebenbei" liefern, da die Operationen parallel ausgeführt werden.

Deshalb wurde eine erste Implementierung des Steepst Slope Algorithmus umgesetzt und getestet. Das Design ist in Abbildung D.1 gezeigt und wurde mittels der Hardwarebeschreibungssprache VHDL umgesetzt. Zur Simulation konnte die am Institut für Hardwareinformatik der Universität Heidelberg vorhandene Software der Firma Synopsis [46] benutzt werden. Zum Testen wurden die ORCA FPGAs der Firma Lucent [41] verwendet. Diese befinden sich auf einem vom Institut für Hardwareinformatik mitentwickeltem Testboard [54]. Zur Synthese des Designs standen zwei Programme zur Auswahl: der FPGA Compiler II der Firma Synopsis und die Tools der Firma Lucent. Obwohl der FPGA Compiler II bessere Syntheseergebnisse liefern soll, wurde, mangels Erfahrung mit dieser Software, den Lucent Tools Vorzug gegeben (Es war nicht bekannt, wie Ausgänge eines Designs mit den Pins des FPGAs verdrahtet werden können. Da die Pins fest mit den Steckern des Testboard verschaltet sind, war dies allerdings notwendig. Dieser Vorgang ist für die Lucent Tools bekannt.).

Nachdem die logische Funktion des Design mittels Simulation erfolgreich getestet wurde, war die Synthese des Designs an der Reihe. Hierbei stellte sich schnell heraus, dass schon alleine die Addition zweier 12 Bit Zahlen bei 40 MHz Taktfrequenz für die ORCA

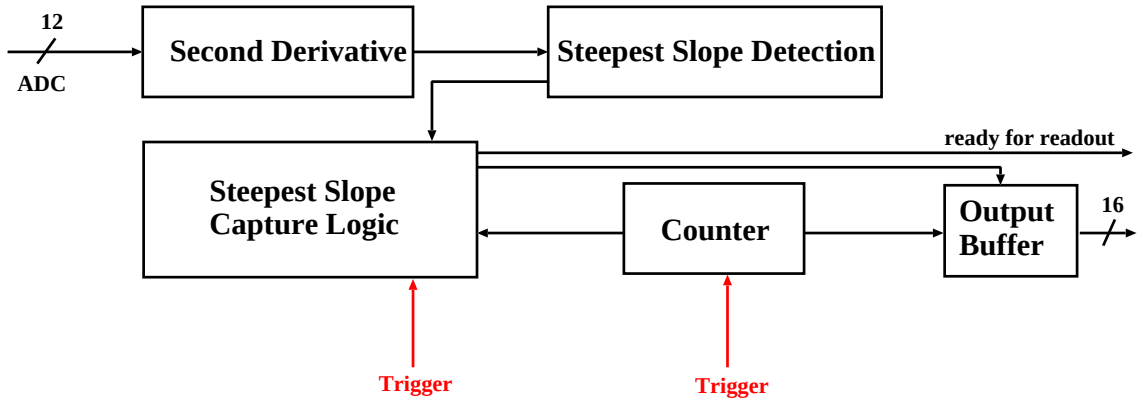

Abbildung D.1: Struktur des, mittels VHDL implementierten, Steepest Slope Algorithmus. Das Design benötigt hauptsächlich zwei Eingaben: die ADC Werte und ein schnelles Triggersignal. Als Ausgabe liefert es die Zeit (in 25 ns) zwischen Trigger und Steepest Slope. Der Einfluß durch das Pipelining muss dabei unbedingt beachtet werden. Ein zusätzliches Signal zeigt das Ende der Analyse an. Diese Ende wird durch eine einstellbare Analyselänge gesteuert.

Abbildung D.2: Schema des Testaufbaus. Die Testdaten wurden in einen Pattern Generator einprogrammiert.Dieser ist in der Lage, Signale mit einer maximalen Rate von 100 MHz zu erzeugen. Diese wurden direkt per Kabel auf das Orca Board gegeben. Ebenfalls per Kabel wurden die Ausgänge des FPGA vom Board abgegriffen und mit einem Logic Analyzer begutachtet.

FPGAs eine große Herausforderung darstellt. Der kritische Pfad (Pfad mit der größten Verzögerung) konnte nicht soweit erniedrigt werden, dass das Design mit den angepeilten 40 MHz laufen würde. Deshalb mussten zusätzliche Pipeline Register eingebaut werden. Der kritische Pfad dieses Designs ergab, dass zumindest mit einem XILINX XCS30XL FPGA die geforderten 40 Mhz zu verarbeiten sind. Nachdem das Design in die ORCA FPGAs geladen war, konnte man dieses mit Hilfe eines Pattern Generators mit Eingaben füttern. Als Testdaten wurden Coresignale des MINIBALL Detektors verwendet, die mit der XIA Karte aufgenommen wurden. Der benutzte Testaufbau ist in Abbildung D.2 gezeigt. Die Antwort des Design auf die Eingabedaten, wurde vom Testboard abgegriffen und mit einem Logic Analyzer dargestellt. Leider konnte es nicht mit Frequenzen, die höher als 10 MHz waren, betrieben werden. Dies war nicht erwartet worden, da nach den Angaben der Synthesesoftware Frequenzen bis 30 MHz möglich sein sollten. Für höhere Frequenzen 10 MHz kam es zu Spikes auf den *Eingangssignalen*. Da davon auch das Resetsignal betroffen war, wurde das Design ständig zurückgesetzt. Es ist möglich, dass das programmierte Design dafür verantwortlich ist, allerdings erscheint es warscheinlicher, dass die Impedanz des Pattern Generators nicht an das Testboard angepasst war. Der Pattern Generator stellt zwei mögliche Abschlußwiderstände zur Auswahl. Aus Zeitmangel konnte diese Erklärung leider nicht verifiziert werden, da es außerdem denkbar ist, dass beide Impedanzen nicht mit dem Testboard zu vereinbaren sind.

Nachdem der Test schon beschrieben wurde, soll nun noch der Aufbau des Designs beschrieben werden. Das getestete Design ist schematisch in Abbildung D.1 gezeigt. Aus den ankommenden Daten des ADCs wird die zweite Ableitung berechnet und diese an die `Steepest Slope Detection` weitergegeben. Diese Einheit benachrichtigt, die `Steepest Slope Capture Logic`, falls sie ein neues Minimum entdeckt. Die `Capture Logic` (ein endlicher Automat) wird durch ein Triggersignal eingeschaltet. Das Triggersignal setzt außerdem den Wert des `Counters` gleich Null, der sich mit jedem Takt inkrementiert. Benachrichtigt nun die `Detection` die `Capture Logic` im eingeschalteten Zustand von der Ereignung eines neuen Minimums, so erlaubt die `Capture Logic` die Übernahme des `Counter` Wertes in das Ausgaberegister. Damit der gesamte Vorgang nach Ende des Pulsanstiegs beendet wird, muss eine Analyselänge an das Design übergeben werden. Überschreitet der Wert des `Counters` den Wert der Analyselänge, so stoppt dieser mit einem Signal die `Capture Logic`. Gleichzeitig wird ein Signal erzeugt, welches das Ende der Datenaufnahme anzeigt. Ist dieses Signal gesetzt, kann der Wert des Zähler zum Zeitpunkt des letzten Minimums aus dem `Output Buffer` gelesen werden. Diese gibt also die zeitliche Differenz zwischen Triggerereignis und Steepest Slope Ereignis an. Da der Trigger auch das Timing der FIFOs steuert, sollte sich mit dieser Information die Position des Steepest Slope in den Tracedaten im DSP Bufferbestimmen lassen.

Die Bestimmung der induzierte Ladung von nicht getroffenen Segmenten läßt sich mit ähnlich wenig Aufwand verwirklichen. Für die getroffenen Segmente muss ein entsprechender Algorithmus allerdings erst noch entwickelt werden.

Literaturverzeichnis

[1] D. Habs et. al.
The REX-ISOLDE Project,
Nucl. Instr. Meth. A616 (1997) 29c-38c

[2] D.Habs et. al.
Physics with Ge-Miniball-Arrays,
Prog. Part. Nucl. Phys. Vol.38, 1997, p.111-126

[3] Heinz-Georg Thomas
Entwicklung eines Germanium-CLUSTER-Detektors für das Gamma-Spektrometer EUROBALL,
Dissertation am Institut für Kernphysik, Köln

[4] Glenn F. Knoll
Radiation Detection and Measurement,
John Wiley & Sons, Inc., 1989

[5] Hans Frauenfelder und Ernest M. Henley
Teilchen und Kerne: Die Welt der subatomaren Physik,
R. Oldenburg Verlag, 1999

[6] W.R. Leo
Techniques for Nuclear and Particle Physics Experiments,
Springer-Verlag, 1994

[7] F.S. Goulding and D.A. Landis
Ballistic Deficit Correction in Semiconductor Detector Spectrometers,
IEEE Trans. Nucl. Sci. NS-35 (1988)

[8] Neil W. Ashcroft and N. David Mermin
Solid State Physics,
Saunders College Publishing, 1976

[9] F.S Goulding and D.A.Landis
Signal Processing for Semiconductor Detectors,
IEEE Trans. Nucl. Sci. NS-29 (1982), p.1125-1141

[10] Christoph Gund
Eigenschaften des zweifach segmentierten Prototypen eines MINIBALL Cluster

Moduls,
Diplomarbeit am Max-Planck-Institut für Kernphysik, Heidelberg, 1996

[11] Christoph Gund
The Sixfold Segmented MINIBALL Module Simulation and Experiment,
Dissertation am Max-Planck-Institut für Kernphysik, Heidelberg, 2000

[12] Dirk Weisshaar
Dissertation am Institut für Kernphysik, Köln, In Arbeit

[13] G. Cavalleri, G.Fabri, E. Gatti and V. Svelto
On The Induced Charge in Semiconductors Detectors,
Nucl. Instr. Meth. 21 (1963), p.177-178

[14] P.A. Tove, K. Falk
Transit Time of Charge Carriers in the Semiconductor Ionization Chamber,
Nucl. Instr. Meth. 12 (1961), p.278-290

[15] V.Radeka
Low-Noise Techniques in Detectors,
Ann. Rev. of Nucl. Part. Sci. 29, 1, 1982, p.764-768

[16] Stephan Schwebel
Pulsform-Analyse von High Purity Germanium-Detektorsignalen mit Flash ADC's,
Diplomarbeit am Max-Planck-Institut für Kernphysik, Heidelberg, 1992

[17] Luis Palafox Gamir
A New Method for the Determination of the Entry Position of γ-rays in HPGe Detectors by Current Analysis,
Dissertation am Max-Planck-Institut für Kernphysik, Heidelberg, 1997

[18] X-ray Instrumentation Associates
DGF-4C: 4 channel, High Speed Digital γ-ray Spectrometer,
Diverse Manuals, Technical Notes und Papers,
http://www.xia.com/downloads.html

[19] P. Deuflhard, Andreas Hohmann
Numerische Mathematik Bd.1, Eine algorithmisch orientierte Einführung,
de Gruiter

[20] Z.H. Cho and R.L. Chase
Comparative Study of the Timing Techniques Currently Employed with Ge Detectors
Nucl. Instr. Meth. 98 (1972), p.335-347

[21] Richard G. Lyons
Understanding Digital Signal Processing,
Addison-Wesley Longman, Inc., Reading, MA, 1997

[22] Valentin T. Jordanov, Glenn F. Knoll
Digital synthesis of puse shapes in real time for high resolution radiation spectroscopy,
Nucl. Instr. Meth. A345 (1994), p.337-345

[23] E. Fairstein
Linear Unipolar Pulse-Shaping Networks: Current Technology,
IEEE Trans. Nucl. Sci. NS-37 (1990), p.382-397

[24] S.M. Hinshaw and D.A. Landis
A Practical Approach to Ballistic Deficit Correction,
IEEE Trans. Nucl. Sci. NS-37 (1990), p.374-377

[25] Billy W. Loo and Fred S. Goulding
Ballistic Deficits in Pulse Shaping Amplifiers,
IEEE Trans. Nucl. Sci. NS-35 (1988), p.114-117

[26] F.S. Goulding, D.A. Landis and S.M. Hinshaw
Large Coaxial Germanium Detectors - Correction for Ballistic Deficit and Trapping Losses,
IEEE Trans. Nucl. Sci. NS-37 (1990), p.417-423

[27] F.S.Gouldig
Pulse-Shaping in Low-Noise Nuclear Amplifiers: A Physical Approach to Noise Analysis,
Nucl. Instr. Meth. 100 (1972), p.493-504

[28] DSP Kernels
http://developer.intel.com/software/idap/os/dsp/,
Intel Corporation

[29] Streaming SIMD Extensions
http://developer.intel.com/software/idap/processor/ia32/pentiumiii/sse.htm,
Intel Corporation

[30] Inside 3DNow! Technology
http://www.amd.com/italy/cpg/k623d/inside3d.html,
Advanced Mikro Devices, Inc.

[31] AltiVec Technology
http://www.mot.com/SPS/PowerPC/AltiVec/, Motorola Inc.

[32] Steven W. Smith
The Scientist and Engineer's Guide to Digital Signal Processing,
California Technical Publishing,
http://www.dspguide.com

[33] *http://www.analog.com*

[34] *http://www.xilinx.com/products/logicore/dsp/index.htm*

[35] *http://www.quicklogic.com/devices/dsp/default.htm*

[36] *http://products.analog.com/products/info.asp?product=ADSP-2192*

[37] Stephen Brown, Jonathan Rose
FPGA and CPLD Architektures: A Tutorial,
http://www.eecg.toronto.edu/ jayar/pubs/brown/survey.ps.gz

[38] Xputer Projekt
http://x_puters.informatik.uni-kl.de/index_d.html

[39] John V. Oldfield, Richard C. Dorf
Field Programmable Gate Arrays: Rekonfigurable Logic for Rapid Prototyping and Implementation of Digital System,
John Wiley and Sons, Inc., 1995

[40] *http://www.xilinx.com*

[41] Lucent Technologies ORCA Series 3 FPGAs
http://www.lucent.com/micro/netcom/orca/index.html

[42] David A. Patterson, John L. Hennessy
Computer Organisation and Design: The Hardware/Software Interface
Morgan Kaufmann Publishers, Inc., 1998

[43] Volker Lindenstruth und Falk Lesser
Seminarunterlagen: Logischer Entwurf Digitaler Systeme,
http://www.ihep.uni-heidelberg.de/Hardwinf/Lehre/gradkurs1099/page2.html

[44] G. Lehmann, B. Wunder und M. Selz
Schaltungsdesign mit VHDL: Synthese, Simulation und Dokumentation digitaler Schaltungen,
http://www-itiv.etec.uni-karlsruhe.de/FORSCHUNG/VEROEFFENTLICHUNGEN/lws94/lws94.html

[45] A. Mäder
VHDL Kurzbeschreibung,
Universität Hamburg
http://tech-www.informatik.uni-hamburg.de/vhdl/doc/kurzanleitung/vhdl.ps.gz

[46] Synopsys, Inc.*http://www.synopsys.com/*

[47] Joao M.R. Cardoso, J.Basilio Simoes, Carlos M.B.A. Correia
A mixed analog-digital pulse spectrometer,
Nucl. Instr. Meth. A 422 (1999) p. 400-404

[48] Pulse Processing ADC (PPADC)
Institut für Kernphysik, Forschungszentrum Jülich GmbH, Forschungszentrum Rossendorf und target systemelectronic GmbH, Solingen.

[49] V. Radeka
Trapezoidal Filtering of Signals from Large Germanium Detectors at High Rates,
IEEE Trans. Nuc. Sci. NS-19, No. 1, 412 (1972)

[50] Benutzerhandbuch VC32-CC32,
ARW-Elektronik, Wiesloch und W-IE-NE-R, Plein & Baus GmbH, Burscheid

[51] *http://www.yale.edu/fastcamac/*

[52] Frank Köck
Datenerfassung für kernphysikalische Experimente in einer heterogenen Rechnerumgebung,
Dissertation am Max-Planck-Institut für Kernphysik, Heidelberg, 1994

[53] FIREWIRE (IEEE1394)
http://www.howstuffworks.com/firewire.htm
kurze Beschreibung plus viele Links

[54] OR3TP12 PCI Evaluation Kit
http://www.morethanip.com/products/or3eval/index.html