

Ein digitales Test-System für das Miniball-Experiment

Fabian Träger, Matrikel-Nr.: 3602670

21.08.2011

Zusammenfassung

Das Miniball-Experiment, mit dem zahlreiche kernphysikalische Experimente mit radioaktiven Sekundärstrahlen durchgeführt werden, ist Hauptnutzer des REX-ISOLDE-Beschleunigers am CERN – der Europäischen Organisation für Kernforschung. Aufgrund der Komplexität der Experimente und der Besonderheiten des verwendeten Datenaufnahmesystems besitzt das verwendete Trigger-System eine relativ spezielle und umfängliche Struktur. Diese soll für die Zukunft in ein einziges programmierbares Digitalmodul integriert werden. Ziel der vorliegenden Bachelorarbeit war es, ein digitales Test-System für das Trigger-System des Miniball-Experiments zu entwickeln. Damit sollen einerseits die Signale, die das Trigger-System verarbeiten muss, simuliert und andererseits die Daten und Steuersignale, die das Trigger-System wieder ausgibt, analysiert werden, um dessen Funktionalität zu testen und Fehler identifizieren zu können. Zu diesem Zweck wurde ein neu entwickeltes VME-Modul in der Programmiersprache „C“ programmiert, konfiguriert und so die erforderliche Logik und Funktionalität in einen einzigen FPGA (Field Programmable Gate Array) integriert. Das System wurde im Rahmen dieser Arbeit fast fertig gestellt und erprobt und kann schon bald am CERN eingesetzt werden.

Inhaltsverzeichnis

1.	Einleitung	3
2.	Das Experiment	4
2.1	Der REX-ISOLDEBeschleuniger.....	4
2.2	Die Zeitstruktur von REX-ISOLDE	6
2.3	Der Miniball-Aufbau	6
2.4	Die auftretenden Signale und deren Eigenschaften.....	8
3.	Die Verwendeten Hilfsmittel	10
3.1	Das Modul.....	10
3.2	Die Firmware	10
4.	Die Konfiguration des Moduls	12
4.1	Übertragung der Firmware.....	12
4.2	Programmierung des Konfigurationsprogramms.....	13
5.	Simulation der Signale	21
6.	Analyse der Signale	24
7.	Die Konfiguration des Test-Systems	27
8.	Fazit und Ausblick	29
9.	Anhang	30

1. Einleitung

Aufgabe dieser Arbeit war es, ein digitales Test-System für das Trigger-System des Miniball-Experiments am REX-ISOLDE-Beschleuniger beim CERN¹ zu entwickeln. Der Aufbau des Trigger-Systems für das Miniball-Experiment ist verhältnismäßig komplex gestaltet. In diesem Experiment werden aus einer Ionenstrahlquelle radioaktive, stark geladene Ionen auf ein Target geschossen, wo die durch den Stoß mit dem Target ausgelösten physikalischen Reaktionen mittels des Miniball-Detektor-Aufbaus gemessen werden. Dieser besteht aus mehreren Germanium-Detektoren zur Gamma-Detektion, die jeweils in Segmente unterteilt sind, sowie einem Silizium-Teilchen-Detektor, der stark segmentiert ist. Zudem gibt es mehrere Monitor-Signale von der Ionenstrahlquelle, aus denen Informationen über den Strahl gelesen und auch die Messung und die Datenaufnahme gesteuert werden. Aufgrund dieser großen Anzahl an Kanälen und der komplexen Steuerung des Experiments ist eine spezielle Trigger-Elektronik nötig, die die Signale verarbeitet. Diese ist nicht einfach zu pflegen, da schon ein falsch verbundenes Kabel oder ein einzelnes defektes Modul in einer Fehlfunktion des Systems resultieren kann, die nicht sofort entdeckt wird. Des Weiteren ist zu erwähnen, dass dieser Aufbau seit 10 Jahren im Betrieb ist, somit auch veraltete Module beinhaltet und im Laufe der Zeit immer mehr, teils umständliche Modifikationen daran gemacht wurden, was den Überblick und auch die Wartung des Systems sehr erschwert. Um diese Probleme zu umgehen, ist der Wechsel von diesem System aus vielen Modulen zu einem digitalen Trigger-System in einem einzigen Modul geplant. Ein solches bietet eine bessere Benutzerfreundlichkeit, da kein spezielles Fachwissen über einzelne Module von Nöten ist und es leichter zu überschauen ist. Zudem ist sowohl durch die Implementierung in einem einzigen anstatt in vielen Modulen, als auch die stark verminderte Anzahl an Kabeln die Fehleranfälligkeit sehr gering und eine gute Zuverlässigkeit gewährleistet. Modifikationen sind ebenfalls einfacher, da dazu nur die Konfiguration des Moduls verändert werden muss. Ein solches digitales Trigger-System wurde in der Arbeit von Richard Salentin² in einem VULOM3³-Modul umgesetzt.

Für den Test und die Analyse der komplexen Funktionalität dieses Trigger-Systems sollte ein digitales Test-System, ebenfalls in einem VULOM3, aufgebaut werden. In diesem sollten die Signale, die vom Experiment an das Trigger-System gesendet werden, wie Teilchen- und Gammatrigger simuliert und dazu die Daten, die von diesem ausgegeben werden, durch Zähler und Logik-Funktionen auf Fehler überprüft werden. Dadurch lässt sich das Trigger-System, schon bevor es im Experiment eingesetzt wird, einfach testen, ohne dass wertvolle Strahlzeit benötigt wird. Aber auch am Experiment könnte das Test-System nach einer Modifikation zur weiteren Überwachung genutzt werden, indem es Zusatzinformationen generiert und den Nutzer bei Defekten warnt. Diese Arbeit beschränkt sich auf die Implementierung des Tests von Coulex-Experimenten, wobei es möglich ist, das System später auch für Transfer-Experimente zu konfigurieren.

¹ Die Europäische Organisation für Kernforschung, „www.cern.ch“

² Richard Salentin, „Ein digitales Triggersystem für das Miniball-Experiment“

³ VME Universal Logic Module, GSI (Gesellschaft für Schwerionenforschung), „http://www.gsi.de/informationen/wti/ee/elekt_entwicklung/vulom3.html“

2. Das Experiment

Um die Signale möglichst realistisch simulieren zu können, sollte zuallererst das Experiment in dessen Aufbau und Durchführung betrachtet werden, die auftretenden Signale erfasst und deren Eigenschaften, hinsichtlich Häufigkeit, bedingtes Auftreten und Länge, analysiert werden. In diesem Experiment werden exotische Kerne untersucht, die durch einen radioaktiven Strahl (Radioactive Ion Beam, kurz RIB) aus einem Beschleuniger erzeugt werden.

2.1 Der REX-ISOLDE Beschleuniger

Der gekoppelte Massenseparator ISOLDE⁴ (Isotope Mass Separator On-Line) ist eine Einrichtung, die zur Produktion einer großen Vielfalt von radioaktiven Ionenstrahlen für viele verschiedene Experimente und Anwendungen in zahlreichen Feldern bestimmt ist. Diese sind neben der Kernphysik, mit der sich hier beschäftigt wird, auch die Atom-Physik, die Festkörperphysik, die Materialwissenschaft und Umweltwissenschaft. Die Einrichtung befindet sich beim Proton-Synchrotron Booster (PSB) der Europäischen Organisation für Nuklearforschung CERN. Im Betrieb werden radioaktive Nuklide durch Spallations⁵-, Spaltungs- oder Fragmentierungsreaktionen in einem dicken Target, das mit einem Protonenstrahl vom PSB bei einer Energie von ca. 1,4 GeV und einer Intensität von ungefähr $1,3 \cdot 10^3$ Protonen pro Puls bei einer Frequenz von 0,83 Hz bestrahlt wird, hergestellt. Die instabilen Produkte der nuklearen Reaktion werden vom Hochtemperatur-Target in die Ionenquelle durch chemisch selektive Prozesse abgegeben und als radioaktiver Ionenstrahl extrahiert. Laserionisation für bestimmte chemische Elemente erlaubt weitere Auslese und kann in einigen Fällen einen Strahl liefern, der sich schon in einem speziellen isomerischen Zustand befindet.⁶

Anschließend an den ISOLDE-Massenseparator ist der REX-Beschleuniger (Radioactive beam Experiment) gekoppelt, mit dem sich massenseparierte, einfach geladene und radioaktive Ionen auf effiziente Weise bündeln, weiter ionisieren und nachbeschleunigen lassen (siehe Abbildung 1). Das Bündeln des radioaktiven Ionenstrahls ist einerseits nötig für die Zeit zur weiteren Ionisation, die typischerweise zwischen 20ms und 200ms liegt, andererseits auf Grund des Betriebszyklus des Linearbeschleunigers und zuletzt, um das Verhältnis von Signal zu Untergrund bei Messungen mit geringen Ionenstrahlintensitäten zu verbessern.⁷

⁴ „<http://isolde.web.cern.ch/ISOLDE/>“

⁵ „**Spallation** (von engl. *to spall* „absplitteln“) ist eine nichtelastische Wechselwirkung eines [Atomkerns](#) mit einem Projektil ([Neutron](#), [Proton](#), einem anderen Kern oder einem [Elementarteilchen](#)) hoher kinetischer [Energie](#) (> 100 [MeV](#)). Der Atomkern wird dabei praktisch zerschmettert; in kleinere Bruchstücke und in der Regel mehrere Neutronen.“
(Wikipedia - Spallation, 2011)

⁶ (The Isolde facility)

⁷ (Walle, 2006), S. 48

Das REX besteht aus einer Penning-Falle (REX-TRAP), einer Elektronenstrahl-Ionenquelle (EBIS – Electron Beam Ion Source), einem Massen-Ladungs-Separator (A/q Separator) und einem Linearbeschleuniger (REX-Linac).⁸

In der Penning-Falle werden die radioaktiven Ionen durch eine hohe Spannung und Bremsgas (Argon oder Neon) auf wenige eV abgebremst, um die transversale Emittanz des Strahls zu verbessern. Danach werden die Ionen in Bündeln aus der Falle extrahiert, wieder beschleunigt und in die EBIS geleitet. Diese Station dient dazu, die Ionen durch Elektronenbeschuss-Ionisation weiter zu ionisieren. Mit einer Elektronenkanone wird ein Elektronenstrahl monoenergetischer Elektronen erzeugt und mit dem Magnetfeld eines supraleitenden Magneten radial fokussiert. Dabei werden die Ionen aufgrund der Potentialabfalls, hervorgerufen durch die negative Raumladung der Elektronen, radial und mittels Potentialbarrieren durch zylindrische Elektroden longitudinal gebunden. Abhängig von der Zeit, die nötig ist, um den gewünschten Ionisationsgrad der untersuchten Ionen zu erreichen, werden Ionenpakete von der EBIS in den Massen-Ladungs-Separator eingespeist.⁹

Der Ionenstrahl, der aus der EBIS kommt, enthält unerwünschte radioaktive Ionen, die in der ISOLDE erzeugt wurden, restliche Gasionen von der REX-TRAP und die untersuchten radioaktiven Ionen. Um diese zu trennen, wird eine Massentrennung vorgenommen, die mit dem richtigen Massen-Ladungsverhältnis übereinstimmt. Zuerst wird der Strahl elektrostatisch nach seiner Energie aufgeteilt, da die Ionen durch den Elektronenstrahl in der EBIS eine breite Energieverteilung besitzen. Dann durchläuft der Strahl einen Ablenkmagneten, mit dem man das optimale Masse-Ladungs-Verhältnis in der Brennebene justieren kann. Nach der Separation erfolgt die Beschleunigung der Ionen im REX-Linac, von dem aus der Strahl auf ein Target gelenkt wird.¹⁰

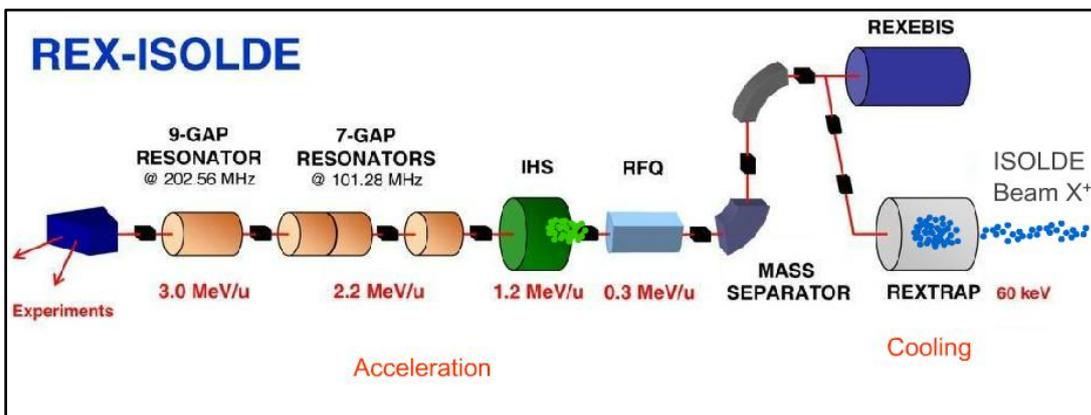


Abbildung 1: Der REX-ISOLDE-Beschleuniger¹¹

⁸ (Walle, 2006), S. 48 ff.

⁹ (Walle, 2006), S. 48

¹⁰ (Walle, 2006), S. 50 ff.

¹¹ (REX-ISOLDE, 2011)

2.2 Die Zeitstruktur von REX-ISOLDE

Zur späteren Analyse von Messergebnissen und zum Triggern der Messungen gibt es mehrere Timing-Signale, die zum Arbeitsablauf von REX-ISOLDE in Beziehung stehen.

Zuallererst gibt es ein Signal, das zu jedem Beginn eines neuen „Supercycles“ des Proton-Synchrotron Boosters gesendet wird. Die Periode davon ist immer ein Vielfaches von 1,2 Sekunden. Wird Laser-Ionisation im Target benutzt, kann damit ein Verschlussmechanismus gesteuert werden, der das Laserlicht auf die dünne Ionisationsröhre blockiert. In der Datenaufnahme wird die Stellung dieses Verschlusses aufgezeichnet.

Der zweite Zeitpuls ist der T1 Protonenpuls. Er wird jedes Mal erzeugt, wenn die Hochspannung des ISOLDE abgeschaltet wird und gibt an, dass der Protonenstrahl auf das Primärtarget auftreffen wird. Von diesem Moment an werden also mehr radioaktive Ionen aus dem Target diffundieren. Indem man Bedingungen an die Zeitdifferenz zwischen Teilchenankunft und Protoneneinfall stellt, werden Informationen über das ISOLDE Strahlen-Gatter erschlossen. Stark zu T1 in Beziehung stehend gibt es das T2 Signal, das den Anstieg der Hochspannung nach dem Protoneneinfall anzeigt. Jedoch wird dieses Signal in laufenden Experimenten nicht verwendet.

Die letzte Zeitinformation (im Folgenden EBIS-Puls genannt) hängt vom Auslass der Teilchen aus der EBIS in den Linearbeschleuniger ab und wird verwendet, um die Datenaufnahme für eine „Im Strahl“-Messung zu triggern (On-Beam Window). Zudem wird damit auch der REX-Linac mit den ankommenden Paketen der Teilchen synchronisiert.¹²

Das genannte On-Beam Window wird in regelmäßigen Abständen von 20-200ms, also der Brutzeit für den gewünschten Ladungszustand in der EBIS, generiert. Obwohl alle Ionen in einem Zeitfenster von ca. 100-400 μ s aus der EBIS extrahiert werden, wählt man für die eigentliche Messung zur Sicherheit ein On-Beam Window der Länge 800 μ s.

2.3 Der Miniball-Aufbau

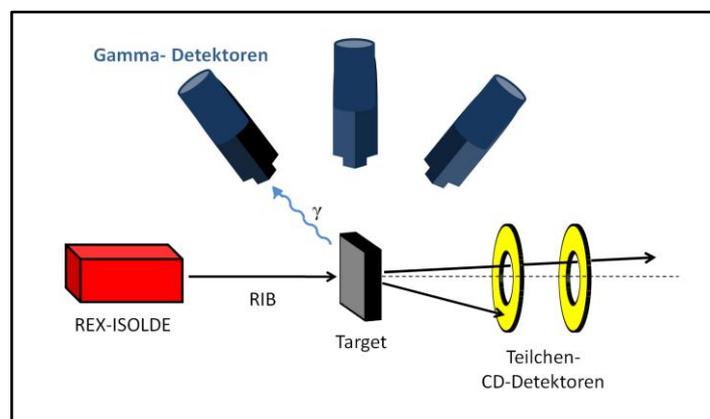


Abbildung 2: Der Miniball-Aufbau

¹² (Walle, 2006), S. 54, f.

Die Gamma-Detektoren

Der Miniball-Aufbau besteht aus hochauflösenden Germaniumdetektoren, die in einer geschlossenen Geometrie aufgebaut sind, um die totale Effizienz zu maximieren. Weitergehende Segmentierung der Germaniumkristalle und eine Pulsform-Verarbeitung erhöhen die Granularität und bieten die Möglichkeit der Dopplerkorrektur von Gammastrahlung, die aus den sich schnell bewegenden Teilchen emittiert wird. Insgesamt existieren 8 Cluster, in denen sich je drei individuell ummantelte Hyper Pure Germanium-Kristalle befinden. Jeder Kristall ist zudem elektrisch in 6 Teile segmentiert.

Die vorverstärkten Signale aus allen 24 Kristallen („Cores“) und den 144 Segmenten werden zu den „Digital Gamma Finder“ (DGF¹³) gesendet. Pro Kristall werden zwei DGF-Module verwendet. Von dort wird das vorverstärkte Signal vom Core zu einem „Master Trigger“ Kanal geschickt, während die übrigen 7 Inputs der DGF-Module für die 6 Segment Signale benutzt werden (1 Eingang bleibt unverbunden). Diese Belegung der Signale wird für die Pulsform-Analyse der verschiedenen Segmente verwendet. Der Core-Kontakt nimmt immer ein wechselwirkendes Gammateilchen wahr, wobei es vom Wechselwirkungspunkt im Kristall abhängt, ob ein Segment das Signal detektiert oder nicht. Der DGF-Kanal, der mit dem Core-Signal belegt ist, dient als Trigger für das Auslesen der zugehörigen Segmentkanäle. Auf diese Weise existiert eine direkte Korrelation zwischen der aufgenommenen Core-Energie und der Segmentenergie. Die Segmentinformation wird zur Dopplerkorrektur von Gammastrahlung, die während des Fluges emittiert wird, genutzt. Es wird angenommen, dass die erste Wechselwirkung der Gammastrahlung, welche die Emissionsrichtung des Gammaquants festlegt, mit der Hauptwechselwirkung (Wechselwirkung, bei der die meiste Energie in einem Compton-Ereignis deponiert wird) übereinstimmt.

Der Teilchendetektor

In der Targetkammer sind je nach Experiment verschiedene hochsegmentierte doppelseitige Siliziumstreifen-Detektoren, CD-Detektoren genannt, angebracht. Diese Detektoren bestehen aus 4 identischen und unabhängigen Quadranten, die jeweils wieder aus 16 ringförmigen (Vorderseite) und 24 radialen Streifen (Rückseite) zusammengesetzt sind (siehe unten Abbildung 3: Der CD-Detektor, A: Schematische Zeichnung, B: Bild des CD mit einem montierten, ringförmig segmentierten Quadranten)¹⁴.

Der Teilchen-Bereich wird nicht vollständig digital ausgelesen. Um die Teilchen- und Gammainformationen zu verknüpfen, wird die Verarbeitung des zeitlichen Ablaufs des CD Energie-Signals, wie die Gamma-Signale, mit DGF-Modulen durchgeführt. Das Signal wird zu CAEN¹⁵ ADCs¹⁶ gesendet, die analoge Peaks erfassen. Die vier ADCs arbeiten unabhängig von einander. Bei jedem akzeptierten Teilchen-Signal, das zu einem ADC gesendet wurde,

¹³ X-Ray Instrumentation Associates, „http://www.xia.com/DGF_products.html“

¹⁴ (Walle, 2006), S. 63

¹⁵ „<http://www.caentechnologies.com/>“

¹⁶ Analog-to-Digital-Converter, (Wikipedia - Analog-Digital-Umsetzer, 2011)

wird ein Block-Puls zu einem Eingangskanal eines DGF-Moduls gesendet. Somit wird der Zeitpunkt des Teilchens als digitaler Zeitstempel in einem zugehörigen DGF-Modul gespeichert. Die Zeitzuordnung zwischen Gammas und Teilchen ist in der Offline-Analyse auf einen Abgleich dieser digitalen Zeitstempel beschränkt.¹⁷

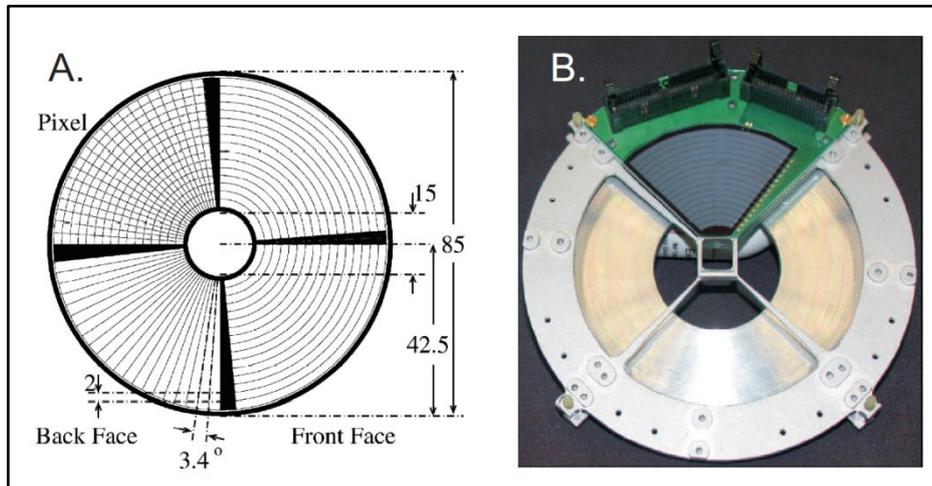


Abbildung 3: Der CD-Detektor, A: Schematische Zeichnung, B: Bild des CD mit einem montierten, ringförmig segmentierten Quadranten¹⁸

2.4 Die auftretenden Signale und deren Eigenschaften

Da der Teilchenstrahl aufgrund der obig beschriebenen Funktionsweise der EBIS gepulst ist, gibt es im echten Experiment ein Signal, das den Start jedes Teilchenstrahl-Pulses signalisiert. Dieses Signal ist der oben diskutierte EBIS-Puls genannt. Aus diesem Puls wird im Trigger-System ein Gate, das On-Window, generiert, das ein Zeitfenster von 800µs beschreibt, in dem Teilchen auf das Target auftreffen und somit auch detektiert werden können. Die Messdatenaufnahme beschränkt sich somit auf dieses On-Window und ein später folgendes Off-Window, das die gleiche Länge besitzt und zur Untergrundmessung dient, also nur in Zeiten auftreten kann, in denen der REX-Beschleuniger keinen Teilchenstrahl liefert. Innerhalb dieser Fenster werden die gemessenen Daten in den Analog zu Digital Wandlern (ADC) gespeichert. Die Zeiten zwischen diesen Zeitfenstern werden jeweils dafür genutzt, die Daten aus den Puffern auszulesen, gegebenenfalls weiterzuverarbeiten und dann zum Datenaufnahme-Rechner zu speichern. Damit sind die Puffermodule im nächsten Messfenster wieder voll aufnahmefähig. Das Verfahren besitzt den Vorteil, dass sich die Totzeiten durch Datentransport auf die Zeiten beschränken, in denen kein Strahl vorhanden ist. Damit gehen nahezu keine Messdaten wegen Pufferauslastung oder Auslesetotzeiten verloren, was die Effektivität des Messsystems enorm erhöht.

Vom Experiment wird ein weiteres logisches Signal empfangen, das T1-Signal. Es signalisiert, wenn Protonen das ISOLDE-Target treffen, und dient später zur Erzeugung eines T1-Zeitstempels. Analog dazu gibt es ein Signal, das PS-Signal genannt wird. Dieses wird

¹⁷ (Walle, 2006), S. 65 f.

¹⁸ (Walle, 2006), S. 63

gesendet, wenn der Protonen Supercycle startet, was bedeutet, dass die Extraktionsreihenfolge im Synchrotron von vorne beginnt. Daraus wird später ebenfalls ein PS-Zeitstempel produziert, der zur Überprüfung der T1-Zeitstempel hergenommen werden kann.

Wie oben erwähnt, werden die Messdaten bis zum Auslesen in Digital-Modulen gepuffert. Wenn während eines On- oder Off-Beam Windows der Speicher eines dieser Module voll ist, wird ein Signal gesendet, das das Auslesen startet und im jeweiligen Fenster alle weiteren ADC-Gates verbietet. Dieser Fall sollte im Normalbetrieb nur sehr selten auftreten, da er die spätere Analyse der Daten verkompliziert. Dieses Signal wird DGF-Busy genannt und als Synch-Signal weiter an alle anderen DGF-Module verteilt (die Bezeichnungen DGF-Busy und Synch werden im Folgenden äquivalent gebraucht), um das Auslesen zu synchronisieren. Das Signal muss ebenfalls im Testsystem simuliert werden, da es sich hierbei um Module handelt, die sich nicht innerhalb des Trigger-Systems befinden, und dieses Signal als Veto in die Logik der Trigger-Systems eingeht. Das Synch-Signal wird im Normalfall vom Trigger-System immer dann selbst erzeugt, wenn am Ende des On- und Off-Windows jeweils ein Forced Readout stattfindet, der das Readout-Fenster startet.

Zudem gibt es ein ADC-Busy-Signal, das aktiv ist, wenn die ADC-Module, die die Teilchendetektoren auslesen, beschäftigt sind. Hier wird individuell für jedes Segment, das von einem unabhängigen ADC ausgelesen wird, ein $14\mu\text{s}$ langes Totzeit-Signal generiert, das die sogenannte interne Konversionszeit abbildet.

Wenn man sich weiter mit der Elektronik beschäftigt, findet man hinter den Digital-Modulen die Data Acquisition (deutsch: Datenaufnahme), kurz DAQ, die das Speichern und Sortieren der Messdaten übernimmt. Diese sendet beim Auslesen ein Deadtime-Signal, das anzeigt, dass gerade Daten aus den Puffern ausgelesen werden und somit währenddessen auch keine neuen Daten geschrieben werden sollen. Es geht ebenfalls in die Logik des Trigger-Systems ein, muss also auch simuliert werden. Die Bedingung dafür ist jedoch ein DAQ-Trigger, der erzeugt wird, wenn das Synch-Signal eingeschaltet ist, wenn also entweder der Speicher eines Puffer-Moduls voll ist oder vom Trigger-System ein Forced Readout gesendet wird.

Jedes nachgewiesene Teilchen oder Gammaquant erzeugt weiterhin einen Trigger, der im Experiment zufällig, also irregulär, auftritt. Dabei sind die Gammatrigger deutlich – etwa um den Faktor 1000 – häufiger als die Teilchentrigger. An die Trigger-Box wird im Experiment ein „ODER“ aller Gammatrigger gesendet, aus dem ein Gate von 800ns Länge erzeugt wird. Diese Zeit kann aufgrund der langsamen Anstiegszeit der Signale in den Germanium-Detektoren nicht stark verkürzt werden. Da der Teilchendetektor bei Coulex-Experimenten in 4 Segmente zusammengefasst wird, gibt es vier Teilchen-Trigger.¹⁹

¹⁹ (Warr, 2010)

3. Die Verwendeten Hilfsmittel

3.1 Das Modul

Für die Realisierung des Testsystems wurde ein VULOM3 verwendet, das vom GSI (Gesellschaft für Schwerionenforschung mbH) entwickelt wurde. Es besitzt jeweils 18 Eingänge und Ausgänge, von denen je 16 ECL²⁰-Anschlüsse und zwei davon LEMO^{®21}-Buchsen für negative NIM-Logik sind. Zusätzlich gibt es noch 16 Inputs/Outputs – ebenfalls ECL-Anschlüsse – die sich je nach Bedarf als Ein- oder Ausgänge definieren lassen. Das Herzstück des Moduls ist ein FPGA²² (Field Programmable Gate Array), ein „Integrierter Schaltkreis der Digitaltechnik, in den eine logische Schaltung programmiert werden kann“²³ und der mit 100MHz getaktet ist (entspricht einem Clock-Zyklus von 10ns).²⁴

3.2 Die Firmware

Da die spezielle Programmierung eines solchen FPGA-Moduls eine fundierte Kenntnis der Hardware-, aber auch aller Software-Schnittstellen erfordert, kann diese nur vom Entwickler selbst oder von Experten, die von diesem ausgewiesen sind, durchgeführt werden. Um eine Programmierung der Funktionalität auch für die Nutzer zu erlauben und ein Höchstmaß an Flexibilität zu erreichen, wurde bei der Programmierung auf die Firmware von Håkan Johansson, einem Entwickler vom GSI, zurückgegriffen, die schon in der digitalen Triggerlogik des LAND/R³B des GSI zur Verwendung gekommen ist.²⁵ Diese Firmware lieferte schon die benötigten Funktionen und ermöglichte deren Konfiguration mit Hilfe eines C-Programms:

Das Grundwerkzeug hierbei ist ein sogenannter Multiplexer (Schema unten in Abbildung 4 dargestellt), mit dem sich Eingänge, Funktionen und Ausgänge nahezu beliebig verbinden lassen. In dieser programmierbaren Schaltmatrix gibt es einen Katalog an „Sources“, den Quellen der Signale, wie z.B. einem ECL-Eingang oder einem Funktions-Ausgang, sowie einen weiteren Katalog an „Destinations“, den Zielen, an welche die Signale geleitet werden können, z.B. ein ECL-Ausgang oder ein Funktions-Eingang.

²⁰ Emitter-Coupled-Logic, (Wikipedia - Emittergekoppelte Logik, 2011)

²¹ „<http://www.lemo.com/defaultwidgets.asp>“

²² FPGA: Typ VIRTEX-4 LX25 – XILINX, „<http://www.xilinx.com/support/documentation/virtex-4.htm>“

²³ (Wikipedia - Field Programmable Gate Array, 2011)

²⁴ (Hoffmann, 2008)

²⁵ (Henriques, 2010)

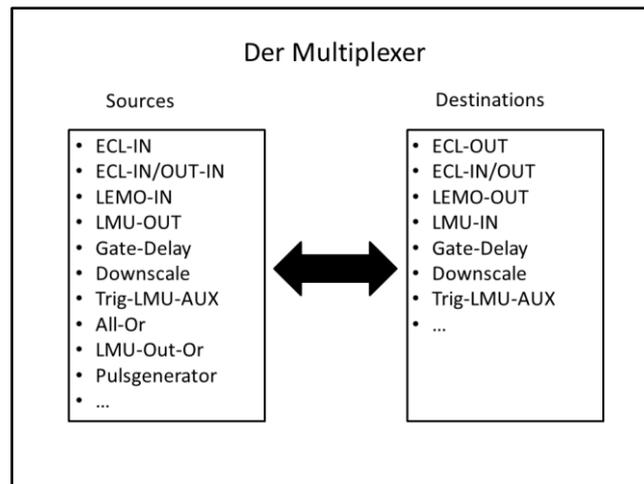


Abbildung 4: Der Multiplexer

Die einzelnen Funktionen kann man sich als voneinander unabhängige Module vorstellen, die je nach ihrer Art einen oder mehrere definierbare Parameter besitzen. So gibt es Pulsgeneratoren, die kurze Pulse mit einstellbarer Periode erzeugen und auch Gate-Delays, eine Kombination aus Gategenerator und Delay, die dazu dient, aus Pulsflanken Gates zu erzeugen und bzw. oder diese um eine gewünschte Zeit zu verzögern. Fan-In/Fan-Outs („All OR“), die ein logisches Oder von mehreren Signalen erzeugen, und Scaler, mit denen sich die Pulse jeder Quelle des Multiplexers zählen und latchen lassen. Eine etwas komplexere Funktion ist die Logic-Matrix-Unit, im Folgenden als LMU bezeichnet. Sie besitzt mehrere Ein- und Ausgänge, wobei sich für jeden Ausgang eine von acht Logikkombinationen der acht Eingänge definieren lässt (siehe Kap. 4.2, „Die Logic-Matrix-Unit (LMU)“, S.15) und ein Ausgangssignal mit einem Gate-Delay analog zu dem schon genannten verarbeitet werden kann.

Zusätzlich gibt es noch eine weitere LMU, die in der Triggerlogik-Funktion der Firmware eingebettet ist. Diese Funktion wurde bis auf diese Trigger-LMU, die aus Gründen der begrenzten Kanalanzahl der „externen“ LMU gebraucht wurde, nicht weiter genutzt und somit auch nicht ausführlicher diskutiert.²⁶ Die Konfiguration der LMU ist in Abschnitt 4.2 erklärt.

Es sind noch einige andere Funktionen in der Firmware vorhanden, jedoch werden hier nur die für das Testsystem relevanten und auch verwendeten Funktionen genannt und näher behandelt. Diese sind noch einmal mit ihrer verfügbaren Anzahl in Tabelle [Funktionstabelle] aufgelistet. Einen vollständigen, umfangreicheren Katalog an Funktionen und Quellen bzw. Zielen für den Multiplexer findet man in der von Håkan Johansson erstellten Dokumentation.²⁷

²⁶ (Johansson)

²⁷ TRLO II – description and definitions:
 „http://fy.chalmers.se/~f96hajo/trloii/descr_defs_frame.html“

4. Die Konfiguration des Moduls

Wie bereits erwähnt, lassen sich die Funktionen des Moduls durch eine Datei konfigurieren und aufrufen, die mit der Programmiersprache C geschrieben wird. Dazu werden C-Standardbibliotheken (siehe Anhang) und die Header-Dateien „vulom3defs.h“, in der Konstanten vordefiniert sind, und „trlo_defs.h“, in welchem ebenfalls Konstanten, Typdefinitionen und Makros beschrieben sind, genutzt. Das Programm wird dann auf der Kontroll-CPU des VME-Bus kompiliert – für diese Arbeit wurde ein RIO4 8070/72 von CES²⁸ mit dem Betriebssystem LynxOS 4.0 verwendet. Durch dessen Ausführung werden alle entsprechenden Register des Logikmoduls über den VME-Bus auf den FPGA übertragen. Damit ist die Funktionalität im Modul vorhanden, solange es mit seiner Versorgungsspannung verbunden ist.



Abbildungen 5 und 6: Links die Module am VME-Bus, rechts ein Test-Aufbau mit Oszilloskop.

4.1 Übertragung der Firmware

Für die Nutzung der Firmware ist es jedoch zuerst nötig, diese auf das VULOM3 zu übertragen. Dies geht von statten, indem man sich, nachdem das Modul in den VME-Bus gesteckt wurde, per „Remote-Shell“ auf der Kontroll-CPU einloggt, in das lokale Verzeichnis mit den Dateien von Håkan Johansson²⁹ (Liste im Anhang) wechselt und folgende Befehle ausführt:

```
eraseflash        YZ
```

Wobei hier **Y** die am Modul eingestellte VME-Adresse in hexadezimaler Schreibweise ist und **Z** (0-7) eine der 8 Speicherbänke des Moduls ist. Dabei sollte nicht Z=0 gewählt werden, da auf dieser Programm zum Starten des Moduls liegt und man somit das Modul

²⁸ „<http://www.ces.ch/>“

²⁹ Liste der dazu benötigten Dateien im Anhang [...]

unbrauchbar machen kann. Mit diesem Befehl wird alles, was in der Speicherbank Z ist, gelöscht.

Anschließend:

```
progflashxf YZ file1.rbt file2.txt
```

Hier sind **Y** und **Z** wieder dieselben Werte wie zuvor.

file1 entspricht dem jeweiligen Dateinamen der Firmware, wie zum Beispiel:

```
„vlogic_1_20100803_2306“ , also vlogic_1_20100803_2306.rbt
```

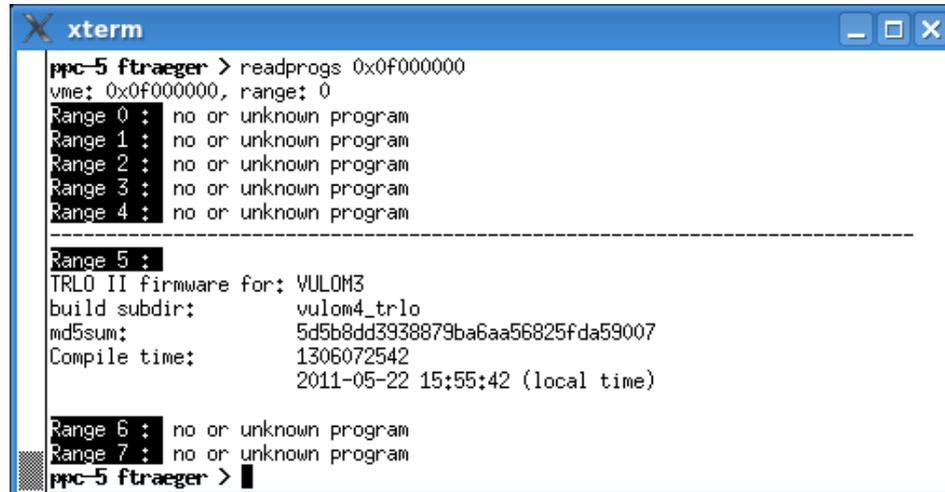
file2 dem Namen der Textdatei mit den Kompilierungsdaten, wie zum Beispiel:

```
„trlo_compile_20100803_2306“ , also trlo_compile_20100803_2306.txt.
```

Der nächste Schritt ist:

```
readprogs Y
```

Damit wird ausgelesen, in welchen Bänken sich welche Programme befinden. Möglicherweise muss der Befehl öfter hintereinander ausgeführt werden, damit das Lesen der Speicher funktioniert.



```
xterm
ppc-5 ftraeger > readprogs 0x0f000000
vme: 0x0f000000, range: 0
Range 0 : no or unknown program
Range 1 : no or unknown program
Range 2 : no or unknown program
Range 3 : no or unknown program
Range 4 : no or unknown program
-----
Range 5 :
TRLO II firmware for: VULOM3
build subdir:      vulom4_trlo
md5sum:           5d5b8dd3938879ba6aa56825fda59007
Compile time:     1306072542
                  2011-05-22 15:55:42 (local time)
Range 6 : no or unknown program
Range 7 : no or unknown program
ppc-5 ftraeger > █
```

Abbildung 7: Ausgabe nach dem Befehl "readprogs"

Wird etwas äquivalent zu der Ausgabe in Abbildung 7 (siehe oben) angezeigt, kann der FPGA wie folgt neu gestartet werden:

```
restart Y Z
```

4.2 Programmierung des Konfigurationsprogramms

Im Folgenden wird nun die Funktionsweise, der Aufruf und die Konfiguration der Funktionen des Moduls erklärt und das Ganze an einem kurzen Beispielprogramm näher

gebracht. Prinzipiell wird die Konfiguration so durchgeführt, indem in Arrays und Variablen, die in der Headerdatei „trlo_defs.h“ definiert sind, entweder eigene Werte oder vordefinierte Konstanten geschrieben werden, welche bei Programmausführung an das VULOM3 übertragen werden.

Der Multiplexer

Wie schon bereits erwähnt, lassen sich mit dem Multiplexer verschiedene Quellen (Sources) mit verschiedenen Zielen (Destinations) verbinden. Um dies nun im Programm zu bewerkstelligen, verwendet man folgende Zeile, mittels der ein vordefinierter Wert an eine bestimmte Position im Multiplexer-Array gespeichert wird:

```
hw->setup.mux[TRLO_MUX_DEST_XXX] = TRLO_MUX_SRC_XXX;
```

Damit wird die Quelle TRLO_MUX_SRC_XXX mit dem Ziel TRLO_MUX_DEST_XXX verbunden. Hier lassen sich alle Quellen bzw. Ziele aus bereits dem genannten Katalog³⁰ in der Dokumentation einsetzen.

Die Pulsgeneratoren

Die Periode der Pulsgeneratoren lässt sich in Prozessor-Clock-Zyklen, also 10ns-Schritten einstellen. Sie sind über den Multiplexer in Form einer Quelle verfügbar und es gibt insgesamt 8 voneinander unabhängige Pulsgeneratoren.

```
hw->setup.period[i] = n;
```

Hier ist für den Pulsgenerator i eine Periode von $n \cdot 10\text{ns}$ gesetzt. Für n sind Zahlenwerte bis zu $n=4294967295$, also ca. 42,95s, möglich (der mögliche einsetzbare Wert ist immer in der jeweiligen Maske, der Funktion definiert).

Das Gate-Delay

Die Länge und das Delay der Gate-Delays sind wie die Periode in 10ns-Schritten festzulegen. Zur Nutzung eines Gate-Delays muss das Signal mit Hilfe des Multiplexers mit dessen Eingang verbunden werden, das resultierende Signal kann an dessen Ausgang abgegriffen werden. Es sind in der Firmware 4 eigenständige Gate-Delays vorhanden.

```
hw->setup.stretch[0] = n;  
hw->setup.delay[0] = l;
```

Die obere Zeile beschreibt die Gate-Länge n ($n \cdot 10\text{ns}$), die untere das Delay l ($l \cdot 10\text{ns}$). Der maximal mögliche Wert für n und l ist jeweils 16383 (also 163,83 μs).

³⁰ (Johansson)

Zudem gilt es noch den sogenannten Restart-Modus der Gate-Delays zu bestimmen, der angibt, wie das Gate generiert wird.

```
hw->setup.restart_mode[0] = TRLO_RESTART_MODE_XXX;
```

Dazu gibt es vier Möglichkeiten, was für XXX eingesetzt werden kann:

LEADING_EDGE

Das Gate wird mit der steigenden Flanke des Eingang-Signals gestartet

TRAILING_EDGE

Analog zu LEADING_EDGE, nur dass hier mit der fallenden Flanke gestartet wird

LEAD_IF_INACT

Analog zu LEADING_EDGE, nur dass das Gate nicht neu gestartet wird, wenn es noch aktiv ist

WHEN_PRESENT

Das Gate ist so lang wie der Puls des Eingang-Signals und startet mit dessen steigender Flanke

Die Scaler

In der Firmware sind Scaler implementiert, mit denen sich die steigenden Flanken der Pulse aller Quellen des Multiplexer zählen lassen. Um diese Zähler auslesen zu können müssen sie zuerst über VME fixiert („gelatcht“) werden. Dies wird über einen Puls gemacht:

```
hw->pulse.pulse = TRLO_PULSE_MUX_SRC_SCALER_LATCH
```

Die Scaler lassen sich ähnlich auch zurücksetzen:

```
hw->pulse.pulse = TRLO_PULSE_MUX_SRC_SCALER_RESET
```

Für das Auslesen der gelatchten Scaler greift man auf den Scaler-Wert der entsprechenden Quelle des Multiplexers zu:

```
hw->scaler.mux_src [TRLO_MUX_SRC_XXX]
```

Die Logic-Matrix-Unit (LMU)

Um die LMU zu benutzen muss das Signal wieder durch den Multiplexer mit einem Eingang der LMU verbunden werden. Aus mehreren Signalen lässt sich dann für einen Ausgang eine von 8 verschiedenen Logikkombinationen (4 UND-Bedingungen und 4 ODER-Bedingungen, siehe Tabelle 1) erzeugen. Diese werden definiert, indem in entsprechende Array-Positionen bestimmte Werte geschrieben werden. Um eine der 4 UND-Bedingungen für

den Ausgang j der LMU zu verwenden, muss in der Variable **lmu_not** eine Binärzahl gespeichert werden, in der das j-te Bit gleich 1 gesetzt ist.

lmu_not = 1 << j;

Soll eine der 4 ODER-Bedingungen zur Verwendung kommen, muss an dieser Bit-Stelle (die j-te Stelle) eine 0 stehen.

Ähnlich wird eine Logikkombination definiert. In der Tabelle 1 werden Bit-Paare [k,l] für den jeweiligen Eingang benutzt, um die erforderliche Einstellung zu bezeichnen. Dabei sind k bzw. l entweder 0 oder 1. Die Kombination [0,1] bedeutet beispielsweise:

lmu_and[j] = 0 << i;

lmu_nand[j] = 1 << i;

Also muss im Binär-Wert von **lmu_and[j]** an i-ter Stelle eine 0 stehen und in **lmu_nand[j]** and i-ter Stelle eine 1.

Die Kombination hingegen [1,0] steht beispielsweise für Folgendes:

lmu_and[j] = 1 << i;

lmu_nand[j] = 0 << i;

Tabelle 1: Mögliche Logikkombinationen der LMU:

Die 4 UND-Bedingungen <i>lmu_not = 1 << j</i> (für Ausgang j)	
Durchgang ohne Bedingung: [0,0]	Ein Signal wird immer vom jeweiligen Eingang zum gewählten Ausgang durchgelassen
Koinzidenz erfordern: [0,1]	Am Ausgang erscheint dann nur ein Signal, wenn die gewählten Eingänge koinzident sind (Logisches „UND“ der Eingänge)
Anti-Koinzidenz erfordert: [1,0]	Signal nur dann am Ausgang, wenn die gewählten Eingänge nicht koinzident sind (Inversion des logischen UND der Eingänge)
Eingang nicht benutzen: [0,0]	Der gewählte Eingang hat keinerlei Einfluss auf das Ausgangssignal

Die 4 ODER-Bedingungen $lmu_not = 0 \ll j$ (für Ausgang j)	
Kein Output: [0,0]	Das Eingangssignal wird nicht durchgelassen
Logisches ODER: [0,1]	Das Ausgangssignal ist das logische ODER der gewählten Eingangssignale
Logisches ODER der negierten Eingänge: [1,0]	Das Ausgangssignal ist die Inversion des logischen ODERs der gewählten Eingangssignale
Immer an: [1,1]	Das Eingangssignal wird immer ohne Bedingung durch geleitet

Für jeden LMU-Ausgang gibt es zudem noch ein optionales Gate-Delay, das äquivalent zum schon beschriebenen Gate-Delay funktioniert. Zu beachten ist jedoch, dass hier für n und l im Gegensatz zum normalen Gate-Delay nur Werte bis zu 1023 (entspricht 10,23µs) eingesetzt werden dürfen.

```
hw->lmu_delay[j] = n;
hw->lmu_stretch[j] = l;
hw->lmu_restart_mode[j] = TRLO_LMU_RESTART_MODE_XXX ;
```

Da dieses Gate-Delay optional ist, kann es an- oder abgeschaltet werden:

```
hw->lmu_restart_mode[j] = TRLO_LMU_RESTART_GATE_ENABLE;
oder
```

```
hw->lmu_restart_mode[j] = TRLO_LMU_RESTART_GATE_DISABLE;
```

Die Trigger-LMU funktioniert analog zur externen LMU, nur gibt es hier 16 Eingänge, die nicht vorher verbunden werden müssen, sondern direkt den 16 ECL-Eingängen entsprechen, zudem noch 4 AUX-Eingänge, die man mit dem Multiplexer verbinden kann. Zudem lassen sich auch die 8 Eingänge der externen LMU mit in die Logikkombination für die 16 Ausgänge der Trigger-LMU mit einbeziehen.

Für Verwendung der AUX-Eingänge werden die entsprechenden Bits der Eingänge (Analog zu Tabelle 1) für den Ausgang j der Trigger-LMU gesetzt:

```
hw->trig_lmu_aux_and[j] = 1<<i;
hw->trig_lmu_aux_nand[j] = 1<<i;
```

Ebenso für die Eingänge der externen LMU:

```
hw->trig_lmu_mux_and[j] = 1<<i;
```

```
hw->trig_lmumux_nand[j] = 1<<i;
```

Die Ausgänge der Trigger-LMU lassen sich nicht direkt abgreifen, da sie, wie schon in Kapitel 3.2 erwähnt, in die Triggerlogik-Funktion des Moduls eingebettet ist. Jedoch gibt es hier auch Scaler, die gleich den anderen Zählern gelatcht (hier mit TRLO_TRIG_SCALER_LATCH) und per

```
hw->scaler.before_deadtime[j]
```

ausgelesen werden können (j ist der jeweilige Trigger-LMU Ausgang).

Das ALL OR (Logisches ODER)

Zur Verwendung des All OR werden vordefinierte Makros verwendet, um die Eingangssignale festzulegen. Dabei können alle Quellen des Multiplexers damit verknüpft werden, wobei die Anzahl an Eingängen nur auf die Anzahl der Quellen beschränkt ist:

```
TRLO_ALL_OR_SET(hw->setup.all_or_mask[i], TRLO_MUX_SRC_XXX1);  
TRLO_ALL_OR_SET(hw->setup.all_or_mask[i], TRLO_MUX_SRC_XXX2);
```

...

Der Ausgang des All OR ist die entsprechende Quelle **TRLO_MUX_SRC_ALL_OR(i)** des Multiplexers. Insgesamt gibt es 2 All OR Kanäle.

Beispielprogramm

Um den Aufbau eines Programms anschaulich zu machen, ist im Anhang (siehe Anhang, Teil A. S. 30) ein kurzes Beispiel-Programm eingefügt. In dem genannten Programm wird mittels eines Pulsengenerators ein Puls einer Periode von 3µs erzeugt und durch den Multiplexer mit zwei Gate-Delays und einem ECL-Ausgang verbunden. Das eine Gate-Delay erzeugt ein Gate der Länge 500ns und das andere verzögert dieses Gate wiederum um 1µs. Die Signale wurden durch den Multiplexer ebenfalls mit zwei Ausgängen verbunden, um diese auszugeben. Mit einem Oszilloskop wurden die Ausgänge aufgezeichnet, wie unten in Abbildung 8 dargestellt. Dabei erscheinen die Signale nicht zeitlich akkurat, da der Multiplexer 2 Clock-Zyklen benötigt (also 20ns, der Prozessortakt beträgt 100 MHz), um eine Quelle mit einem Ziel zu verbinden.³¹

³¹ (Henriques, 2010), S. 56

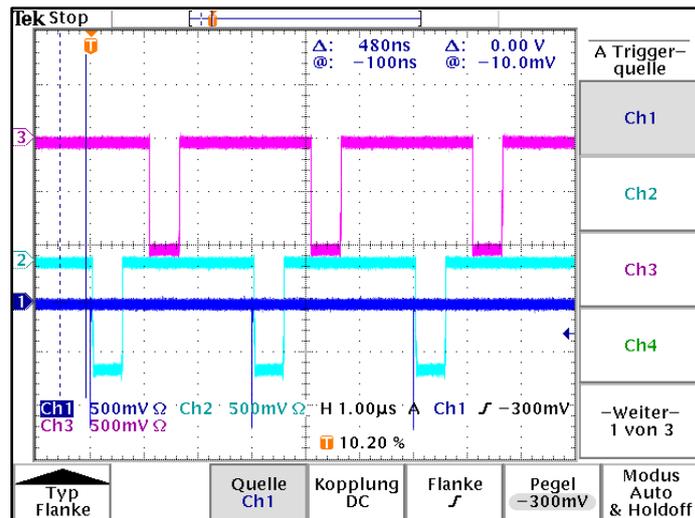


Abbildung 8: Das dunkelblaue Signal ist der vom Pulsgenerator erzeugte Puls, das türkise ist damit erzeugte Gate und das lilafarbene Signal zeigt das verzögerte Gate.

Im Programm findet man nach den Bibliotheks- und Headerdatei-Einbindungen einige Funktions- und Konstanten-Definitionen, die für das richtige Ansteuern des Moduls über den VME-Bus nötig sind und somit erhalten bleiben sollten. Die Funktion, mit der das System konfiguriert wird, wurde in diesem Beispiel mit „BeispielConfig“ benannt. Zu Beginn wurden mit der Funktion

```
clean_state(hw);
```

alle Einstellungen zurückgesetzt, um mögliche Probleme mit alten Einstellungen zu vermeiden. Dann wurde die Periode des verwendeten Pulsgenerators definiert. Diese beträgt hier $3\mu\text{s}$, der Puls hat also eine Frequenz von ca. $0,33\text{kHz}$:

```
hw->setup.period[0] = 300;
```

Dieser Pulsgenerator wird durch den Multiplexer mit dem ECL-Ausgang 0 verbunden:

```
hw-> setup.mux[TRLO_MUX_DEST_ECL_OUT(0)] = TRLO_MUX_SRC_PULSER(0);
```

Als nächstes wurde der Pulser mit dem Gate-Delay 0 verbunden, die Länge auf 500ns und der Restart-Modus des Gates auf „steigende Flanke“ eingestellt. Der Ausgang des Gate-Delays wurde mit dem ECL-Ausgang 1 verbunden:

```
hw->setup.mux[TRLO_MUX_DEST_GATE_DELAY(0)] = TRLO_MUX_SRC_PULSER(0);
```

```
hw->setup.stretch[0] = 50;
```

```
hw->setup.restart_mode[0] = TRLO_RESTART_MODE_LEADING_EDGE;
```

```
hw->setup.mux[TRLO_MUX_DEST_ECL_OUT(1)] = TRLO_MUX_SRC_GATE_DELAY(0);
```

Als letztes wurde der Ausgang des Gate-Delay 0 noch mit dem Eingang des Gate-Delay 1 verbunden, dessen Delay auf eine Zeit von $1\mu\text{s}$ und der Restart-Modus auf WHEN_PRESENT gesetzt, und der Ausgang des Gate-Delay 1 mit dem ECL-Ausgang 2 verbunden:

```
hw->setup.mux[TRLO_MUX_DEST_GATE_DELAY(1)] = TRLO_MUX_SRC_GATE_DELAY(0);
```

```
hw->setup.delay[1]= 100;
```

```
hw->setup.restart_mode[1] = TRLO_RESTART_MODE_WHEN_PRESENT;
```

```
hw->setup.mux[TRLO_MUX_DEST_ECL_OUT(2)] = TRLO_MUX_SRC_GATE_DELAY(1);
```

In der "main"-Funktion wird in der ersten Zeile der Offset für die VME-Adressen definiert. Hier ist die am VULOM3 eingestellte VME-Adresse einzugeben.

5. Simulation der Signale

Nun geht es darum, die in Kapitel 2.4 beschriebenen Signale mittels der vorhandenen Logikfunktionen, die schon in der Trigger-Logik Firmware für das Vulom3-Modul vorhanden sind, möglichst wie im echten Experiment zu simulieren. Da zahlreiche Eigenschaften des gesamten Systems gleichzeitig implementiert werden müssen, ist mit den begrenzten Ressourcen im VULOM sehr sparsam umzugehen. Alle hier simulierten Signale werde durch ECL-Ausgänge zum Trigger-System gesendet.

Um den EBIS-Puls zu erzeugen, wird einer der eingebauten Pulsgeneratoren verwendet. Da das Signal keine bestimmte Länge besitzen muss, wird es direkt mit einem der Ausgänge des Moduls verbunden. Entsprechend wird mit den T1- und PS-Pulsen verfahren, da diese ebenso ablaufen. Sie können korreliert sein, was aber für das Test-System nicht wichtig ist. Die Periode der Pulse ist durch eine Konfigurationsdatei zwischen 10ns und ca. 42,95s variabel (näheres dazu in Kapitel 7).

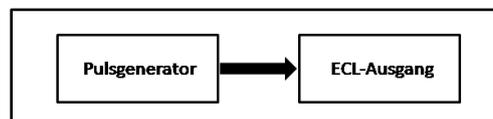


Abbildung 9: Schema der Erzeugung des EBIS-Pulses, T1-Pulses und PS-Pulses

Die Gammatrigger und Teilchentrigger werden ähnlich dazu simuliert, nur sollten sie möglichst zufällig erzeugt werden. Leider sind bisher noch keine Zufalls-Pulsgeneratoren im Modul vorhanden, deren Frequenzspektrum einstellbar ist. Solche sind jedoch theoretisch möglich. Deshalb werden hier vorerst ebenfalls die Pulsgeneratoren mit einstellbarer Periode hergenommen. Die vier Teilchentrigger sind auf Grund der begrenzten Anzahl an Pulsgeneratoren zu zwei Gruppen zusammengefasst, die jeweils vom selben Pulsgenerator gespeist werden. Somit lässt sich nur die Periode von diesen Zweiergruppen ändern, jedoch kann man zudem ein Delay zwischen den zwei Teilchentriggern einer Gruppe definieren (siehe unten Abbildung 12). Die Längen der Trigger-Pulse lassen sich mit Gate-Delays nahezu beliebig verändern. Da nur 4 unabhängige Gate-Delays zur Verfügung stehen, werden der Gamma-Puls sowie die Teilchen-Pulse 1 und 3 mit Gate-Delays der LMU verarbeitet und lassen sich somit auf eine Pulslänge von bis zu 10,23 μ s einstellen. Die Teilchentrigger 2 und 4 werden durch die „externen“ Gate-Delays definiert, da damit längere Delays (bis zu 163,83 μ s) erzeugt werden können, also eine erhöhte Unabhängigkeit von Teilchentrigger 1 und 3 bei der Einstellung gewährleistet ist. Durch diese Anordnung lassen sich die Trigger näherungsweise zufällig erzeugen.

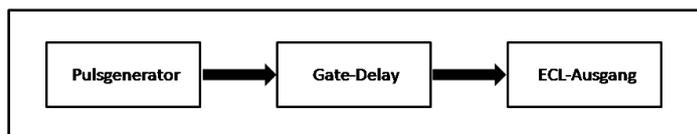


Abbildung 10: Erzeugung der Gammatrigger

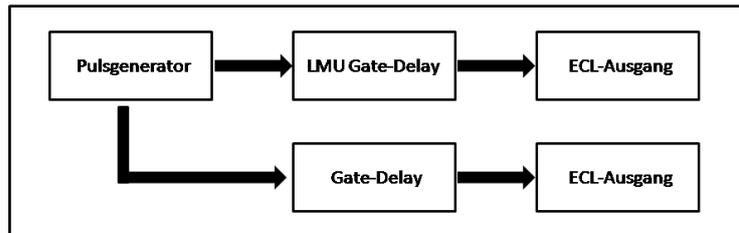


Abbildung 11: Erzeugung eines der beiden Teilcentrigger-Paare

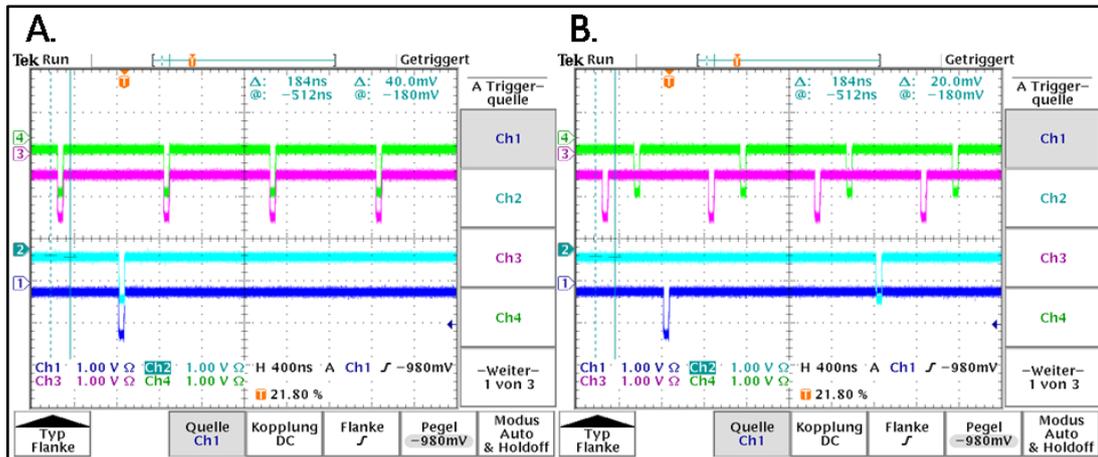


Abbildung 12: A. Teilcentrigger-Paare 1 und 2 (unten in blau und türkis) bzw. 3 und 4 (oben in grün und lila) mit jeweils gleicher Periode; B. Teilcentrigger 2 und 4 jeweils mit Delay

Da die DAQ-Deadtime nur auftritt, wenn ein DAQ-Trigger vorangegangen ist, wird dieses Signal vom Trigger-System abgegriffen und daraus ein Gate erzeugt, dessen Länge und Verzögerung variabel ist (jeweils bis zu 163,83 μ s). Dazu wird eines der vier Gate-Delays benutzt, da damit längere Gates verwirklicht werden können.

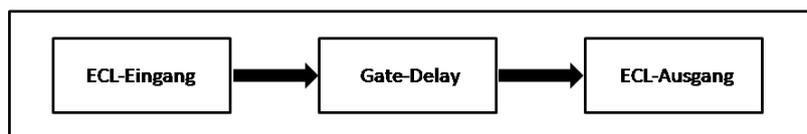


Abbildung 13: Erzeugung der DAQ-Deadtime

Das DGF-Busy- bzw. Synch-Signal kann wie beschrieben vom Trigger-System ausgelöst oder auch unerwartet auftreten, wenn ein Puffer voll ist. Da im Vulom3-Modul wie oben bereits erwähnt kein Zufalls-Pulser vorhanden ist, wird hier wieder ein vordefinierbarer Pulsengenerator benutzt. Als Periode wird hier eine große Zeit empfohlen, da – wie in Kapitel 2.4 erläutert – der Fall eines aktivierten Synch-Signals unabhängig vom Trigger-System nur sehr selten eintritt. Zusätzlich wird mit dem „Forced Readout“-Signal analog zur DAQ-Deadtime ein Gate ausgelöst. Von diesen beiden Signalen wird mit der LMU das ODER erzeugt und dies dann als Synch-Signal ausgegeben. Mit der LMU werden aus den Pulsen ebenfalls Gates mit einstellbarer Länge generiert.³²

³² (Warr, 2010)

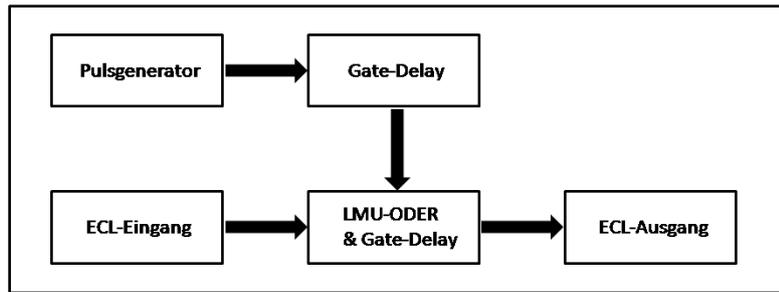


Abbildung 14: Erzeugung des Synchron-Signals

Das letzte fehlende Signal ist das ADC-Busy-Signal. Es wird von den ADC-Gates des Trigger-Systems hervorgerufen. Dazu wird aus den ADC-Gates ein ODER mit einem der beiden „All OR“-Funktionen gebildet und mit diesem ODER mittels eines Gate-Delays Gates mit definierbarer Länge gestartet.

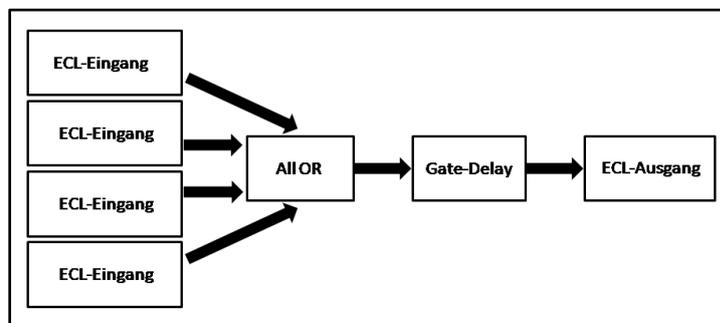


Abbildung 15: Erzeugung des ADC-Busy-Signals

Im Anhang Teil C finden sich die für die Simulation der Signale verwendeten Funktions- und Ein-/Ausgangsbelegungen.

6. Analyse der Signale

Das Wichtige an einem funktionierenden Trigger-System ist, dass einerseits die ausgegebenen Signale nur dann auftreten, wenn diese erwartet werden, und andererseits keine Ereignisse verloren gehen, auf die getriggert werden soll.

Da in der Trigger-Logic-Firmware nur begrenzte Mittel zur Analyse verfügbar sind, werden zur Überprüfung von Ereignissen, die eine Anti-Koinzidenz erfordern, das heißt nicht gleichzeitig auftreten dürfen, eine Koinzidenz detektiert und gezählt. Wird ein solches Gegenereignis gezählt, also registriert, wird eine Zeile ausgegeben, die den jeweiligen Fehler beschreibt und die Anzahl der gezählten Gegenereignisse ausgibt. Die Anti-Koinzidenzen und Koinzidenzen der verschiedenen Signale werden aus Platzgründen – nahezu alle Eingänge der LMU sind schon belegt – mit der Trigger-LMU erzeugt und mit deren Scaler gezählt (beschrieben in Kapitel 4.2). Wird auf diese Weise ein Fehler entdeckt, gibt das Programm eine Zeile mit der entsprechenden Fehlermeldung und die Anzahl der entdeckten Fehler aus.

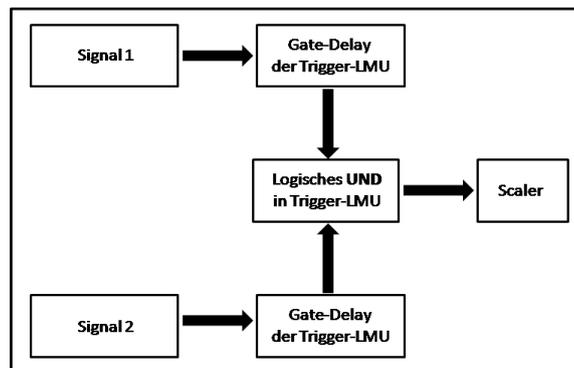


Abbildung 16: Zählen von Koinzidenzen zweier Signale, um zu überprüfen, ob sie ungewollt gleichzeitig auftreten

Zuallererst darf vom Trigger-System kein Forced Readout im On- oder Off-Window gesendet werden, da dies dazu führen würde, dass das Auslesen der Daten startet. Dies würde lange Totzeiten der Module auslösen, wodurch die Messung nicht über das gesamte On- bzw. Off-Window erfolgen könnte. Das erschwert die spätere Analyse und Weiterverarbeitung der Daten immens. Es darf also keine Koinzidenz des „Forced Readout“-Signals mit dem On- oder Off-Window entstehen. Um dies zu überprüfen, wird in der Trigger-LMU eine UND-Bedingung des „Forced Readout“-Signals mit dem On-Window und parallel dazu auch mit dem Off-Window vom Trigger-System definiert und die Pulse der beiden resultierenden Signale gezählt.

Des Weiteren sollen weder ADC-Gates ankommen, wenn das ADC-Busy-Signal an ist, da dann die Daten nicht aufgenommen werden können, und noch, wenn das Synch-Signal aktiv ist. Hierzu wird das von der Simulation des ADC-Busy-Signals bereits vorhandene ODER der Trigger (siehe Kapitel 5, letzter Absatz) hergenommen, damit durch die Trigger-LMU ein UND mit jeweils dem ADC-Busy-Signal und dem Synch-Signal generiert und wiederum die Ereignisse gezählt werden.

Schlussendlich gilt es noch zu kontrollieren, ob alle ADC-Gates, die erzeugt werden sollen, auch vom Trigger-System gesendet werden. Dafür werden diese Gates parallel dazu im Test-System selbst erzeugt und mit denen des Trigger-Systems verglichen. Es wird ein ODER der Teilchentrigger mit der „All OR“-Funktion gebildet und über einen AUX-Eingang in die Trigger-LMU geleitet. Dort werden für zwei Ausgänge, einmal für das On-Window und einmal für das Off-Window, die Bedingungen für einen akzeptierten Trigger eingestellt. Das bedeutet, es wird sowohl eine Koinzidenz der Teilchentrigger mit den Gammatriggern und dem On- bzw. Off-Window (UND-Bedingung), als auch die gleichzeitig nötigen Anti-Koinzidenzen mit den Signalen ADC-Busy, Synch und DAQ-Deadtime (UND-Bedingung mit den invertierten Signalen) definiert. Aus den Gamma- und Teilchentriggern lassen sich dabei mit dem Gate-Delay der Trigger-LMU Gates einer bestimmten Länge, gleich der Einstellung im Trigger-System, erzeugen. Damit erhält man einen Ausgang mit den akzeptierten Triggern im On-Window und einen zweiten mit den Triggern im Off-Window. Um die Gesamtanzahl davon zu erhalten, werden die Zählerwerte dieser beiden Ausgänge addiert. Parallel dazu werden die ADC-Gates, die vom Trigger-System erzeugt werden, ebenfalls gezählt. Die Scaler werden anschließend verglichen und bei Übereinstimmung eine Erfolgsbestätigung bzw. bei Abweichungen eine Fehlerbeschreibung mit den jeweilig gezählten Werten ausgegeben. Da durch unterschiedliche Verarbeitung der Signale im Test-System und im Trigger-System die zeitliche Korrelation zwischen den einzelnen Signalen nicht mehr gegeben sein könnte, kann man für die Gate-Delays der Trigger-LMU für jedes Signal eine Verzögerung definieren, um diese anzupassen.³³

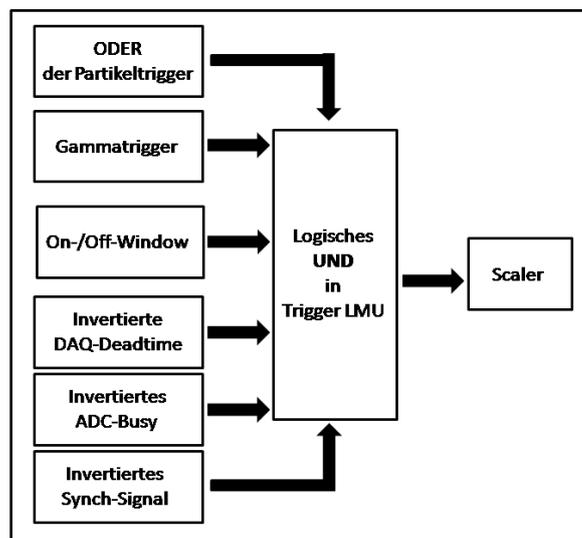
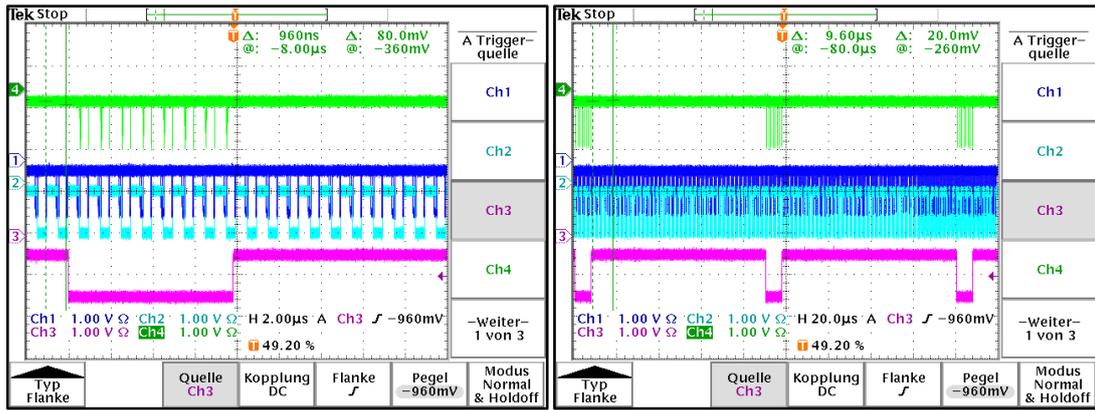


Abbildung 17: Erzeugung eines ADC-Gates im Test-System

³³ (Warr, 2010)



Abbildungen 18 und 19:

Links hohe, rechts niedrige Zeitauflösung.

Beispiel für Koinzidenzen: Das blaue Signal entspricht dem ODER der Teilchentrigger, das türkise den Gammatrigger, das lila Signal ist ein On-/Off-Window und das grüne Signal sind akzeptierte Trigger (also die Koinzidenz drei anderen Signale).

7. Die Konfiguration des Test-Systems

Das Konfigurationsprogramm muss wie in Kapitel 4 nach jeder Änderung von Parametern wie Delay-Länge und Periode der Pulsgeneratoren neu kompiliert werden, bevor es ausgeführt werden kann, um die Änderungen auf das Modul zu übertragen. Es ist häufig auch sehr umständlich, diese Änderungen vorzunehmen. Aus diesem Grund wurde ein OpenSource-Programm³⁴ angepasst und in das Konfigurationsprogramm implementiert. Dazu wurden die modifizierten Dateien „inifile.c“ und „inifile.h“ eingebunden und die dort definierten Funktionen verwendet. Damit lässt sich eine Konfigurationsdatei erzeugen (ini-Datei) und nach ihrer Bearbeitung, sprich nach Änderung von Werten, wieder einlesen. Dadurch ist kein Editieren und somit auch kein erneutes Kompilieren des Test-System-Programms für eine Parameter-Änderung nötig.

Um eine neue Konfigurationsdatei zu erzeugen, führt man das Test-System-Programm folgendermaßen aus:

```
quicktest --create_ini
```

Hierbei ist „**quicktest**“ der Name des Test-System-Programms. Dadurch wird eine ini-Datei namens „miniball.ini“ erzeugt, die ähnlich der im Anhang eingefügten ist (siehe Anhang, Teil D). In dieser Datei lassen sich nun die gewünschten Werte für die jeweiligen Perioden („PeriodeLen“), Delays („Delay“) und Gate-Längen („Stretch“) der simulierten Signale eintragen. Die Bezeichnung des Signals steht dabei in eckigen Klammern und nachfolgend die Parameter-Bezeichnungen, die gleich einem Wert gesetzt werden können. Zu beachten ist dabei, dass diese Werte immer einer Zeitlänge in Clock-Zyklen, also 10ns-Schritten entsprechen. Somit bezeichnet beispielsweise der Wert 1400 eine Länge von $1400 \cdot 10\text{ns} = 14\mu\text{s}$.

```
[Particle1-Trigger]
```

```
Delay=10
```

```
PeriodeLen=500
```

```
Stretch=2
```

Dies würde für den Teilchenpuls 1 ein Delay von 100ns, eine Periode von $5\mu\text{s}$ (also eine Frequenz von 200 kHz) und eine Pulslänge von 20ns definieren. Die Teilchenpulse 2 und 4 haben deswegen keine einstellbare Periode, da sie, wie in Kapitel 5 beschrieben, die gleiche besitzen wie die Teilchenpulse 1 und 3.

Um die in letzten Abschnitt von Kapitel 6 erwähnte Anpassung der Länge und der zeitlichen Korrelation der Signale in der Trigger-LMU vorzunehmen, trägt man – wie schon beschrieben – die gewünschten Werte im jeweiligen Abschnitt ein, der mit dem Signal-Namen und dem Zusatz **Trig_LMU** benannt ist. Die Einstellungen des ODERs der Teilchentriggersignale fügt man also hier ein:

³⁴ Download unter: <http://sourceforge.net/projects/inifile/>,
Beschreibung: <http://inifile.sourceforge.net/>,
LGPL-Lizenz (Free for private and commercial use)

[Particle-OR Trig_LMU]
Delay=0
Stretch=0

Die Zeit für einen Überprüfungsdurchlauf, das heißt wie lange die Scaler beim Analysevorgang zählen sollen bis sie gelatcht werden, lässt sich im Abschnitt **[Counting]** (wiederum in 10ns-Schritten) eingeben.

8. Fazit und Ausblick

In dieser Arbeit wurde der Aufbau eines digitalen Test-Systems für das Miniball-Experiment am CERN, sowie der allgemeine Versuchsaufbau von Coulex-Experimenten an selbigem beschrieben, außerdem eine Dokumentation für die Modifikation und Konfiguration des verwendeten VULOM3-Moduls und speziell des entwickelten Test-Systems erstellt.

Mit diesem Test-System lassen sich die vom Trigger-System zu verarbeitenden Signale des Experiments komplett simulieren, wobei dabei deren Eigenschaften wie Pulslänge und Zeitkorrelation nach Bedarf einstellbar sind. Die vom Trigger-System ausgegebenen Signale können auf ihre Vollzähligkeit und ihre zeitliche und systematisch richtige Erzeugung überprüft und somit das System auf seine fehlerfreie Funktionalität getestet werden. Das System kann durch einfache Modifikation auch für ähnliche Experimente wie, zum Beispiel Transfer-Experimente am Miniball-Aufbau, oder für die On-Line-Kontrolle des Trigger-Systems während eines Experiments angepasst und dabei zur Ausgabe von Zusatzinformationen genutzt werden. Es wurde bis auf einen finalen Test, der nicht durchgeführt werden konnte, da noch kein vollständig funktionsfähiges Trigger-System vorhanden war, nahezu komplett getestet.

Durch eine Erweiterung oder Anpassung der Firmware könnten noch tiefergehende Kontrollen wie die Überprüfung zeitlicher Korrelation von Signalen ermöglicht und eine realistischere Simulation der Signale durch Zufallsgeneratoren erreicht werden.

9. Anhang

A. Verwendete C-Standardbibliotheken

```
<stdlib.h>  
<stdio.h>  
<smem.h>  
<sys/ioctl.h>  
<sys/types.h>  
<signal.h>  
<ctype.h>  
<ces/vmelib.h>  
<string.h>  
<time.h>  
<unistd.h>
```

B. Beispiel-Programm

```
#include <stdlib.h>  
#include <stdio.h>  
#include <smem.h>  
#include <errno.h>  
#include <sys/ioctl.h>  
#include <sys/types.h>  
#include <signal.h>  
#include <ctype.h>  
#include <ces/vmelib.h>  
#include <string.h>  
#include <time.h>  
#include <unistd.h>  
  
#include "vulom3def.h"  
#include "trlo_defs.h"  
  
unsigned int find_controller(unsigned int vmeaddr,  
                           unsigned int vme_len,  
                           unsigned int vme_am,  
                           int x,  
                           int y,  
                           void *param);  
  
void return_controller(unsigned int vme_virt_addr,  
                      unsigned int vme_len);  
  
/* VME Adressen und Offsets */  
  
/* Segmentnamen */
```

```

#define SEG_ADDR 0x10000000
#define SEG_LEN 0x010000
#define MAXLEN 132
#define countof(x) (sizeof(x)/sizeof((x)[0]))
int conv[4];
volatile int *p_vme;
int l_vme_perm;
int l_vme_addr;
char *p_mem;
int l_seg_addr;
struct pdparam_master param;
unsigned long vmeaddr,vme_len,vme_am;
unsigned long vme_virt_addr;

int sighandler(l_signal)
int l_signal;
{
    switch(l_signal)
    {
        case SIGKILL:
            printf("signal SIGKILL received \n");
            break;
        case SIGINT:
            printf("signal SIGINT received \n");
            break;
        case SIGQUIT:
            printf("signal SIGQUIT received \n");
            break;
        case SIGTERM:
            printf("signal SIGTERM received \n");
            break;
        case SIGBUS:
            printf("signal SIGBUS received \n");
            break;
        case SIGSEGV:
            printf("signal SIGSEGV received \n");
            break;
        default:
            printf("signal %d received \n",l_signal);
            break;
    }
    return_controller(vme_virt_addr,vme_len);
    exit(0);
}

void clean_state(volatile trlo_register_map *hw)
{
    int i;

    uint32_t *p = (uint32_t *) &(hw->setup);

    for (i = 0; i < sizeof(hw->setup.mux)/sizeof(uint32_t); i++)

```

```

    *(p++) = TRLO_MUX_SRC_WIRED_ZERO;

    for ( ; i < sizeof(hw->setup)/sizeof(uint32_t); i++)
        *(p++) = 0;
}

void BeispielConfig (volatile trlo_register_map *hw)
{
    clean_state (hw);

    hw->setup.period[0] = 300;
    hw->setup.mux[TRLO_MUX_DEST_ECL_OUT(0)] = TRLO_MUX_SRC_PULSER(0);

    hw->setup.mux[TRLO_MUX_DEST_GATE_DELAY(0)] = TRLO_MUX_SRC_PULSER(0);
    hw->setup.stretch[0] = 50;
    hw->setup.restart_mode[0] = TRLO_RESTART_MODE_LEADING_EDGE;
    hw->setup.mux[TRLO_MUX_DEST_ECL_OUT(1)] = TRLO_MUX_SRC_GATE_DELAY(0);

    hw->setup.mux[TRLO_MUX_DEST_GATE_DELAY(1)] = TRLO_MUX_SRC_GATE_DELAY(0);
    hw->setup.delay[1] = 100;
    hw->setup.restart_mode[1] = TRLO_RESTART_MODE_WHEN_PRESENT;
    hw->setup.mux[TRLO_MUX_DEST_ECL_OUT(2)] = TRLO_MUX_SRC_GATE_DELAY(1);
}

int main (int argc, char *argv[])
{
    int offs = 0x0f000000;
    volatile unsigned int *ptr;

    signal(SIGKILL,(void *)sighandler);
    signal(SIGINT,(void *)sighandler);
    signal(SIGQUIT,(void *)sighandler);
    signal(SIGTERM,(void *)sighandler);
    signal(SIGBUS,(void *)sighandler);
    signal(SIGSEGV,(void *)sighandler);

    if (argc > 1 && strcmp(argv[1],"--addr=",7) == 0)
    {
        offs = strtol(argv[1]+7, NULL, 16);
    }

    param.iack = 1;
    param.rdpref = 0;
    param.wrpost = 0;
    param.swap = SINGLE_AUTO_SWAP;
    vmeaddr = offs ;
    vme_len = 0x1000000;
    vme_am = 0x09;

    vme_virt_addr = find_controller(vmeaddr,vme_len,vme_am,0,0,&param);
    if( vme_virt_addr == -1 )
    {

```

```

printf("Unable to map address 0x%08x\n",(unsigned int) vme_virt_addr);
return(1);
}
ptr = (unsigned int *)vme_virt_addr;
BeispielConfig((trlo_register_map *) ptr);
exit(0);
}

```

C. Funktions- und ECL-Belegungen

Simulation der Signale

Signal	Pulsgenerator Nr.	Gate-Delay Nr.	LMU-Eingang	LMU-Ausgang	ECL-Ausgang	ECL-Eingang
ADC-Busy	---	1	---	---	7	0, 1, 2, 3
DAQ-Deadtime	---	0	---	---	6	4
DGF-Busy/Synch	6	---	6, 7	6	8	12
Gammatrigger	1	---	0	0	4	---
PS-Puls	5	---	---	---	10	---
T1-Puls	4	---	4	4	9	---
Teilchentrigger 1	2	---	1	1	0	---
Teilchentrigger 2	2	2	---	---	1	---
Teilchentrigger 3	3	---	3	3	2	---
Teilchentrigger 4	3	3	---	---	3	---

Analyse der Signale

Koinzidenz	Trigger-LMU Eingang	Trigger-LMU Ausgang	Trigger-LMU AUX-Eingang	All OR
Forced Readout On-/Off-Window	5, 6, 12	12, 13	---	---
ADC-Busy ADC-Gates	---	1	0,3	---
Synch-Signal ADC-Gates	---	2	1, 3	---
ADC-Gate- Erzeugung	5, 6	6, 7	0, 1, 2	0

D. Ini-Datei

```
[ADC-Busy]
Delay=0
PeriodeLen=0
Stretch=0
[EBIS-Pulse]
Delay=0
PeriodeLen=0
Stretch=0
[DAQ-Deadtime]
Delay=0
PeriodeLen=0
Stretch=0
[Gamma-Trigger]
Delay=0
PeriodeLen=0
Stretch=0
[Particle1-Trigger]
Delay=0
PeriodeLen=0
Stretch=
[Particle2-Trigger]
Delay=0
Stretch=0
[Particle3-Trigger]
Delay=0
PeriodeLen=0
Stretch=0
[Particle4-Trigger]
Delay=0
Stretch=0
[PS-Pulse]
Delay=0
PeriodeLen=0
Stretch=0
[Synch]
Delay=0
PeriodeLen=0
Stretch=0
[T1-Pulse]
Delay=0
PeriodeLen=0
Stretch=0
[ADC Trig_LMU]
Delay=0
Stretch=0
[DAQ Trig_LMU]
Delay=0
Stretch=0
```

[Gamma Trig_LMU]
Delay=0
Stretch=0
[On-Off-Window Trig_LMU]
Delay=0
[Particle-OR Trig_LMU]
Delay=0
Stretch=0
[Synch Trig_LMU]
Delay=0
Stretch=0
[Trigger-OR Trig_LMU]
Delay=0
[Counting]
Time=0

E. Literaturverzeichnis

- Henriques, A. I. (2010). A new Trigger Logic system for the LAND/R3B setup. Portugal.
- Hoffmann, J. (13. Mai 2008). *GSI - VULOM3*. Abgerufen am 10. August 2011 von http://www.gsi.de/informationen/wti/ee/elekt_entwicklung/vulom_v3_v4.pdf
- Johansson, H. (kein Datum). *TRLO II - description and definitions*. Abgerufen am 12. August 2011 von http://fy.chalmers.se/~f96hajo/trloii/descr_defs_frame.html
- REX-ISOLDE. (14. Juli 2011). Abgerufen am 19. August 2011 von <http://isolde.web.cern.ch/ISOLDE/REX-ISOLDE/index.html>
- The Isolde facility*. (kein Datum). Abgerufen am 10. August 2011 von <http://isolde.web.cern.ch/ISOLDE/default2.php?index=index/facilityindex.htm&main=facility/facility.php>
- Walle, J. V. (2006). Coulomb excitation of neutron rich Zn isotopes. Niederlande.
- Warr, N. (31. May 2010). Miniball electronics at May 2010.
- Wikipedia - Analog-Digital-Umsetzer*. (3. August 2011). Abgerufen am 18. August 2011 von <http://de.wikipedia.org/wiki/Analog-Digital-Umsetzer>
- Wikipedia - Emittergekoppelte Logik*. (27. Juli 2011). Abgerufen am 8. August 2011 von http://de.wikipedia.org/wiki/Emittergekoppelte_Logik
- Wikipedia - Field Programmable Gate Array*. (15. Juli 2011). Abgerufen am 10. August 2011 von http://de.wikipedia.org/wiki/Field_Programmable_Gate_Array
- Wikipedia - Spallation*. (19. März 2011). Abgerufen am 8. August 2011 von <http://de.wikipedia.org/wiki/Spallation>