Fakultät für Physik und Astronomie
Ruprecht-Karls-Universität Heidelberg

Bachelorarbeit im Studiengang Physik

vorgelegt von

Aleem Ahmad Tariq Sheikh

geboren in Rüdesheim am Rhein (Deutschland)

Hardware-Implementation eines Algorithmus zur Trefferauswahl in einem Triplet Track Trigger

Die Bachelorarbeit wurde von Aleem Ahmad Tariq Sheikh ausgeführt am
Physikalischen Institut der Universität Heidelberg
unter der Betreuung von
Herrn Prof. Dr. André Schöning

Zusammenfassung

Zur genaueren Untersuchung von seltenen Kollisionsprozessen sind Teilchenbeschleuniger in Planung, die besonders hohe Luminositäten erreichen. Da der Pile-up proportional zur Luminosität ansteigt, gibt es eine Notwendigkeit für Triggersysteme, die selbst bei hohem Pile-up effizient arbeiten. Der Triplet Track Trigger (TTT) ist ein Trigger, der den Speicher- und Bandbreitenlimitationen der Experimente gerecht wird, ohne die Triggerschwelle heraufzusetzen [2]. Der TTT nutzt einen Algorithmus zur Spurrekonstruktion, der in Hardware umgesetzt werden kann. Aufgrund der hohen Datenrate ist eine geringe Latenz für die Spurrekonstruktion nötig, weshalb die Spurrekonstruktion in Hardware implementiert werden muss. Ziel dieser Arbeit ist es diesen Algorithmus in einem FPGA auf seine Leistung zu testen.

Abstract

For a more detailed investigation of rare collision processes, particle accelerators are being planned that reach particularly high luminosities. Since the pile-up increases proportionally to the luminosity, there is a need for trigger systems that work efficiently even at high pile-ups. The Triplet Track Trigger (TTT) is a trigger that meets the memory and bandwidth limitations of the experiments without raising the trigger threshold [2]. The TTT uses a track reconstruction algorithm that can be implemented in hardware. Due to the high data rate, low latency is required for track reconstruction, which is why track reconstruction must be implemented in hardware. The aim of this work is to test the performance of this algorithm in an FPGA.

Inhaltsverzeichnis

1 Theorie **3**
1.1 Koordinatensystem und Detektorgeometrie 3
1.2 Spurrekonstruktion 3
1.2.1 Trefferauswahl 4
1.2.2 Berechnung der Spurparameter 4
1.3 FPGA 5
1.3.1 Vor- und Nachteile gegenüber ASICs 5
1.4 Design-Elemente 6
1.4.1 1 aus n Multiplexer 6
1.4.2 FIFO-Puffer 6
1.4.3 Schieberegister 8

2 Implementation der Treffersuche **9**
2.1 Überblick 10
2.2 Erster Vergleich 10
2.3 Zweiter Vergleich 11
2.4 Zwischenpuffer 12
2.5 Auslesen der Zwischenpuffer 13
2.5.1 Verbesserung durch *gnt_selector* Modul 13

3 Ergebnisse und Zusammenfassung **14**

1 Theorie

1.1 Koordinatensystem und Detektorgeometrie

Der Detektor besteht aus drei konzentrisch angeordneten Zylindern. Jede Lage detektiert Teilchen, die diese Lage passieren. In der Mitte der Zylinder befinden sich die entgegenlaufenden Teilchenstrahlen.

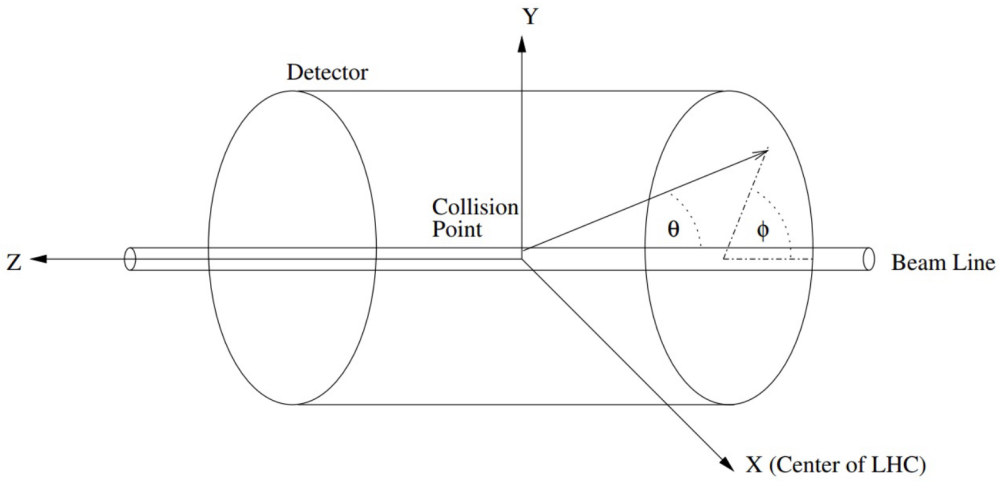

Abbildung 1: Koordinatensystem im Teilchendetektor [3]

1.2 Spurrekonstruktion

In einem Elektromagnetischen Feld bewegen sich geladene Teilchen auf einer Helixbahn. Die Lorentzkraft auf ein Teilchen der Ladung q, das sich mit der Geschwindigkeit $\vec{v}$ in einem homogenen Magnetfeld mit der Magnetischen Flussdichte $\vec{B} = B\vec{e_z}$ bewegt, ist gegeben durch:

$$\frac{d\vec{p}}{dt} = q(\vec{v} \times \vec{B}) \tag{1}$$

Diese Bewegungsgleichung mit den Anfangsbedingungen $\vec{x}(t=0) = 0$, $\vec{v}(t=0) = v_T\vec{e_x} + v_z\vec{e_z}$ wird gelöst durch:

$$\vec{x}(t) = (Rsin(\omega t))\vec{e_x} + (Rcos(\omega t))\vec{e_y} + (v_Z t + z_0)\vec{e_z} \tag{2}$$

mit $\omega = \frac{qB}{m}$ und $R = \frac{v_T}{\omega}$ dem Radius der Helixbahn.

1.2.1 Trefferauswahl

Seien $\vec{x_1}, \vec{x_2}, \vec{x_3}$ die Ortsvektoren der Treffer in den Detektorlagen l1,l2,l3. Um im Detektor Bahnen zu rekonstruieren, werden aus den Koordinaten der Treffer der verschiedenen Detektorlagen Kombinationen gebildet. Diese Kombinationen werden nach folgenden Schritten ausgewählt:

1. Wähle einen Treffer aus der äußersten Detektorebene l3 aus

2. Suche in einem Winkel von $\Delta\Phi_{13}$ nach Treffern in der ersten Detektorebene l1, die maximal Δz_{13} in z-Richtung voneinander entfernt sind.

3. Suche für jede Kombination von Treffern aus l1 und l3, die nach 2. zusammengehören, nach Treffern in der zweiten Detektorebene l2, die im Suchfenster liegen, welches wie folgt definiert ist:
   Da die Abstände der Lagen sehr klein gegenüber dem Radius des Detektors sind ($R_{TTT} = 857mm$, Abstand der Lagen $d = 30mm$), sind die Teilchenbahnen nahezu geradlinig. Daher wird ein Teilchen am Schnittpunkt der Geraden, die durch die Punkte $\vec{x_1}$ und $\vec{x_3}$ definiert ist, und der Detektorlage l2 erwartet. Um diesen Schnittpunkt wird ein Suchfenster eingerichtet, dass von dem Treffer in $l3$ ausgehend einen Winkel von $\Delta\Phi_2$ umfasst und z-Abstände bis zu Δz_2 akzeptiert.

1.2.2 Berechnung der Spurparameter

Nachdem Kandidaten für die Spurrekonstruktion gefunden wurden, die den Kriterien der obengenannten Suchfenster genügen, können aus den Koordinaten der Treffer folgende Spurparameter der Helixbahn berechnet werden.[2, S.109]

1. Die Transversale Impulskomponente $p_T = \sqrt{p_x^2 + p_y^2}$

2. Die Pseudorapidität $\eta = -ln(tan(\theta/2))$, Wobei θ der ursprüngliche Winkel zwischen der z-Achse und der Teilchenspur ist.

3. Die initiale Bewegungsgrichtung $\varphi = arctan(y/x)$.

4. Der Ursprung des Teilchens z_0

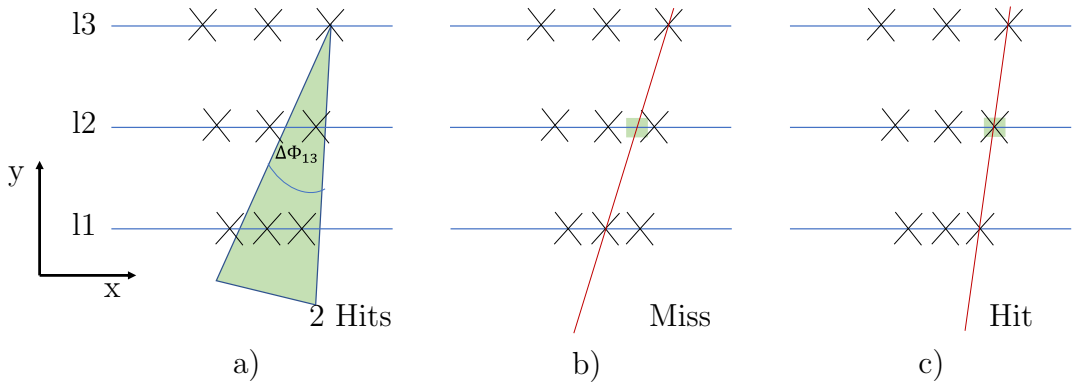

Abbildung 2: Auswahl der Treffer-Tripel mit Hilfe von Suchfenstern für ein festes z
a) Auswahl eines Paares von Treffern zwischen l1 und l3. Suche ausgehend von l3 b) Verwerfen eines Paares aufgrund von mangelnden Kandidaten in l2 c) Bestätigung eines Paares von Treffern

5. Den kleinsten Abstand *dca*, den die Bahn zur z-Achse hat.

Im letzten Schritt werden diese Spurparameter noch finalen Auswahlkriterien ausgesetzt. Aus der Detektorgeometrie können der Ursprung z_0, die Pseudorapidität η und der transversale Impuls nur bestimmte sinnvolle Werte annehmen.

1.3 FPGA

Ein FPGA (Field Programmable Gate Array) ist ein integrierter Schaltkreis, in dem logische Schaltungen simuliert werden können. Der FPGA muss vorher programmiert werden und kann dann in den Grenzen seiner Größe beliebige Logikschaltungen implementieren. Da FPGAs im Gegensatz zu ROMs (Read Only Memory) wieder programmiert werden können, eignen sich diese perfekt um Hardware zu testen, bevor diese in Halbleiterfabriken teuer hergestellt wird.

1.3.1 Vor- und Nachteile gegenüber ASICs

FPGAs brauchen im Gegensatz zu ASICs (Application Specific Integrated Circuit) nur geringe Entwicklungskosten und sind daher schneller im Einsatz. Außerdem sind sie sehr viel flexibler, da eine Reprogrammierung möglich

ist. Dadurch können jederzeit Erweiterungen und Korrekturen vorgenommen werden.
Durch die hohe Strahlenbelastung an zukünftigen Teilchenbeschleunigern eignen sich FPGAs weniger gut im Einsatz vor Ort, da diese empfindlich auf Teilchenstrahlung und elektromagnetische Wellen reagieren. Den Preis, der für die Flexibilität eines FPGAs gezahlt wird, besteht aus einer geringeren möglichen Taktrate des Hardwareentwurfs, einer geringeren Logikdichte und damit einhergehend einem größeren Stromverbrauch für dieselbe Hardware-Funktionalität. Aus diesen Gründen sind ASICs für größere Stückzahlen zu bevorzugen.
Aufgrund der Strahlungsempfindlichkeit und der Größe des kombinatorischen Aufwandes der Treffersuche ist eine Implementierung auf ASICs unerlässlich.

1.4 Design-Elemente

Wie zu einem Werkzeugkoffer eines Angewandten Mathematikers Analysis, Algebra und Aussagenlogik gehören, so gibt es auch für den Design von Hardware einige Werkzeuge, mit denen ein Entwickler vertraut ist. Durch das Entkoppeln von Funktionalitäten in kleinere Module werden Fehlerquellen minimiert und leichter identifiziert. Der Aufwand ein komplexes Hardware-Design zu verstehen reduziert sich immens, da meist nur noch die einzelnen Schnittstellen überprüft werden müssen.

1.4.1 1 aus n Multiplexer

Der 1 aus n Multiplexer ($MUX(1, n)$) wählt aus n eingehenden Signalen ein Ausgangssignal aus und leitet dieses an seinen Ausgang weiter. Er wird von einem Eingangssignal *sel* gesteuert, das mehrere Bits umfassen kann[1, S.111-114]. Die einfachste Variante ist ein Multiplexer, der aus zwei Signalen Eines Auswählt. In diesem Fall umfasst das Steuersignal 1 Bit. Multiplexer können zum Einsatz kommen, wenn aus mehreren Zwischenspeichern, der Reihe nach Daten über einen Bus ausgelesen werden sollen. Der Multiplexer entscheidet dann, welcher Zwischenspeicher ausgelesen wird.

1.4.2 FIFO-Puffer

Ein First-In-First-Out-Puffer (FIFO-Puffer) ist ein getakteter Zwischenspeicher, der eine bestimmte Anzahl $DEPTH$ von Wörtern mit Wortbreite

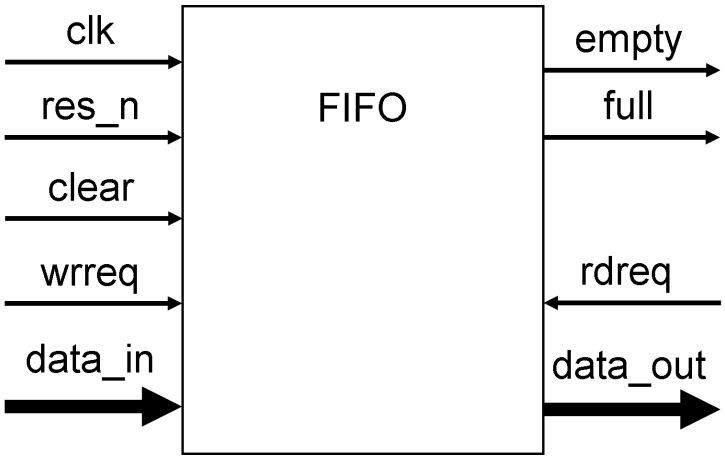

Abbildung 3: Schaltsymbol für einen FIFO-Puffer

$WIDTH$ fassen kann. Hierbei wird das Prinzip einer Warteschlange verwendet. Das zuerst geschriebene Wort wird als erstes wieder ausgelesen. Die Lese- und Schreiboperationen geschehen mit Steuersignalen. Hier wird von einem FIFO in einer Taktdomäne gesprochen. Es gibt jedoch auch FIFOs die zur Synchronisation zwischen verschiedenen Taktdomänen eingesetzt werden. Diese besitzen dann zwei Takte wr_clk und rd_clk.[1, S.174-177]
Per Anfrage mit dem Bit *wrreq* wird zur positiven Taktflanke das anliegende Wort am Anschluss *data_in* in den Zwischenspeicher geschrieben. Analog dazu wird mit dem Bit *rdreq* bis zur positiven Taktflanke des nächsten Taktzyklus das Wort am Ausgang *data_out* ausgegeben. Das asynchrone Reset-Signal res_n, setzt das FIFO zurück. Dies bedeutet, dass das *empty* Bit auf 1 gesetzt wird und das $full$ Bit auf 0. Die Speicherinhalte des FIFOs werden nun als „not valid“behandelt, sodass bei richtiger Verwendung nicht aus einem leeren FIFO gelesen wird. Im Gegensatz zum synchronen Reset-Signal *clear*, das sich in der Funktionalität nur dadurch unterscheidet, dass es zu einer positiven Taktflanke das FIFO leert, ist das asynchrone Reset-Signal invertiert. Dies bedeutet, dass das FIFO solange im Reset-Zustand ist wie das Signal $res\_n = 0$ ist. Der Grund hierfür liegt darin, dass zum Start von einem Chip keine stabile Spannungsversorgung garantiert werden kann. Deshalb kann ein halten auf $V = V\_cc$ nicht garantiert werden, jedoch ist ein stabiles Halten eines Signals auf $V = 0$ problemlos möglich durch einen Anschluss an Masse.

1.4.3 Schieberegister

Schieberegister simulieren das Prinzip einer Eimerkette. Sie bestehen aus einer Reihe von Registern, die aneinander gekettet sind. Zu jedem Taktzyklus wird das Wort in Register i an das Register $i + 1$ übergeben. Im Gegensatz zu einem FIFO muss jedes Wort, das in das erste Register geschrieben wird, alle Register durchlaufen. Außerdem ist das Auslesen von Daten innerhalb der Kette möglich. Deshalb werden Schieberegister zur Parallelisierung von Datenströmen eingesetzt. Eine weitere Verwendung kann das Verteilen von einem Wort auf mehrere folgende Module sein.[1, S.131-135]

2 Implementation der Treffersuche

Die Treffersuche aus 1.2.1 wird unter folgenden Annahmen in einem Chip *TTT_pre* mit dem Interface (siehe Abbildung 4) implementiert:

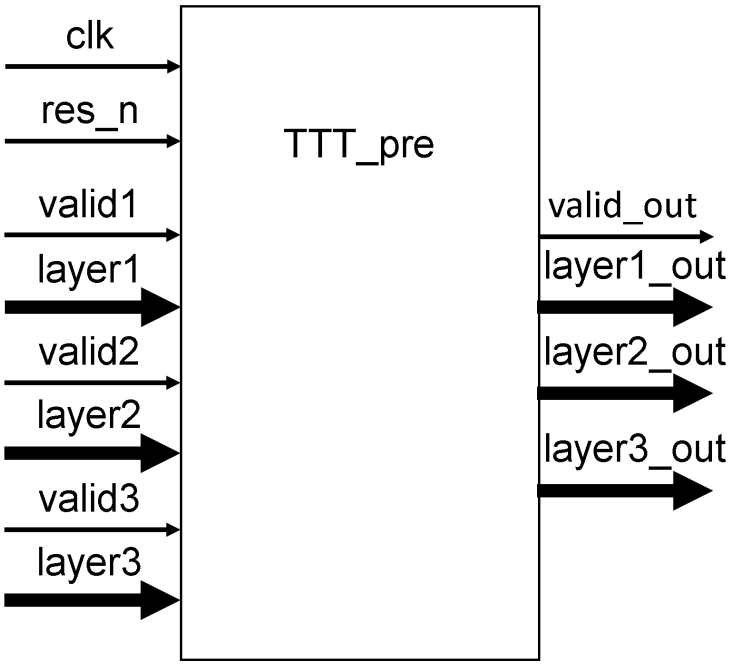

Abbildung 4: Interface für die Trefferauswahl

1. Zu jedem Taktzyklus erhält der Chip die Koordinaten eines Treffers. Die ersten 13 Bit stellen die x-Koordinate auf dem quadratischen Pixelarray des Detektors dar. Die letzten 13 Bit stellen die y-Koordinate auf dem quadratischen Pixelarray des Detektors dar.

2. Es sind maximal 20 Treffer pro Lage in einem Ereignis möglich [1], was zu einem kombinatorischen Aufwand von $20*20*20 = 8000$ Vergleichen führt.

[1] Mit Treffer sind Cluster von Treffern gemeint. Der Algorithmus findet erst nach dem Clustern von Treffern statt, da es nicht sinnvoll ist, die Spuren einzelner Teilchen zu rekonstruieren

3. Es ist keine Rückflusskontrolle möglich. Das heißt, dass das nachfolgende Modul die Daten in jedem Taktzyklus aufnehmen muss, in dem valide Daten ausgegeben werden, da ansonsten nach 1.2.1 valide Kandidaten zur Spurrekonstruktion verloren gehen würden.

2.1 Überblick

Der Chip besteht im Kern aus zwei Schieberegisterketten, die gegenläufig ausgelesen werden und für den ersten Vergleich die Trefferkoordinaten aus Detektorebene 1 und Detektorebene 3 an die $FirstCut$ Module weitergeben. Die $FirstCut$ Module entscheiden durch den eigentlichen Vergleich, ob die Treffer ein valides Paar bilden. Die $SecondCut$ Module erhalten ein Paar $(l1, l3)$ von Treffern und die Trefferkoordinaten aus Detektorebene 2. Hier wird erneut entschieden, ob diese eine valide Kombination bilden. Valide Kombinationen werden zwischengespeichert und anschließend werden alle Puffer nacheinander durch das Mux Modul ausgelesen und ausgegeben.

2.2 Erster Vergleich

Zu aller erst werden die Koordinaten in die Schieberegister eingeladen. Sobald beide Schieberegister zur Hälfte gefüllt sind beginnen die $FirstCut$ Module zu arbeiten, da valide Daten anliegen. Diese erzeugen innerhalb von drei Taktzyklen das dazugehörige valid Bit. Da der Vergleich nicht in einem Taktzyklus geschieht, muss an dieser Stelle eine Pipeline eingebaut werden, die dafür sorgt, dass das $FirstCut$ Modul zu jedem Taktzyklus ein neues Paar aufnehmen kann.
Ein erstes Problem ergibt sich, wenn man sorgfältig zählt, ob alle $5 * 5 = 25$ Kombinationen an Koordinaten auch wirklich an die $FirstCut$ Module übergehen. Da die gegenläufigen Schieberegister mit einer effektiven Geschwindigkeit von 2 Registern pro Taktzyklus schieben, wird jede zweite Kombination verpasst. Dieses Problem wird behoben, indem eine weitere Reihe an $FirstCut$ Modulen angeschlossen wird, in der der Eingang für die Koordinaten von Detektorebene 1 um einen Taktzyklus versetzt ist (siehe Abbildung 6). Dadurch wird sichergestellt, dass keine Kombination doppelt vorkommt und alle Kombinationen erfasst werden.

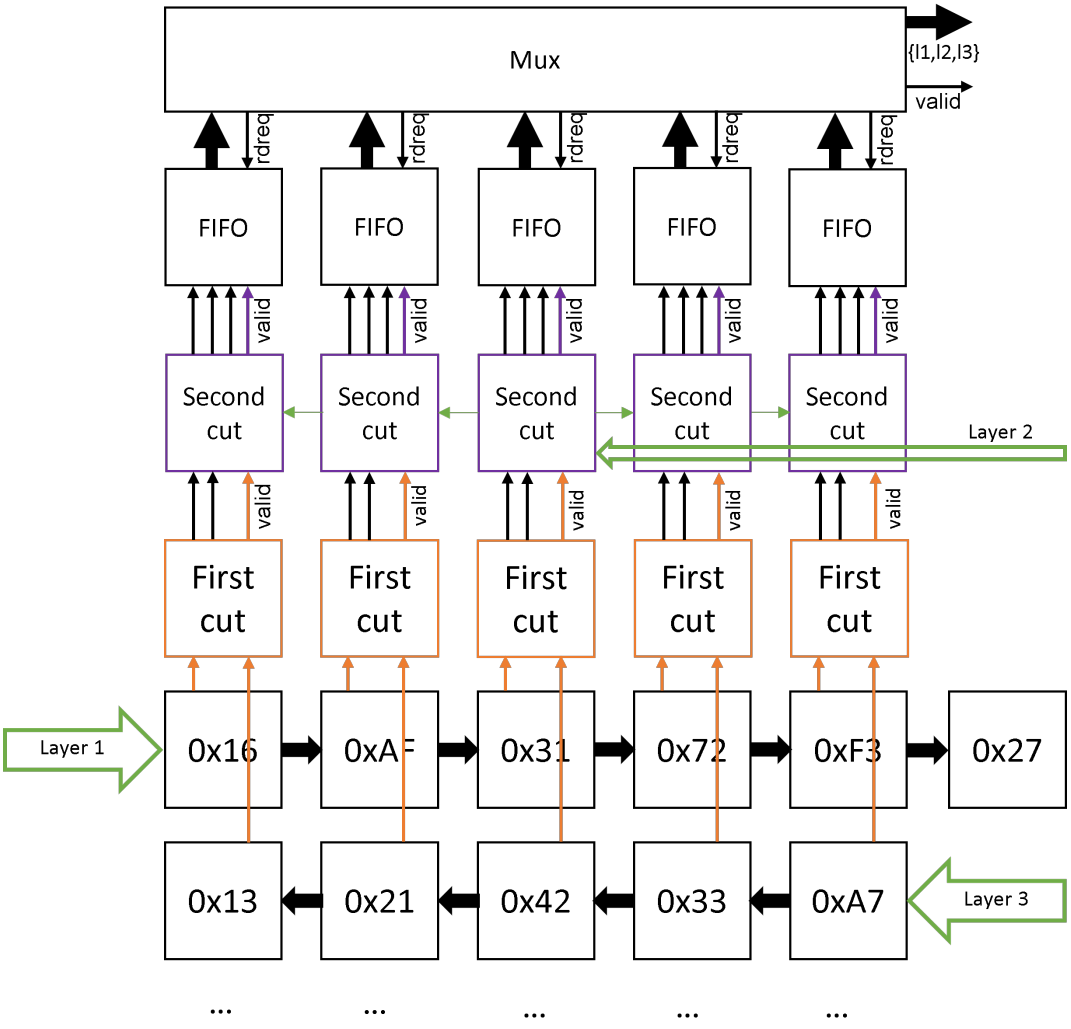

Abbildung 5: Architektur des Chips (vereinfacht)
Es sind nur 5 von eigentlich 40 *FirstCut* Modulen dargestellt

2.3 Zweiter Vergleich

Ähnlich wie beim ersten Vergleich wird beim zweiten Vergleich mit Schieberegistern vorgegangen, um alle Kombinationen zu erhalten. Hierbei muss darauf geachtet werden, dass die Koordinaten für Detektorebene 2 verzögert werden müssen, bis die ersten Paare aus den *FirstCut* Modulen ankommen.

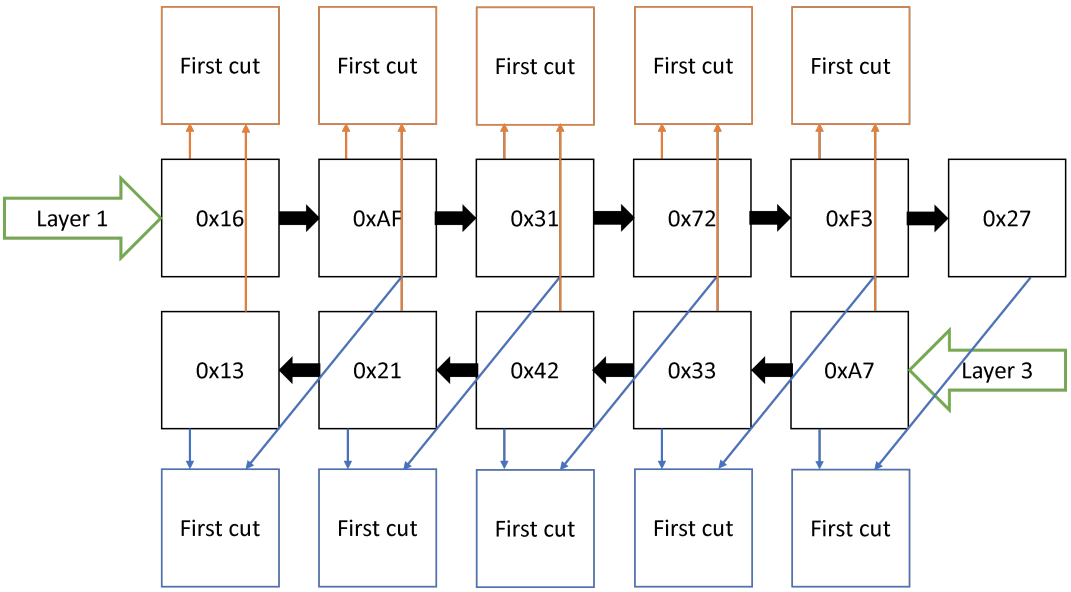

Abbildung 6: *FirstCut* Module in Orange und Blau (um einen Taktzyklus versetzt)
Zur Übersichtlichkeit wurden die Schieberegister für die valid Bits weggelassen

Diese Verzögerung beträgt für das mittlere Modul $10 + 3 = 13$ Taktzyklen und erhöht sich nach außen gehend jeweils um einen weiteren Taktzyklus.

2.4 Zwischenpuffer

Da die Datenrate im Schnitt nach dem ersten vergleich um das zwanzigfache ansteigt, muss diese nach dem zweiten Vergleich wieder um das zwanzigfache abfallen, da zu jedem Treffer pro Lage genau ein Teilchen gehört (siehe Abbildung 7). Da jedoch nicht garantiert werden kann, dass immer nur ein *SecondCut* Modul von 40 pro Taktzyklus eine valide Kombination von Treffern findet, müssen diese gepuffert und anschließend nacheinander aus den Puffern ausgelesen werden. Durch obige Überlegung kann bei einem hinreichend großen Puffer mit sehr großer Wahrscheinlichkeit, davon ausgegangen werden, dass diese Puffer nicht voll werden. Dies würde ein großes Problem sein, da keinerlei Rückflusskontrolle vorhanden ist, um die Vorraussetzungen an die geringe Latenz der Spurrekonstruktion zu erfüllen.

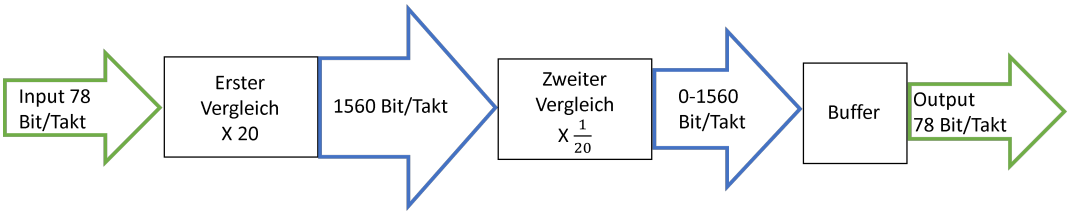

Abbildung 7: Datenflussdiagramm durch den Chip

2.5 Auslesen der Zwischenpuffer

Das Auslesen der Zwischenpuffer wird durch das *Mux* Modul vorgenommen. Dieses erhält die *data_out* Ausgänge der einzelnen FIFOs und die *empty* Bits. Nun wird nach folgendem Algorithmus ausgelesen:

1. Setze $gnt = 0$
2. Wenn FIFO nummer *gnt* nicht leer ist, so sende eine Leseanfrage
3. Wenn *gnt* größer als die Anzahl an FIFOs: setze *gnt* auf 0, ansonsten: Erhöhe *gnt* um 1

Dieser Algorithmus hat den Vorteil, dass er sehr leicht zu implementieren ist und auch keinerlei Logik benötigt, um zu entscheiden, welcher Zwischenpuffer als nächstes ausgelesen wird. Jedoch kann es oft dazu kommen, dass kein Zwischenpuffer ausgelesen wird, obwohl es einen gibt, der nicht leer ist. Zur verbesserung könnte ein weiteres Modul *gnt_selector* implementiert werden:

2.5.1 Verbesserung durch *gnt_selector* Modul

Dieses Modul erhält den aktuellen *gnt* und die *empty* Bits der FIFOs, die one-hot codiert sind. Anschließend wird aus den Eingaben berechnet, welches FIFO im Rundlauf als nächstes nicht leer ist. Dadurch wird das unnötige Überprüfen von leeren FIFOs verhindert und somit die gesamtllatenz des Algorithmus verringert. Außerdem verringert sich die Gefahr, dass einzelne Zwischenpuffer voll laufen.

3 Ergebnisse und Zusammenfassung

Nachdem das Design mit Quartus Prime Version 20.1.1 Build 720 11/11/2020 SJ Standard Edition für den FPGA Arria 10: 10AX115N3F45l2SG getestet wurde ergab sich, dass der Chip auf diesem FPGA eine Frequenz von bis zu 150 MHz erreichen würde. Die Logik-Ressourcen des FPGA wurden mit dem Design zu circa 80% ausgelastet. Erhöht man die Anforderungen an den Chip und möchte mehr als 20 maximale Treffer pro Event zulassen, so erhöhen sich die Logik-Ressourcen quadratisch. Außerdem wird durch das erhöhen der maximalen gleichzeitigen Treffer pro Event die Latenz linear ansteigen, da die Schieberegister sich linear vergrößern. Sei n die maximale Anzahl an gleichzeitigen Treffern pro Event, dann ist die mittlere Latenz t_{pre} in Taktzyklen gegeben durch:

$$t_{pre} = t_{fc} + t_{sc} + n/2 + t_{FIFO} \tag{3}$$

wobei $t_fc = 3$, $t_sc = 6$ und t_{FIFO} bezeichnet die Zeit, die im Mittel zum Auslesen der FIFOs benötigt wird. Im Worst-Case gilt für $t_{FIFO} <= 2 * n^2$. Verwendet man den besseren Algorithmus zur Auslese der FIFOs, kann dies auf $t_{FIFO} <= 2 * n$ verbessert werden.
Die Latenz für diese Implementation liegt bei einer Taktrate von 150 MHz in einem Bereich von

$$133ns <= t_{pre} < 527ns \tag{4}$$

Durch diese Implementation der Treffersuche, die den logisch aufwendigsten Teil des TTT darstellt, konnte gezeigt werden, dass eine Implementation selbst auf einem FPGA mit einer Latenz von weniger als einer Mikrosekunde möglich ist. Durch das Verbessern der Auslese und die Implementation auf ASICs könnte die Taktfrequenz und Latenz weiter verringert werden.

Literatur

[1] Klaus Fricke. *Digitaltechnik*. Springer Fachmedien Wiesbaden, 2021. DOI: 10.1007/978-3-658-32537-4. URL: https://doi.org/10.1007/978-3-658-32537-4.

[2] Tamasi Rameshchandra Kar. „A Triplet Track Trigger for Future High Rate Collider Experiments“. In: (2020). DOI: 10.11588/HEIDOK.00029043. URL: http://archiv.ub.uni-heidelberg.de/volltextserver/id/eprint/29043.

[3] Matthias Schott und Monica Dunford. „Review of single vector boson production in pp collisions at $$\sqrt{s} = 7$$ TeV“. In: *The European Physical Journal C* 74.7 (Juli 2014), S. 2916. ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-014-2916-1. URL: https://doi.org/10.1140/epjc/s10052-014-2916-1.

Erklärung

Hiermit bestätige ich, dass die vorliegende Arbeit nur mit Hilfe der angegeben Hilfsmittel in Eigenarbeit angefertigt wurde.

Aleem Sheikh

Heidelberg, den 14. Februar 2022