Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Fabian Borstel

Konzept und Entwicklung eines Hochgeschwindigkeits-Auslesesystems für einen Röntgenstrahlendetektor auf einem SoC

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Fabian Borstel

Konzept und Entwicklung eines Hochgeschwindigkeits-Auslesesystems für einen Röntgenstrahlendetektor auf einem SoC

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Lutz Leutelt
Zweitgutachter : Prof. Dr.-Ing. Jürgen Vollmer

Abgegeben am 12. März 2019

Fabian Borstel

Thema der Bachelorthesis

Konzept und Entwicklung eines Hochgeschwindigkeits-Auslesesystems für einen Röntgenstrahlendetektor auf einem SoC

Stichworte

Strahlendetektor, hybrider Pixeldetektor, Auslesesystem, Zynq Ultrascale+, FPGA, Firmwareentwicklung, Timepix3, SoC, Datenerfassung

Kurzzusammenfassung

Diese Bachelorthesis beschäftigt sich mit dem Konzept und der Entwicklung eines Auslesesystems für einen Timepix3 basierenden Strahlendetektor am DESY. Die entwickelte Firmware auf dem SoC sowie die Software auf dem Datenerfassungssystem ermöglichen es, mit Timepix3 zu kommunizieren. Durch Aufnahmen und Konfigurationen mit Timepix3 wird ein Grundstein für die Entwicklung eines Timepix4 basierenden Auslesesystems gelegt.

Fabian Borstel

Title of the paper

Concept and development of a high-speed readout system for an X-ray detector using a SoC

Keywords

X-ray detector, hybrid pixel detector, readout system, Zynq Ultrascale+, FPGA, firmware development, Timepix3, SoC, data acquisition (DAQ)

Abstract

This bachelor thesis presents the concept and development of a readout system for a Timepix3 based X-ray detector at DESY. Firmware for a SoC and software for a data acquisition system were developed to allow communication with the Timepix3 chip. This made it possible to tune the configuration of the chip and acquire frames of data from it, thus laying the foundation for the future development of a Timepix4 readout system.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

Diese Bachelorarbeit beschäftigt sich mit dem Konzept und der Entwicklung eines Auslesesystems für einen Röntgenstrahlendetektor am Deutschen Elektronen Synchrotron (DESY). Das DESY ist ein Forschungszentrum der Helmholtz-Gemeinschaft mit Sitz in Hamburg und Zeuthen. Jährlich nutzen über 3000 Gastforscher die Möglichkeiten am DESY, um Forschungsprojekte im Bereich der Grundlagenforschung durchzuführen [6]. Grundlagenforschung beschäftigt sich mit der Untersuchung von Prozessen und Strukturen, welche als Grundlage für naturwissenschaftliche und medizinische Experimente gelten [7]. Dabei ist es eine wichtige Aufgabe von DESY Teilchenbeschleuniger, die diese Forschung ermöglichen, zu betreiben und zu entwickeln. Teilchenbeschleuniger sind meist in Tunnel verbaute Strukturen zur Beschleunigung von elektrisch geladenen Teilchen und Photonen auf nahezu Lichtgeschwindigkeit. Somit kann durch Teilchenbeschleuniger sehr helles Röntgenlicht auf Materialien geschossen werden, um Informationen über den inneren strukturellen Aufbau dieser zu gewinnen. Zusätzlich kann die energiereiche Strahlung in der Medizin dafür genutzt werden, Krebszellen zu zerstören und so Erkenntnisse über den Aufbau der Zellen zu gewinnen [8]. Um den Ausgang der Experimente für den Menschen messbar zu machen, werden Strahlendetektoren verwendet. Strahlendetektoren erfassen die Strahlung und formen diese in elektronische Daten um. Anschließend können aus den Daten Informationen gewonnen werden, die Rückschlüsse auf das durchgeführte Experiment ermöglichen.

Wie in Abbildung 1.1 zu sehen ist, setzt sich ein Strahlendetektor aus einem hybriden Pixeldetektor und einem Auslesesystem zusammen. Der hybride Pixeldetektor wandelt Strah-

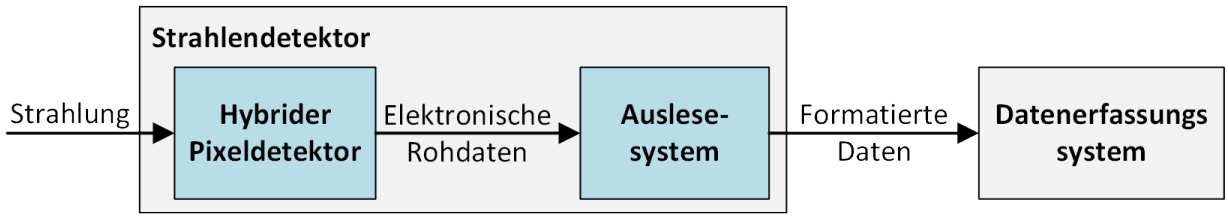

Abbildung 1.1.: Aufbau von Strahlendetektoren

lung in elektronische Rohdaten um und sendet diese an das Auslesesystem. Das Auslesesystem befindet sich auf einer Leiterplatte, die mit Hilfe von integrierten Schaltkreisen und Mikroprozessoren mit dem hybriden Pixeldetektor kommuniziert. Es formatiert die Rohdaten und sendet diese in einem geeigneten Format an das Datenerfassungssystem. Abhängig von der Datenrate ist das Datenerfassungssystem ein handelsüblicher Bürorechner oder ein Hochleistungsrechner. Im Datenerfassungssystem werden die Daten abgespeichert und analysiert.

Die Photon-Science Detector-Group (FS-DS) beschäftigt sich am DESY mit der Entwicklung von Strahlendetektoren. Darunter befindet sich auch der Timepix3 basierende Strahlendetektor dieser Arbeit. Timepix3 ist ein hybrider Pixeldetektor, der vom CERN entwickelt wurde, um die Anzahl, die Ankunftszeit und die Energie von ionisierender Strahlung zu detektieren. Zu ionisierender Strahlung gehören elektrisch geladene Teilchen und elektromagnetische Strahlung [9]. Timepix3 verwendet hochmoderne Ausgabemodi, die den Informationsgehalt der ausgegebenen Daten erhöhen. Somit ist Timepix3 in der Lage, bis zu 1300 Bilder pro Sekunde aufzunehmen und mit einer Datenrate von 5 GBit/s auszugeben [10]. Gerade durch die Eigenschaft, die Ankunftszeit und die Energie eines Teilchens hochauflösend darzustellen, sind Timepix-Systeme bei Nutzern der Strahlendetektoren immer mehr gefragt. Aus diesem Grund beginnt die Gruppe FS-DS im Rahmen dieser Arbeit mit der Entwicklung von Timepix basierenden Auslesesystemen. Dabei liegt der Fokus auf den Mitte 2019 erscheinenden hybriden Pixeldetektor Timepix4. Folglich ist das Ziel dieser Arbeit, mit der Entwicklung eines Timepix3 basierenden Strahlendetekors einen Grundstein für weitere Timepix-Systeme zu legen. Dafür stellt die Gruppe FS-DS ein Prototyp der Hardware des Auslesesystemes zur Verfügung. Demnach beschäftigt sich die Arbeit primär mit der Firmwareentwicklung des Strahlendetektors sowie der Softwareentwicklung des Datenerfassungssystems. Dennoch muss geprüft werden, ob die Hardware für ein Timepix-System geeignet ist. Infolgedessen sollen die Anforderung an ein Timepix-System analysiert werden, indem die grundlegende Kommunikation und Funktion von Timepix3 erprobt wird. Dafür ist es notwendig, mit Timepix3 zu kommunizieren und aus den Rohdaten Informationen zu gewinnen, die eine korrekte Funktion von Timepix3 bestätigen. Für die Kommunikation ist auf dem Auslesesystem ein Zynq Ultrascale+ implementiert. Der Zynq Ultrascale+ setzt sich aus einem Field Programmable Gate Array (FPGA) und einem Mikroprozessor zusammen, welche die Schnittstelle zwischen Timepix3 und Datenerfassungssystem bilden [11].

Nach der Einführung in diesem Kapitel beschäftigt sich das nächste Kapitel mit den Grundlagen eines Timepix3 basierenden Auslesesystemes. Außerdem wird die bereits entwickelte Hardware vorgestellt. Im dritten Kapitel werden die Anforderung an das System analysiert. Anschließend wird im vierten Kapitel ein Konzept ausgearbeitet, welches es ermöglicht, mit Timepix3 zu kommunizieren. Dabei werden verschiedene Module konzipiert, die abhängig von der Aufgabe in verschiedenen Teilen des Systemes realisiert werden. Im fünften Kapitel werden die einzelnen Module implementiert. Bereits hier werden Teile der Module durch Simulationen und erste Kommunikationen mit Timepix3 verifiziert. Danach wird im sechsten Kapitel ein Funktionstest mit dem System durchgeführt. Zusätzlich werden Systemoptimierungen für weitere Timepix-Systeme vorgestellt. Im letzten Kapitel werden die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick für die weitere Entwicklung von Timepix-Systemen gegeben.

2. Grundlagen

Dieses Kapitel erläutert die relevanten Grundlagen für die Entwicklung eines Timepix3 basierenden Strahlendetektors. Dafür wird zuerst die bestehende Hardware im Abschnitt 2.1 vorgestellt. Sie setzt sich unter anderem aus einem Timepix3 und einem Zynq Ultrascale+ zusammen. Demzufolge wird im Kapitel 2.2 der hybrider Pixeldetektor Timepix3 beschrieben. Anschließend werden im Kapitel 2.3 die signifikanten Merkmale von einem Zynq erklärt.

2.1. Bestehende Hardware

Die für diese Arbeit bestehende Hardware ist in Abbildung 2.1 dargestellt. Der Strahlende-

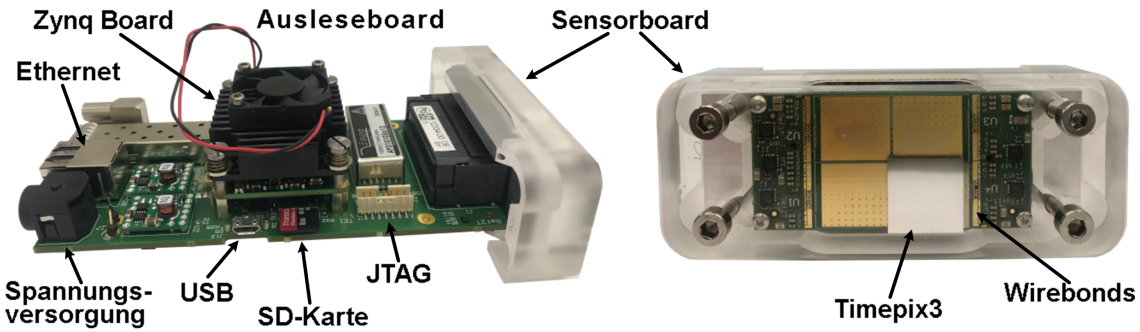

Abbildung 2.1.: Timepix3 basierender Strahlendetektor

tektor setzt sich aus einem Sensorboard und einem Ausleseboard zusammen. Auf dem Sensorboard ist einer von vier möglichen Timepix3-Chips geklebt und über Wirebonds elektrisch mit dem Board verbunden. Timepix3 ist ein hybrider Pixeldetektor, der im nachfolgenden Abschnitt genauer erklärt wird. Der hybride Pixeldetektor kommuniziert mit einem Zynq Ultrascale+ vom Typ ZU3CG, welcher sich auf dem Zynq Board des Ausleseboards befindet. Ein Zynq ist ein programmierbarer integrierter Schaltkreis, der im Abschnitt 2.3 näher erläutert wird. Die Aufgabe des System On Chip (SoC) ist es, Timepix3 zu konfigurieren, Daten von Timepix3 zu erfassen sowie diese an das Datenerfassungssystem zu übertragen. Dafür kann über eine 1G-Ethernet-Schnittstelle mit dem Datenerfassungssystem kommuniziert werden. Für das Verwalten, Formatieren und Zwischenspeichern von Daten kann der SoC mit 1 GB Synchronous Dynamic Random-Access Memory (SDRAM) kommunizieren, welcher sich ebenso auf dem Zynq Board befindet. Des Weiteren befindet sich für Entwicklungszwecke eine JTAG-Schnittstelle und eine USB-Schnittstelle auf dem Ausleseboard. Über einen SD-Kartenslot kann beim Starten des Strahlendetektors ein Programm in den

SoC geladen werden. In der Entwicklungsphase findet dieser Prozess in der Regel über die JTAG-Schnittstelle statt. Die Spannungsversorgung erfolgt durch ein externes Netzteil mit einer Spannung von 12V. Auf dem Ausleseboard werden über Spannungsregler verschiedene Spannungen für den Zynq und für Timepix3 bereitgestellt.

2.2. Timepix3

Um Timepix3 genauer zu beschreiben, wird in Sektion 2.2.1 die Funktionsweise von hybriden Pixeldetektoren erklärt. Anschließend wird der interne Aufbau von Timepix3 in Sektion 2.2.2 beschrieben. Danach wird das elektronische Interface in Sektion 2.2.3 erklärt. In der letzten Sektion werden Protokolle zur Kommunikation mit Timepix3 vorgestellt.

2.2.1. Hybride Pixeldetektoren

Wie in Abbildung 2.2 zu sehen ist, besteht ein hybrider Pixeldetektor aus einer Sensorschicht und einer Elektronikschicht. Die Sensorschicht ist dabei elektronisch über Bump Bonds mit

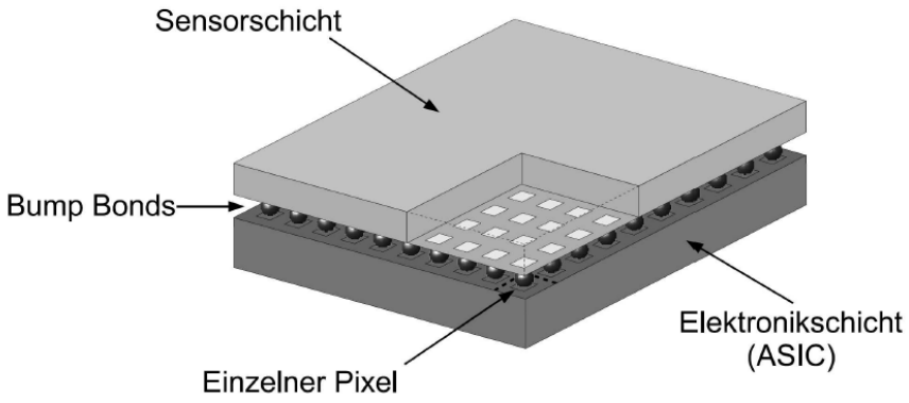

Abbildung 2.2.: Aufbau eines hybriden Pixeldetektors [1, S. 3]

der Elektronikschicht verbunden, wodurch die Anschlussfläche reduziert wird [12]. Demzufolge ist es möglich, mehrere hybride Pixeldetektoren nebeneinander anzuordnen und die Sensorfläche bündig zu erhöhen. Die Bump Bonds sind äquidistant und quadratisch angeordnet, sodass jeder von ihnen ein Pixel einer zweidimensionalen Pixelmatrix repräsentiert. Abhängig vom Halbleitermaterial der Sensorschicht, reagiert diese auf eintreffende Teilchen, indem Ladung in der Sensorschicht freigesetzt wird. Die Elektronikschicht detektiert die Ladung über Bump Bonds und verarbeitet diese zu elektronischen Rohdaten. Auf diesem Weg können Informationen, wie zum Beispiel die Anzahl an eingetroffenen Teilchen pro Pixel, aus der Strahlung gewonnen werden. [13]

Timepix3 ist ein hybrider Pixeldetektor und der Nachfolger von Timepix. Er wurde entwickelt, um die Anzahl, die Ankunftszeit und die Energie von eintreffender Teilchen auf einem Pixel zu detektieren. Die 256 x 256 große Pixelmatrix bildet mit einer Pixelgröße von 55 x

55 µm eine Sensorfläche von 1.4 x 1.4 cm. Timepix3 besitzt einen Dateneingang sowie acht Datenausgänge für Datenraten von bis zu 5 GBit/s. Eine Besonderheit von Timepix3 ist der Ausgabemodus “Event-Driven-Readout“. Bei herkömmlichen Ausgabemodi wird bei einer Aufnahme die komplette Pixelmatrix ausgegeben. Dabei werden ebenso Informationen über Pixel gesendet, die nicht von Strahlung getroffen wurden. Das “Event-Driven-Readout“ hingegen ignoriert solche Pixel. Allerdings muss beim “Event-Driven-Readout“ jedem Pixel eine Adresse zugeordnet werden, damit die ausgegebene Information dem entsprechenden Pixel zugewiesen werden kann. Dennoch wird das “Event-Driven-Readout“ als schnellere und zukunftsweisende Technik bei der Entwicklung von hybriden Pixeldetektoren betrachtet.[2][10]

2.2.2. Physische Beschreibung

Wie in Abbildung 2.3 zu sehen ist, besteht Timepix3 aus einer Pixelmatrix und einer Peripherie, die mit der Pixelmatrix und dem Auslesesystem kommuniziert. Die Pixelmatrix

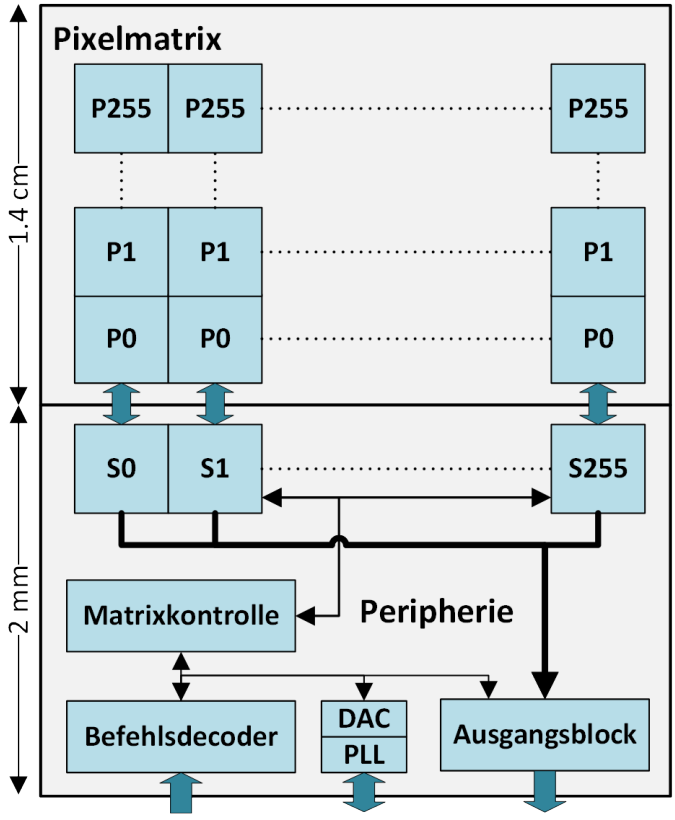

Abbildung 2.3.: Vereinfachter Aufbau von Timepix3

besteht aus 65536 Pixeln (P), die elektronisch in Spalten (S) angeordnet sind. Dabei besitzt jeder Pixel eine feste Adresse und Register, welche Informationen über eingetroffene Teilchen repräsentieren. Bei einer Aufnahme wird die Pixelmatrix spaltenweise auf Pixel mit neuen Informationen überprüft. Abhängig von dessen Konfiguration codiert der Ausgangsblock die

Informationen und sendet sie in einem vordefinierten Format an das Auslesesystem. Eine interne Phase-Locked Loop (PLL) versorgt Timepix3 mit verschiedenen Takten. Über Digital-to-Analog Converter (DAC)s lassen sich analoge Schwellspannungen einstellen, welche die Sensitivität auf die Energie von ionisierenden Teilchen beeinflussen. Der Befehlsdecoder in der Peripherie dekodiert vom Auslesesystem gesendete Befehle. Mit Befehlen (vgl. Abschnitt 2.2.4) können Timepix3-Register konfiguriert und Aufnahmen über die Matrixkontrolle eingeleitet werden. Folglich können über Register der Ausgangsblock, die PLL und die analogen Spannungen der DACs konfiguriert werden. Da Timepix3 verschiedene Informationen über eingetroffene Teilchen detektiert, können diese über Register selektiert werden. Zusätzlich besitzt jedes Pixel ein Pixelregister. Über dieses Register kann beispielsweise ein Pixel maskiert werden, sodass keine Daten von dem Pixel erfasst werden. Für alle Konfigurationsoptionen und den genauen Registersatz von Timepix3 wird auf den Anhang A.1 verwiesen.

Im Folgenden wird erklärt, wie Timepix3 ein ionisierendes Teilchen erfasst. Um die eintreffende Strahlung in digitale Information umzuwandeln, muss ein analoges Pixelfrontend auf Ladungstrennung in der Sensorschicht reagieren. Das analoge Pixelfrontend ist eine elektronische Schaltung, die sich hinter jedem Pixel der Pixelmatrix befindet. Trifft ein ionisierendes Teilchen auf den Sensor, wird durch freie Ladungsträger in der Sperrschicht des Sensors ein Strompuls erzeugt, der vom analogen Frontend detektiert wird. Dabei kann die Sensorschicht als Stromquelle, die auf ionisierende Teilchen reagiert, betrachtet werden.

In Abbildung 2.4 wird das analoge Frontend von Timepix3 vereinfacht dargestellt. Der

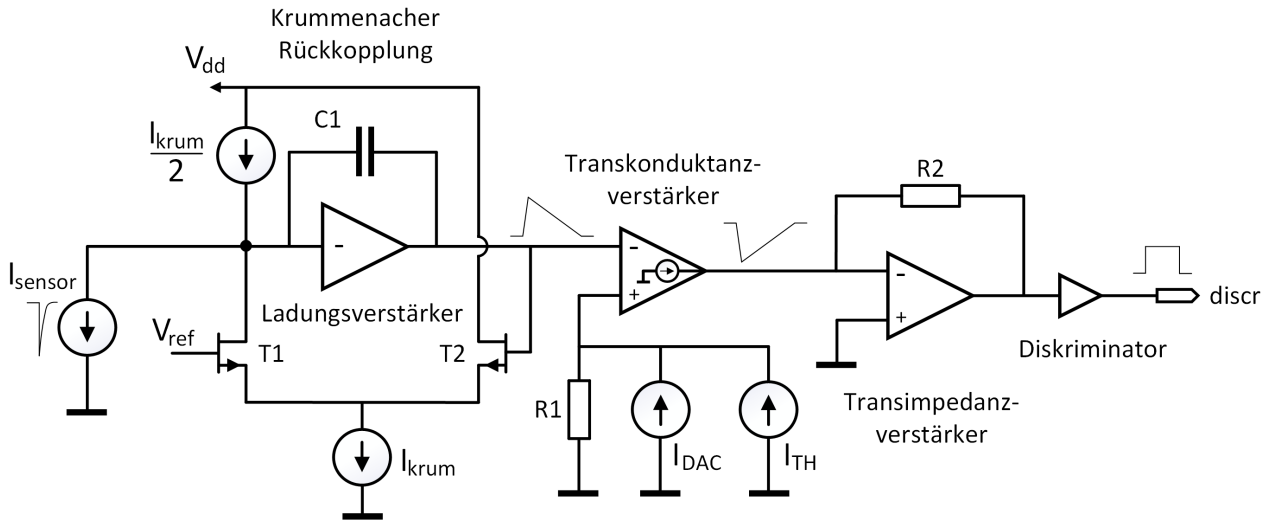

Abbildung 2.4.: Vereinfachter Funktionsplan des analogen Pixelfrontends

Sensor wird mit der Stromquelle "I_{sensor}" simuliert. Durch die Krummenacher Rückkopplung ist es möglich, schnell hintereinander eintreffende Teilchen zu detektieren. Des Weiteren wird über die Krummenacher Rückkopplung der Leckstrom des Sensors kompensiert. Die Leckstromkompensation ist in dem vereinfachten Funktionsplan nicht dargestellt. Im Ruhezustand ist der Kondensator "C1" der Krummenacher Rückkopplung geladen. Trifft ein Teilchen auf den Sensor, erzeugt "I_{sensor}" einen Strompuls, der den Kondensator "C1" entlädt.

Infolgedessen steigt die Ausgangsspannung der Krummenacher Rückkopplung an. Daraufhin erhöht sich der Drainstrom durch den Transistor “T2“. Da die Summe der Ströme durch beide Transistoren immer “I_{krum}“ ist, sinkt der Stromfluss durch “T1“. Demzufolge erhöht sich der Stromfluss in den Kondensator, sodass dieser sich linear auflädt. Auf Grund dessen sinkt die Ausgangsspannung der Krummenacher Rückkopplung. Die Höhe des Stromes “I_{krum}“ entscheidet dabei, wie schnell sich der Kondensator wieder auflädt. Die Ausgangsspannung wird über einem Transkonduktanzverstärker mit einer Schwellspannung verglichen. Durch die Höhe der Schwellspannung wird entschieden, ob der erzeugte Stromfluss durch das ionisierende Teilchen, als Treffer gewertet werden soll. Die Höhe der Schwellspannung wird über die Ströme “I_{TH}“ und “I_{DAC}“ gesteuert. Wobei “I_{TH}“ global für alle Pixel eingestellt wird und “I_{DAC}“ für jeden Pixel separat über ein Pixelregister eingestellt werden kann. Am Ende des Frontends digitalisiert ein Diskriminator die von dem Transimpedanzverstärker ausgegebene Spannung. Es folgt ein digitaler Puls “discr“, der vom digitalen Pixelfrontend ausgewertet wird.[14][15]

Abhängig von der Information, die Timepix3 ausgeben soll, gibt das digitale Pixelfrontend unterschiedliche digitale Informationen über das Signal “discr“ aus. Abbildung 2.5 zeigt wie der Puls ausgewertet wird. Dabei werden in dieser Arbeit lediglich die eingetroffenen Teilchen pro Pixel betrachtet. Der Puls wird mit dem Signal Shutter UND-Verknüpft, sodass dieser

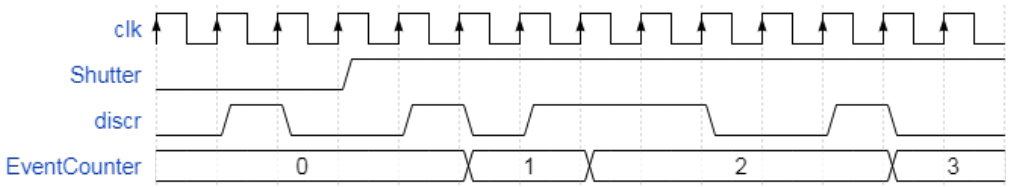

Abbildung 2.5.: Zeitablaufdiagramm digitales Pixelfrontend

nur verarbeitet wird, wenn Shutter Eins ist. Das Signal “EventCounter“ zählt die Anzahl der Pulse von “discr“ und repräsentiert somit die eingetroffenen Teilchen pro Pixel. Bei anderen Informationsmodi können beispielsweise über die Länge des Pulses Rückschlüsse auf die Energie des eingetroffenen Teilchens gewonnen werden.[2, S. 14]

2.2.3. Interface

Timepix3 besitzt für die Kommunikation sieben Steuereingänge, ein Dateneingang und acht Datenausgänge. Dazu kommen zwei Takteingänge und ein Taktausgang. Des Weiteren verfügt der Chip über 73 Eingänge für die Spannungsversorgung und einen Ein- und Ausgang für externe DAC-Operationen. Die Datenübertragung zum Chip erfolgt seriell mit einem Takt von 40 MHz über die Eingänge DataIn und ClkIn40. Alle Eingangssignale werden bei steigender Flanke von ClkIn40 aufgenommen. Die Datenausgänge DataOut[7:0] sind taktsynchron mit ClkOut. ClkOut taktet abhängig von der Konfiguration mit Frequenzen von bis zu 320 MHz. Zusätzlich werden die Daten im Double Data Rate (DDR) Format übertragen, sodass Daten mit steigender und fallender Taktflanke gesendet werden. Demnach kann jeder DataOut-Ausgang maximal 640 MBit/s senden. Zusätzlich ist es möglich, über interne

Register von Timepix3 einzelne Datenausgänge zu aktivieren und zu deaktivieren. ClkInRefPLL ist der Takteingang der internen PLL. PLLOut kann abhängig von der internen Konfiguration verschiedene interne Signale von Timepix3 ausgeben. Der einpolig geerdete CMOS-Eingang SLVS_TERM ermöglicht die Aktivierung von einem internen Abschlusswiderstand für die differentiellen Eingangssignale. DACOut und ExtDAC sind analoge Signale, die es Ermöglichen, interne analoge Signale zu überschreiben oder auszulesen.

Mit der Ausnahme von DACOut, ExtDAC und SLVS_TERM sind alle Ein- und Ausgangssignale differentiell realisiert. Während die Eingänge wahlweise im Low Voltage Differential Signaling (LVDS) oder Scalable Low Voltage Signaling (SLVS) Schnittstellen-Standard betrieben werden können, senden die Ausgänge lediglich im SLVS Standard [2]. SLVS unterscheidet sich von LVDS dadurch, dass der Spannungspegel und die Amplitude niedriger ausfallen [16]. Dadurch verbraucht eine Übertragung im SLVS-Standard weniger Leistung als im LVDS-Standard. Die genauen Aufgaben der einzelnen Signale sind in Tabelle 2.1 zusammengefasst.

Richtung	**Name**	**Aufgabe**
Eingang	ClkIn40	Eingangstakt synchron zu den Eingängen
Eingang	DataIn	Dateneingang zum Senden von Befehlen
Eingang	EnableIn	Zeigt, dass ein Befehl gesendet wird
Eingang	Reset	Setzt den Chip zurück
Eingang	T0_Sync	Synchronisiert interne Zähler für Zeitmessungen
Eingang	Shutter	Startet eine Aufnahme
Eingang	EnablePower Pulsing	Aktiviert die analoge Spannungsversorgung der Pixelmatrix
Eingang	ExtTPulse	Ein Testpuls für das Testen einzelner Pixel
Eingang	ClkInRefPLL	Eingangstakt für die interne PLL
Ausgang	ClkOut	Ausgangstakt synchron zum Datenausgang
Ausgang	DataOut[7:0]	Datenausgang
Ausgang	PLLOut	Konfigurierbarer Signalausgang von verschiedenen internen Signalen
Ausgang	DACOut	Analog, kann interne analoge Signale ausgeben
Eingang	ExtDAC	Analog, kann interne analoge Signale überschreiben
Eingang	SLVS_TERM	CMOS, aktiviert die internen 100 Ohm Abschlusswiderstände der Eingänge

Tabelle 2.1.: Interface Timepix3

Zusätzlich zum DDR-Format kann über Timepix3-Register der 8B/10B-Leitungscode für die Datenausgänge aktiviert werden. In der 8B/10B-Codierung werden 8 Bit auf 10 Bit abgebildet, um die Taktrückgewinnung aus dem Datensignal zu ermöglichen und einen Gleichspannungsausgleich durchzuführen [17, S. 7]. Dabei gibt es drei Arten von 10-Bit-Codewörtern, die unterschiedliche Anzahlen von “1“ und “0“ führen:

- Neutral (fünf “0“ und fünf “1“)

- Positiv (vier “0“ und sechs “1“)
- Negativ (sechs “0“ und vier “1“)

Ein 8-Bit-Wert wird entweder auf einem neutralem Codewort oder auf einem positiven und einem negativen Codewort abgebildet. Der Codierer entscheidet, ob ein positives oder ein negatives Codewort gesendet wird. Damit ist sichergestellt, dass die Differenz der Anzahl von “0“ und “1“ in einem mindestens 20-Bit-langen Signal nicht größer als zwei ist. Zusätzlich werden nicht mehr als fünf “0“ oder “1“ hintereinander gesendet. Ein weiterer Vorteil ist, dass das 8b/10b-Protokoll durch die Abbildung von acht Bit auf zehn Bit die Verwendung von zusätzlichen Steuerzeichen ermöglicht. Diese werden beispielsweise für Synchronisationssequenzen genutzt. Demnach hat die 8B/10B-Codierung folgende Vorteile:

- Vollständiger Gleichspannungsausgleich
- Taktinformation im Signal enthalten
- Zusätzliche Steuerzeichen
- Mehr Nutzdaten als beispielsweise 1B2B-Format (Manchester-Code)

2.2.4. Kommunikationsprotokolle

Die Kommunikation mit Timepix3 erfolgt über festgelegte Operationen. Bei allen Operationen werden über den Dateneingang Pakete seriell an Timepix3 gesendet sowie über die Datenausgänge Pakete seriell von Timepix3 empfangen. Sofern mehr als ein Datenausgang aktiviert ist, sendet jeder Datenausgang seriell vollständige Pakete. Pakete sind in den meisten Operationen 48 Bit groß und können Operation starten, beenden oder Daten enthalten. Bei einzelnen Operationen ist es zusätzlich nötig, Steuersignale wie EnableIn oder Shutter zu setzen. Für die Kommunikation mit Timepix3 gibt es zwei wichtige Operationsgruppen. Über Peripherie-Operationen ist es möglich, mit den internen Timepix3-Registersatz zu kommunizieren. Pixelmatrix-Operationen dienen dazu, Aufnahmen mit der Pixelmatrix einzuleiten oder mit pixelinternen Registern zu kommunizieren. Unabhängig von der Operationsart wird beim Start einer Operation immer ein 40 Bit langer Sync-Header gesendet.

Peripherieoperationen

Wie in Abbildung 2.6 zu sehen ist, folgt bei Peripherie-Operationen nach dem Sync-Header ein OperationHeader. Der OperationHeader legt die Peripherie fest, mit der kommuniziert

Abbildung 2.6.: Timepix3 Peripherie Befehl [2, S. 26]

werden soll. Zusätzlich unterscheiden sich die OperationHeader in Schreib- und Lese-Header,

welche entscheiden, ob ein Register beschrieben oder ausgelesen werden soll. Ist der OperationHeader ein Schreib-Header, muss das DataIN-Feld die Daten enthalten, die in das Register geschrieben werden sollen. Ist der OperationHeader ein Lese-Header, wird das DataIN-Feld ignoriert. Das DataIN-Feld ist maximal 16 Bit lang, was für das vollständige Beschreiben der meisten Register ausreicht. Einige Register von Timepix3 sind jedoch 48 Bit lang. In diesem Fall gibt es drei unterschiedliche OperationHeader, die jeweils 16 Bit des Registers beschreiben können.

In Abbildung 2.7 ist der komplette Protokollablauf einer Peripherie-Schreiboperation dargestellt. Bevor ein Peripherie-Befehl über "DataIn" gesendet wird, muss EnableIn von Eins

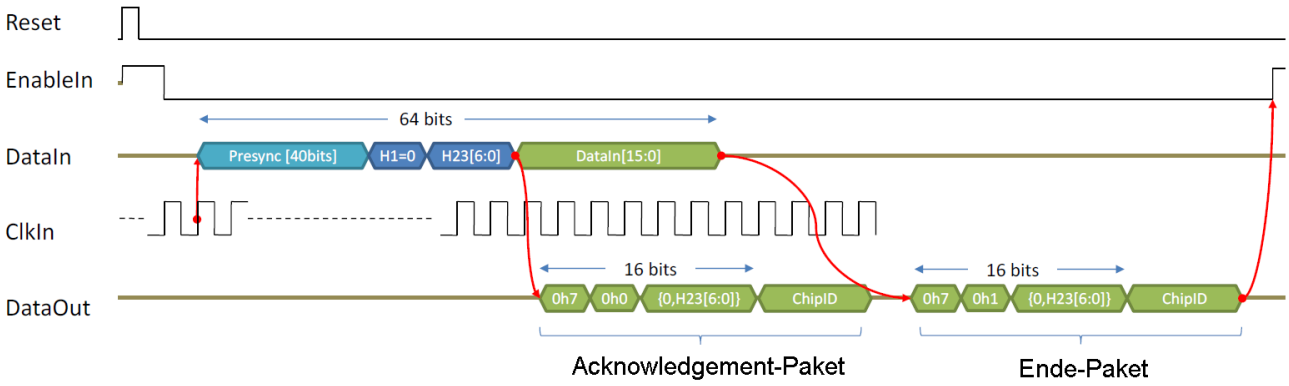

Abbildung 2.7.: Timepix3 Peripherie Operation [2, S. 33]

auf Null gesetzt werden. Timepix3 antwortet auf den Peripherie-Schreibbefehl über "DataOut" mit zwei Paketen, welche als Header den gesendeten OperationHeader besitzen. Das erste Paket ist das Acknowledgement-Paket. Es zeigt, dass Timepix3 den Peripherie-Befehl bis zum DatenIN-Feld verstanden hat. Ob das Acknowledgement-Paket gesendet werden soll, ist optional konfigurierbar. Hat Timepix3 den Befehl umgesetzt und kann somit eine neue Operation durchführen, wird das Ende-Pakete gesendet. Bei einer Peripherie-Leseoperation sendet Timepix3 zusätzlich ein drittes 48-Bit-Paket. Dieses wird nach dem Acknowledgement-Paket und vor dem Ende-Pakete gesendet. Es enthält den OperationHeader und die erfragten Daten aus dem Register. Bevor ein neuer Befehl gesendet werden kann, muss EnableIn wieder auf Eins gesetzt werden.

Pixelmatrixoperation

Pixelmatrix-Operationen sind komplexer als Peripherie-Operationen und können abhängig von der Operation in ihrem Aufbau variieren. Dabei können über Pixelmatrix-Operationen Aufnahmen eingeleitet, oder die Pixelregister ausgelesen oder beschrieben werden. Im Folgenden werden die sequentielle Aufnahme und das Beschreiben der Pixelregister vorgestellt.

Der Operationsvorgang für das Beschreiben der sechs Bit langen Pixelregister ist in Abbildung 2.8 dargestellt. Die Operation wird über DataIn mit einem Sync-Header eingeleitet, gefolgt von dem Byte "0x80". Danach erwartet Timepix3 ein ColumnMask-Paket, das 256

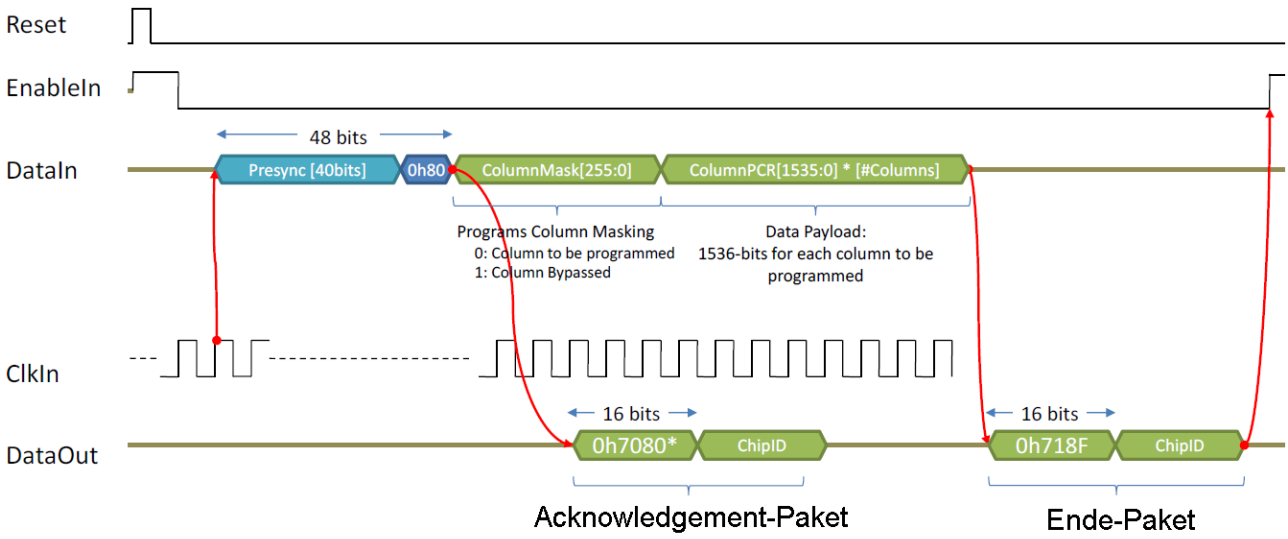

Abbildung 2.8.: Timepix3 Pixelregister Operation [2, S. 45]

Bit lang ist. Über das ColumnMask-Paket wird festgelegt, in welcher Spalte die Pixelregister beschrieben werden sollen. Folglich repräsentiert jedes Bit des ColumnMask-Paketes eine Spalte der Pixelmatrix. Im darauffolgenden ColumnPCR-Paket werden die Registerinhalte von den Pixeln in der ausgewählten Spalten angegeben. Abhängig davon, wie viele Spalten im ColumnMask-Paket selektiert werden, variiert die Länge vom ColumnPCR-Paket. Wird beispielsweise nur eine Spalte im ColumnMask-Paket selektiert, ist das ColumnPCR-Paket 1536 Bit lang. Das entspricht genau sechs Bit für 256 Pixel. Werden im ColumnMask-Paket alle Spalten selektiert, so ist das ColumnPCR-Paket 393216 Bit lang. Das entspricht genau sechs Bit für 65536 Pixel. Durch definierte Strukturierung des ColumnPCR-Paketes ist es möglich, jedes Pixelregister der Pixelmatrix individuell zu konfigurieren. Hat Timepix3 die Operation verarbeitet, wird ein Ende-Paket gesendet.

Um Daten aus der Pixelmatrix zu erhalten, muss eine Aufnahme gestartet werden. Bei einer sequentiellen Aufnahme findet die Aufnahme getrennt von der Ausgabe der Daten statt. Demgemäß besteht die Aufnahme aus zwei Operationen. Die erste Operation leitet die Aufnahme ein und die zweite Operation ließt die durch die Aufnahme entstandenen Daten aus. Der Operationsablauf ist in Abbildung 2.9 dargestellt. Die Operation wird über das ReadMatrixSeq-Paket eingeleitet. Es beginnt mit einem Sync-Header, gefolgt von dem Byte “0xA0“. Im Anschluss wird ein 256-Bit-Paket gesendet, das Spalten der Pixelmatrix für die Aufnahme aktiviert oder deaktiviert. Wurde der Befehl korrekt empfangen, sendet Timepix3 drei Pakete, welche die korrekte Verarbeitung des Befehls signalisieren. Anschließend muss EnableIn auf Eins gesetzt werden. Bis hierhin werden noch keine eintreffenden Teilchen, welche über das Signal Qin repräsentiert werden, in der Pixelmatrix aufgenommen. Erst wenn das Signal Shutter auf Eins geht, wird das Ausgangssignal des analogen Pixelfrontends auf das digitale Pixelfrontend gesendet und eintreffende Teilchen aufgenommen. Gehen Shutter und EnableIn wieder auf Null, ist die Aufnahme abgeschlossen. Um die Daten auszulesen, muss erneut das ReadMatrixSeq-Paket gesendet werden. Darauf folgt ein 256-Bit-Paket,

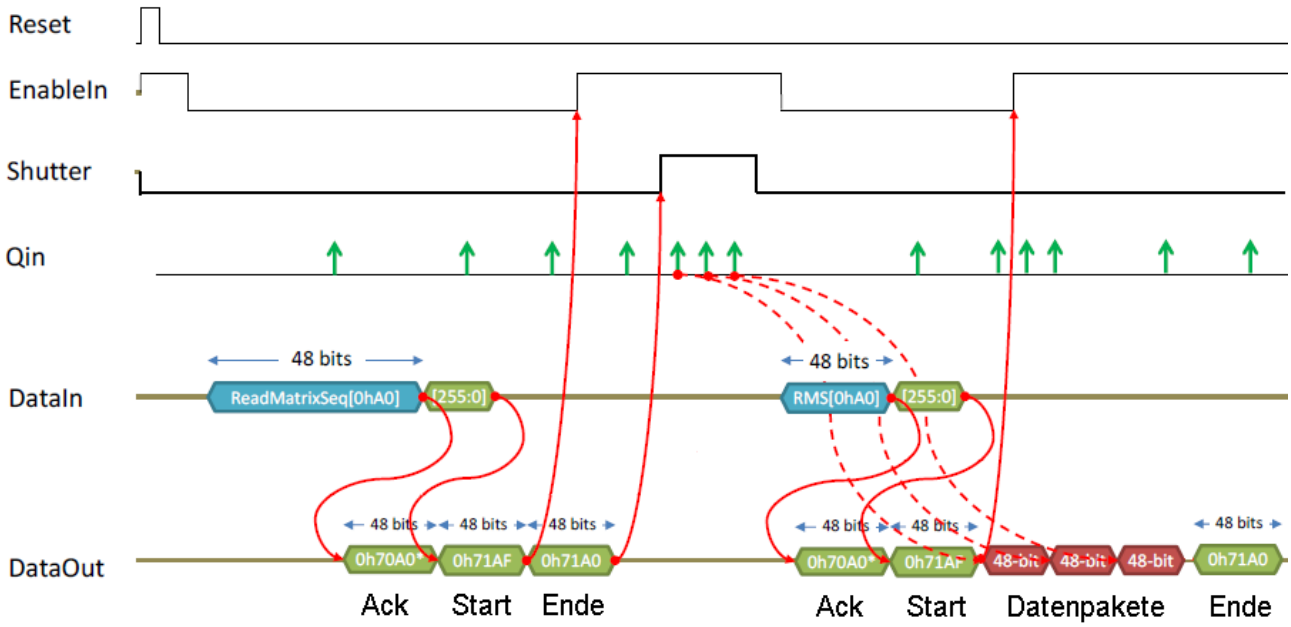

Abbildung 2.9.: Timepix3 Aufnahme Operation [2, S. 47]

das Spalten der Pixelmatrix bei der Ausgabe der Daten aktiviert oder deaktiviert. Danach zeigt Timepix3 mit zwei Paketen an, dass der gesendete Befehl akzeptiert wurde. Es folgen Datenpakete, die Informationen über die eingetroffenen Teilchen enthalten.

Abbildung 2.10 zeigt, wie ein Datenpaket aufgebaut ist. Ein Datenpaket ist 48 Bit lang. Um

Abbildung 2.10.: Timepix3 Datenpakete [2, S. 9]

das Paket als Datenpaket identifizieren zu können, beginnt jedes Datenpaket mit einem vier Bit breiten Header. Über das 16 Bit lange Adressfeld ist es möglich, die Informationen von einem Datenpaket einen der 65536 Pixel zuzuordnen. Die Informationen über eingetroffene Teilchen werden durch die restlichen 28 Bit dargestellt. Abhängig vom Informationsmodi variiert dieser Teil des Datenpaketes. Durch den Ausgabemodus “Event-Driven-Readout“ werden nur Datenpakete für Pixel mit eingetroffenen Teilchen ausgegeben. Daher ist es vor der Aufnahme unbekannt, wie viele Datenpakete Timepix3 sendet. Mit dem Ende-Paket signalisiert Timepix3 das Ende der Aufnahme. [2]

2.3. ZYNQ Ultrascale+

Der ZYNQ Ultrascale+ ist ein SoC der Firma Xilinx [18]. SoCs werden in eingebetteten Systemen eingesetzt und sind Chips, die neben integrierten Schaltkreisen, Taktgebern und Speicher auch Mikroprozessoren beinhalten. Wie in Abbildung 2.11 zu sehen ist, kann der

Zynq Ultrascale+ in zwei funktionelle Einheiten eingeteilt werden. Das Processing System

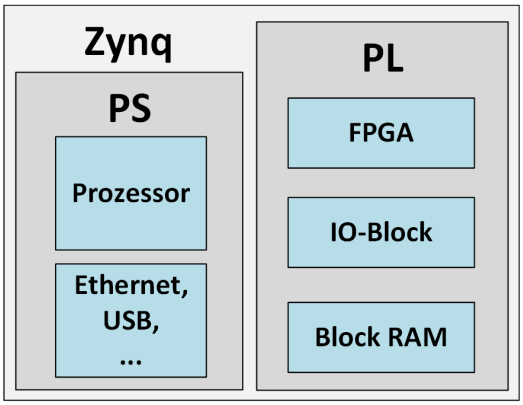

Abbildung 2.11.: Blockschaltbild Zynq

(PS) setzt sich aus Prozessoren und Peripherieschnittstellen wie zum Beispiel USB und Ethernet zusammen. Außerdem besitzt das PS Speichercontroller, die es ermöglichen, mit externen RAM zu kommunizieren. Über die Programmable Logic (PL) kann das PS spezifisch erweitert werden. Dafür besteht die PL unter anderem aus einem FPGA und Block-RAM. Block-RAM sind Speicherzellen, die zum Beispiel für die Implementierung von First In First Out (FIFO)s verwendet werden können. Zusätzlich können in der PL bereits bestehende IP-Cores der Xilinx Bibliothek oder VHDL-Module implementiert werden. Des Weiteren verfügt die PL über Input Output (IO)-Blöcke, welche direkt mit Pins des Zynqs verbunden sind. Dabei verfügen IO-Blöcke über eine Reihe an Primitiven. Primitive sind fest integrierte Schaltungen wie zum Beispiel differentielle Ausgangs- und Eingangsbuffer für das Senden und Empfangen von differentiellen Signalen. PS und PL können über AXI4-Bussysteme miteinander kommunizieren.

Die Entwicklungsumgebung Vivado in Verbindung mit Xilinx SDK ermöglicht es, Programme für den Zynq zu entwickeln. Dabei ist Vivado für die Entwicklung, Synthese und Implementierung von digitalen Schaltungen im Zynq verantwortlich. Der Prozessor des Zynqs hingegen kann in Xilinx SDK in Programmiersprachen wie C oder C++ programmiert werden. [19]

Bussysteme

Im Zynq werden verschiedene Bussysteme zur Kommunikation genutzt. Die drei wichtigsten Bussysteme des Zynqs sind der AXI4-Memory-Map-, der AXI4-Lite- und der AXI4-Stream-Bus. Sie besitzen unterschiedliche Eigenschaften in Hinsicht auf die Adressierungsfähigkeit, die Datenburstfähigkeit und die Datenbreite. Ein Datenburst findet statt, wenn Daten zusammenhängend hintereinander gesendet werden. In Kombination mit der Adressierungsfähigkeit wird eine Adresse gesendet und mit einem Burst werden Daten fortlaufend geschrieben oder gelesen.

Die Eigenschaften der Bussysteme sind in Tabelle 2.2 aufgeführt. Der AXI4-Memory-Map-

Bus	Adresse	Datenburst	Datenbreite
Memory-Map	Ja	Ja	bis zu 1024 Bit
Lite	Ja	Nein	in Xilinx IP-Cores 32 Bit
Stream	Nein	Ja	variabel, i.d.R. 32 Bit

Tabelle 2.2.: AXI4-Bussysteme

Bus wird für den Transfer von großen Datenmengen eingesetzt. Er ist ressourcenaufwändig und wird in der Regel von bereits bestehenden Xilinx IP-Cores verwendet. Der AXI4-Lite-Bus dient hauptsächlich zur Kommunikation zwischen PS und PL. Demgemäß besitzen viele Xilinx IP-Cores eine AXI4-Lite-Schnittstelle, die es dem PS ermöglicht, mit dem IP-Core zu kommunizieren. Der AXI4-Stream-Bus wird hauptsächlich in der PL des Zynqs verwendet, um zum Beispiel Daten in FIFOs zwischenzuspeichern. Wie Abbildung 2.12 zeigt, kann dieser bereits mit drei Signalen Informationen übertragen. Dafür signalisiert der Sender mit

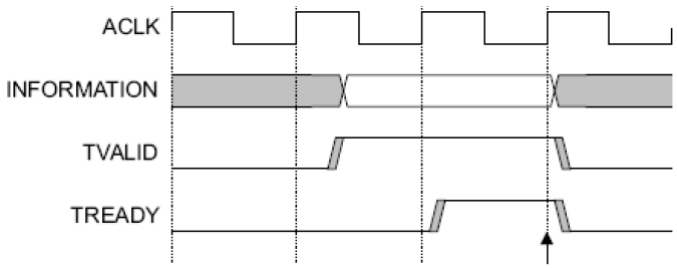

Abbildung 2.12.: AXI4-Stream Protokoll [3, S. 34]

dem Signal “tvalid“, dass “information“ neue Daten enthält. Ist der Empfänger bereit für neue Daten, setzt dieser das Signal “tready“ auf Eins. Dies kann auch geschehen, bevor das Signal “tvalid“ Eins ist. Sind bei einer steigenden Flanke “tready“ und “tvalid“ Eins, findet eine Übertragung statt. Ein Datenburst findet statt, indem die Signale “tready“ und “tvalid“ nach einer Übertragung nicht auf Null gehen und somit bei der nächsten steigenden Flanke die nächste Übertragung stattfindet. [20]

Datentransfer

Um größere Datenmengen zwischen PS und PL zu übertragen, werden diese während der Übertragung zwischengespeichert. Dabei wird entweder ein Direct Memory Access (DMA)-Controller oder ein FIFO verwendet. Ein FIFO befindet sich in der PL des Zynqs und überträgt Daten über den AXI4-Lite-Bus zwischen PS und PL. Der DMA-Controller ist schneller in der Übertragung als ein FIFO und wird im Folgenden näher erklärt.

Wie in Abbildung 2.13 zu sehen ist, kann der Controller den externen SDRAM mit Hilfe des Memory Controllers auslesen und beschreiben. Ein Datentransfer zwischen PL und PS ist möglich, da der Prozessor ebenfalls über den Memory Controller mit dem externen RAM verbunden ist. Gleichzeitig kommuniziert der DMA-Controller über eine AXI4-Lite-Schnittstelle mit dem Prozessor. Dabei werden lediglich Konfigurationen und Informationen

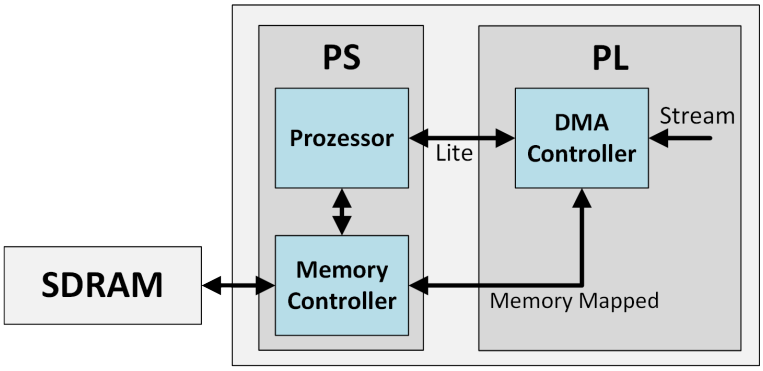

Abbildung 2.13.: Timepix3 DMA-Controller

der Übertragung ausgetauscht. Der Datentransfer findet ausschließlich über den Memory Controller statt. Um eine schnelle Übertragung zu ermöglichen, arbeitet der DMA-Controller mit Speicherbereichen. Die Anzahl und Größe der Speicherbereiche ist über das PS konfigurierbar. Werden Daten von der PL in den DMA-Controller geschrieben, beschreibt dieser fortlaufend die konfigurierten Speicherbereiche im SDRAM. Jeder Speicherbereich wird von einem Deskriptor beschrieben. Im Deskriptor sind Information wie die Adresse des Speicherbereiches und die Größe des beschriebenen Speichers hinterlegt. Über die Deskriptoren kann das PS die Adressen von beschriebenen Speicherbereichen auslesen und die dort hinterlegten Daten verarbeiten. [21]

3. Anforderungsanalyse

In diesem Kapitel werden die Anforderung an das Projekt definiert. Das Ziel dieser Arbeit ist es, ein Fundament für die weitere Entwicklung eines Auslesesystems für den Nachfolger Timepix4 zu legen. Dafür soll eine Auslesefirmware für das Timepix3 basierende Detektorsystem entwickelt werden. Die Hardware des Auslesesystemes wurde zuvor von der Detektorgruppe am DESY entwickelt und wird für dieses Projekt bereitgestellt. Die Hardwareschnittstellen zwischen den einzelnen Komponenten sind vordefiniert, sodass die Aufgabe ausschließlich in der Firmware- beziehungsweise der Softwareentwicklung des Systemes liegt. Um die Anforderung eines Timepix-Systemes besser nachvollziehen zu können, sollen die grundlegende Kommunikation und Funktion von Timepix3 analysiert und erprobt werden.

Es muss möglich sein, mit Timepix3 zu kommunizieren und Datenpakete, die Informationen über die eingetroffenen Teilchen enthalten, von Timepix3 an das Datenerfassungssystem zu senden. Zusätzlich sind Nutzer der Pixeldetektoren an einer hohen Bildfrequenz interessiert. Daher muss das System für die Verarbeitung hoher Datenraten ausgelegt sein. Allerdings steht die Entwicklung eines Systems, das die Erprobung und Analyse von Timepix3 ermöglicht, im Vordergrund. Um die Funktion und die Eigenschaften von Timpix3 schneller analysieren zu können, wird in dieser Arbeit lediglich mit einem von vier Timepix3-Chips kommuniziert. Des Weiteren soll der Chip lediglich mit einen von acht möglichen Ausgangskanälen arbeiten. Dabei soll der Ausgangstakt des Kanals mit 160 MHz betrieben werden. Dies verringert die Datenrate und somit die Bildfrequenz. Jedoch ist durch die niedrigere Datenrate und die längeren Signallaufzeiten die Implementierung einfacher. Somit können Funktionen von Timepix3 schneller erprobt und getestet werden. Dennoch soll das Konzept auf vier Timepix3 Chips mit jeweils acht aktiven Ausgangskanälen erweiterbar sein. Daher muss das System so konzipiert werden, dass es prinzipiell auch Daten mit acht aktiven Ausgangskanälen und vier Timepix3-Chips erfassen kann.

Um die Hauptanforderungen zu erfüllen, muss eine Aufnahme mit Timepix3 durchgeführt werden. Daraus lassen sich folgende Anforderungen an das System ableiten:

- Es muss möglich sein, interne Timepix3-Register zu setzen.
- Um Operationen zu starten, müssen Steuersignale korrekt gesetzt werden.
- Die Eingangsdaten müssen seriell in Paketen gesendet werden, welche Register beschreiben oder Funktionen von Timepix3 ausführen.
- Die Pixelmatrix muss über das Senden eines maximal 48kByte langen Datenpaketes konfiguriert werden können.
- Der gegebenenfalls codierte DDR-Ausgangsdatenstrom von Timepix3 muss korrekt aufgenommen werden.

- Da der Datenstrom außer Pakete auch Synchronisationssignale enthält, müssen die Pakete von den Synchronisationssignalen getrennt werden.

- Bei der Übertragung der Daten dürfen keine Pakete beschädigt werden oder verloren gehen.

Auf dem Datenerfassungssystem muss eine Software mit dem Ausleseboard über Ethernet in Verbindung stehen. Um Aufnahmen zu starten, muss sie Eingaben des Nutzers entgegennehmen und auswerten. Anschließend muss die Software die durch die Aufnahme entstandenen Daten empfangen, gegebenenfalls verarbeiten und in einer Datei abspeichern. Dabei ist es optional, die Daten auf dem Datenerfassungssystem zu dekodieren und anschließend zu visualisieren. In einer Visualisierung könnte die Anzahl der eingetroffenen Teilchen pro Pixel in einem zweidimensionalen Bild, das die Pixelmatrix von Timepix3 repräsentiert, farblich dargestellt werden. Hierzu muss die Pixelmatrix von Timepix3 korrekt konfiguriert und kalibriert werden. Dies würde weitere Einblicke auf die Funktionen von Timepix3 ermöglichen.

Durch die vorgegebene Hardware steht fest, dass alle Signale von und zu Timepix3 mit der PL des Zynqs verbunden sind. Die Ethernetkommunikation geschieht jedoch über das PS. Ob die einzelnen Anforderungen im Zynq oder im Datenerfassungssystem umgesetzt werden, ist freigestellt. Um das System sicher zu testen, sollen die Anforderungen in Modulen aufgeteilt werden, die alle separat verifiziert werden können. Das hat den Vorteil, dass jedes Modul einzeln getestet werden kann und somit Fehler schneller erkannt werden können. Das Gesamtsystem kann getestet werden, indem eine Aufnahme erfolgreich durchgeführt wird und die daraus entstandenen Pakete betrachtet werden. Dafür müssen die vom Datenerfassungssystem abgespeicherten Pakete mit den von Timepix3 gesendeten Paketen übereinstimmen. Die Verifikation des Inhaltes der Pakete ist optional und kann über das Dekodieren und Visualisieren der Daten erfolgen. Dafür muss Timepix3 mit einer bekannten Strahlungsquelle beschossen werden. Über das daraus entstehende Bild kann die Korrektheit der Daten sowie die Übertragungsstrecke des Systems verifiziert werden.

4. Konzeptentwurf

In diesem Kapitel wird die Konzeptionierung des Systems durchgeführt. Es werden dabei unterschiedliche Lösungswege für die Bearbeitung der Anforderungen aus Kapitel 3 gegenübergestellt und analysiert. Der erste Abschnitt dieses Kapitels beschäftigt sich mit dem Gesamtkonzept des Systems. Hier wird diskutiert, in welchem Teil des Systemes welche Anforderung erfüllt werden sollen. In den darauffolgenden Abschnitten werden die aus dem Gesamtkonzept resultierenden Module konzipiert.

4.1. Gesamtkonzept

Um ein Gesamtkonzept aufzustellen, wird sich an den Anforderungen aus Kapitel 3 orientiert. Dabei werden die Anforderungen einzelnen Modulen im System zugewiesen. Durch die Hardware ist Timepix3 mit der PL und das Datenerfassungssystem mit dem PS verbunden. Jedoch können durch die schnelle Kommunikationsmöglichkeit aller Teile sämtliche Module des Systemes im PS, in der PL oder im Datenerfassungssystem realisiert werden.

Zuerst soll festgelegt werden, in welchem Teil des Systemes die Daten an Timepix3 gesendet werden. Die Taktleitung und die Datenleitung sind direkt mit der PL des Zynqs verbunden. Das Senden der seriellen Bits mit dem Takt muss demnach in der PL erfolgen. Die anderen Aufgaben für das Senden der Daten können in der PL sowie in dem PS vom Zynq realisiert werden. Für eine komplette Realisierung in der PL spricht, dass keine Kommunikationsschnittstelle zwischen PS und PL implementiert werden muss. Allerdings sind die Vorteile vom PS wie zum Beispiel das einfache sequentielle Abarbeiten von Aufgaben dadurch nicht nutzbar. Beim Initialisierungsvorgang von Timepix3 müssen verschiedene Register nacheinander gesetzt werden. Das bedeutet, dass Pakete mit unterschiedlichen Inhalten nacheinander an Timepix3 gesendet werden müssen. Für die Auswahl der Pakete, welche gesendet werden sollen, kann es daher durchaus nützlich sein, das PS mit einzubeziehen. Hier können Pakete nacheinander über eine Funktion an die PL gesendet werden. Der Inhalt der Pakete kann auf diesem Weg flexibel im Funktionsparameter geändert werden. Bei einer Realisierung in der PL hingegen kann der Initialisierungsvorgang in einem Zustandsautomaten durchgeführt werden. Dieser kann mit jedem voranschreitenden Zustand ein weiteres Register von Timepix3 beschreiben. Das hat jedoch zur Folge, dass bei jeder Änderung eines Wertes für ein Paket der VHDL-Code für die PL neu synthetisiert und implementiert werden muss. Der Synthese- und Implementationsprozess ist ein zeitaufwändiger und komplexer Vorgang. Das Compilieren von C-Code benötigt jedoch nur wenige Sekunden. Außerdem spricht für die Befehlsauswahl im PS, dass die Funktion für das Senden von Paketen ebenfalls von einem Ereignis der Ethernetkommunikation aufgerufen werden kann, da diese direkt mit dem PS verbunden ist. Folglich kann das Senden von Timepix3-Befehlen

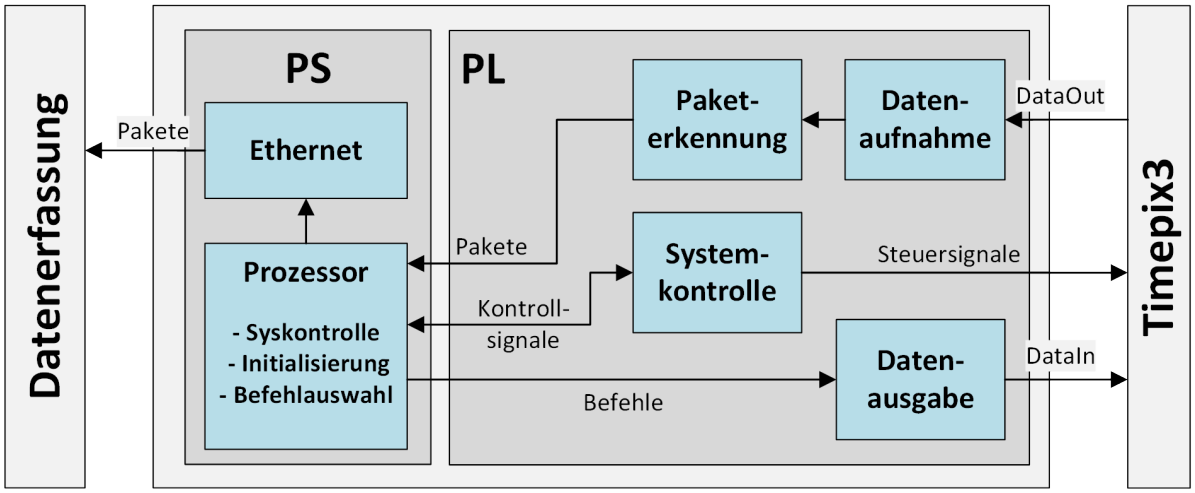

Abbildung 4.1.: Gesamtkonzept Blockschaltbild
Systemkontrolle ist aus Darstellungsgründen nicht mit dem Rest des Systemes verbunden.

vom Datenerfassungssystem einfacher implementiert werden. Aus diesem Grund und wegen der Flexibilität bei der Initialisierung für den Erstbetrieb mit Timepix3 findet die Befehlsauswahl im PS statt. Das vorgestellte Konzept ist unter anderem in Abbildung 4.1 dargestellt. Wie die erforderliche Kommunikationsschnittstelle zwischen PS und PL und das Serialisieren der Pakete konzipiert ist, wird in Abschnitt 4.2 diskutiert.

Zunächst muss festgelegt werden, in welchem Teil des Systemes der Ausgangsdatenstrom von Timepix3 verarbeitet wird. Dabei soll eine Paketerkennung die Pakete von Synchronisationssignalen im Ausgangsdatenstrom von Timepix3 trennen. Zur Auswahl stehen das Datenerfassungssystem, die PL oder der Prozessor des Zynqs. Dabei muss beachtet werden, dass das Detektorsystem in Zukunft mit vier Timepix3-Chips mit jeweils acht Kanälen betrieben werden soll. Ein Lösungsansatz ist, den unverarbeiteten Datenstrom direkt an das Datenerfassungssystem zu senden. Dort kann der Datenstrom mit einem Algorithmus analysiert werden. Für diesen Lösungsansatz spricht, dass in Hochsprachen wie C++ oder Python ein Algorithmus einfacher zu implementieren ist als beispielsweise auf einem FPGA oder Mikroprozessorsystem. Dagegen spricht, dass das Abspeichern der Daten abhängig vom Datenerfassungssystem der Flaschenhals im Gesamtsystem ist [22, S. 2]. Darüber hinaus wird das Datenerfassungssystem zusätzlich belastet, wenn es 24 Datenströme gleichzeitig analysieren muss. Das kann dazu führen, dass Daten durch Überlastung der Ressourcen verloren gehen. Das Gleiche gilt für den Prozessor im Zynq, der mit der Analyse von gleichzeitig 24 Kanälen überlastet ist. In der PL hingegen können mehrere Kanäle parallel analysiert und bearbeitet werden. Der Code muss lediglich für einen Kanal entwickelt und danach für jeden weiteren Kanal instanziiert werden. Zusätzlich entlastet die Bearbeitung in der PL das Datenerfassungssystem und den Ethernet-Übertragungskanal. Obwohl die Realisierung eines Analyse- und Verarbeitungsalgorithmus in VHDL komplizierter als in einer Hochsprache ist, überwiegt der Vorteil der parallelen Verarbeitung. Aus diesen Gründen wird der Datenstrom in der PL verarbeitet. Folglich soll ein Modul den Datenstrom analysieren und

erkannte Pakete ausgeben. Das genaue Konzept der Paketerkennung wird im Abschnitt 4.4 diskutiert.

Aus dieser Entscheidung ergibt sich, dass die erkannten Pakete von der PL zum PS gesendet werden müssen, damit sie von dort über die Ethernetverbindung zum Datenerfassungssystem verschickt werden können. Ein Konzept für das Speichermanagement und die Übertragung der Pakete über Ethernet wird daher im Abschnitt 4.5 erarbeitet. Außerdem muss der Datenstrom von Timepix3, bevor er von der Paketerkennung analysiert werden kann, korrekt im Zynq aufgenommen werden. Da die Ausgangsdatenleitung von Timepix3 mit der PL verbunden ist, muss die Datenaufnahme in der PL realisiert werden.

Des Weiteren muss ein Konzept entwickelt werden, das den Übertragungsablauf von Timepix3 steuert und überwacht. Dabei muss mit allen Modulen, die Teil der Übertragung sind, kommuniziert werden. Zusätzlich muss die Systemkontrolle die Steuersignale von Timepix3 setzen. Warum es nützlich ist, die Systemkontrolle im PS sowie in der PL zu realisieren, wird nach dem genauen Konzept der zu steuernden Module im Abschnitt 4.6 erläutert.

Als letztes muss das Datenerfassungssystem die vom Zynq gesendeten Pakete entgegennehmen und abspeichern. Wie dies genau realisiert ist, wird im letzten Abschnitt dieses Kapitels erläutert. Aus den getroffenen Entscheidungen resultiert die Strukturierung des Systems, die in Abbildung 4.1 dargestellt ist. Im Folgenden werden die einzelnen Abschnitte genauer diskutiert.

4.2. Datenwiedergabe

Es müssen Daten korrekt an Timepix3 gesendet werden. Das Timepix3-Protokoll erwartet die Daten seriell mit einem Takt von 40 MHz. Dabei werden die Daten in Form von Paketen gesendet, die von 48 Bit bis 393264 Bit verschiedene Größen annehmen können. Die Herausforderung liegt darin, die unterschiedlich langen Pakete in einem seriellen Datenstrom zu formatieren und synchron mit dem Takt an Timepix3 zu senden. Dabei muss Timepix3 mit jeder steigenden Taktflanke ein Bit des Datenstromes erfassen.

Wie bereits im Gesamtkonzept entschieden, findet die Paketauswahl im PS statt. Eine weitere Teilaufgabe für das Senden von Daten an Timepix3 ist das Serialisieren der Pakete. Hier muss ein Paket mit einem Takt von 40 MHz serialisiert werden, sodass mit jedem Takt ein Bit des Paketes versendet werden kann. Findet das Serialisieren der Daten im PS statt, muss dies über ein hochpriorisierten Timerinterrupt geschehen. Demnach kann eine Interruptroutine mit einem Takt von 40 MHz aufgerufen werden und immer ein Bit eines Paketes ausgeben. Gegen diesen Ansatz spricht die Häufigkeit des Interrupts. Selbst bei einem Prozessortakt von beispielsweise 400 MHz muss der Interrupt alle 10 Takte auslösen. Da dies für den Aufruf sowie für die Abarbeitung eines Interrupts zu wenig ist, muss das Serialisieren der Pakete in der PL durchgeführt werden [23].

In der PL gibt es mehrere Möglichkeiten, ein Paket zu serialisieren. Eine davon ist, ein Schieberegister in VHDL umzusetzen. Das Schieberegister muss mit 40 MHz takten und serielle Daten synchron mit dem Takt des Schieberegisters ausgeben. Eine weitere Möglichkeit, Daten in der PL zu serialisieren, ist die Verwendung einer Primitive im IO-Block. Primitive liegen bereits in Hardware vor und müssen nicht im FPGA erzeugt werden. Dadurch sind

die Signallaufzeiten des Serialisierers im IO-Block optimiert, sodass der Ausgang synchron zum Takt verläuft. Der Nachteil ist hingegen, dass durch die fest integrierte Hardware die Benutzung der Primitive stark eingeschränkt ist. Aufgrund dessen kann der Serialisierer im IO-Block lediglich acht oder vier Bit serialisieren. Daraus folgt, dass für die Primitive ebenso ein VHDL-Modul implementiert werden muss, da die Pakete in mehrere 8-Bit-Vektoren formatiert werden müssen. Des Weiteren benötigt die Primitive zwei unterschiedliche Takte [24, S. 165]. Da die Takte synchron sein müssen, ist eine zusätzliche PLL als Taktteiler erforderlich. Das VHDL-Schieberegister hingegen benötigt nur einen Takt. Hinzu kommt, dass 40 MHz mit der gegebenen Architektur auch ohne Primitive synchron ausgegeben werden können, indem das letzte FlipFlop des Ausgangssignales im IO-Block platziert wird. Zusammenfassend wird sich für einen Serialisierer entschieden, der als Schieberegister mit einem Ausgangs-FlipFlop im IO-Block realisiert ist.

Wie in Abbildung 4.2 zu sehen ist, müssen Pakete vom PS zum Serialisierer gesendet werden. Für die Kommunikation zwischen PS und PL gibt es generell zwei Möglichkeiten. Der

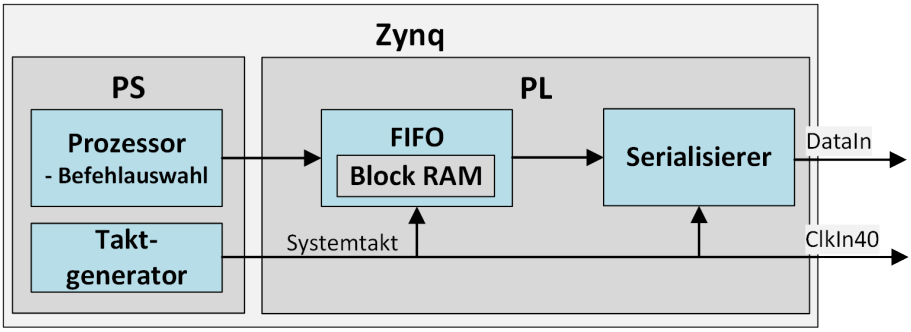

Abbildung 4.2.: Datenwiedergabe Konzept

DMA-Controller bietet eine sehr schnelle und für den Prozessor weniger aufwendige Kommunikationsmöglichkeit. Er ist jedoch aufwändiger zu realisieren und benötigt mehr Ressourcen als ein FIFO. Dieser ist allerdings langsamer in der Übertragung, da die Daten einzeln vom Prozessor in den FIFO geschrieben werden müssen. Jedoch erfordern die Eingangsdaten von Timepix3 keine schnelle Übertragung, weswegen sich für einen FIFO entschieden wird.

4.3. Datenaufnahme

Der Zynq muss die gesendeten Daten von Timepix3 korrekt empfangen. Die Daten werden synchron über acht Datenleitungen und einer Taktleitung von Timepix3 gesendet. Über Timepix3-Register ist es möglich, die Taktgeschwindigkeit sowie die Anzahl der aktivierten Datenausgänge einzustellen. Wie in den Anforderung dargestellt, soll Timepix3 mit einem aktiven Datenausgang und einer Taktgeschwindigkeit von 160 MHz betrieben werden. Somit werden durch das DDR-Format 320 MBit/s übertragen. Des Weiteren ist es möglich, in Timepix3 einzustellen, ob die Daten mit oder ohne 8B/10B-Codierung ausgegeben werden sollen. Die Herausforderung liegt darin, die hohe Datenrate mit DDR-Übertragungsverfahren

sicher in der PL aufzunehmen und zur weiteren Verarbeitung zur Verfügung zu stellen. Dabei muss geprüft werden, ob die Implementierung der 8B/10B-Codierung nützlich ist.

Erstrebenswert ist es, den Ausgangsdatenstrom im 8B/10B-Format zu betreiben, da dies verschiedene Vorteile in der Übertragung hat (vgl. Kapitel 2.2.1). In der Ultrascale+ Architektur ist es allerdings nur möglich, mit GTH- oder GTY-Transceivern ein Signal im 8B/10B-Protokoll zu empfangen [25][26]. Das liegt zum einen daran, dass aus dem Datensignal selbst der Takt zurückgewonnen werden muss und zum anderen, dass die Daten zusätzlich im DDR-Format übertragen werden. Da die Transceiver sehr aufwändig zu realisieren sind, werden diese nicht in jedem Zynq fest integriert. Der Zynq Ultrascale+ ZU3CG besitzt keinen dieser Transceiver, wodurch keine 8B/10B-Dekodierung und -Aufnahme im IO-Block möglich ist [11, S. 4].

Daher muss eine andere Lösung, die den seriellen DDR-Datenstrom aufnimmt und deserialisiert, für die Datenaufnahme konzipiert werden. Es ist zu beachten, dass der FPGA immer im Single Data Rate (SDR)-Format arbeitet, da dieser keine FlipFlops mit der Triggerung auf fallender und gleichzeitig steigender Taktflanke besitzt. Folglich muss das DDR-Format in ein SDR-Format umgewandelt werden. Dazu muss der DDR-Datenstrom zunächst abgetastet werden. Diesen Vorgang kann ein ISERDESE3 übernehmen, der sich im IO-Block befindet und den Datenstrom aufnimmt und deserialisiert [24, S. 161]. Ein anderer Lösungsweg ist, die Daten mit einem VHDL-Modul im FPGA abzutasten.

Die Abtastung im FPGA hat den Nachteil, dass das Signal von zwei FlipFlops parallel aufgenommen werden muss. Dabei tastet ein FlipFlop auf steigender und ein FlipFlop auf fallender Flanke den Datenstrom ab. Demnach muss eine Logik, die mit 320 MHz getaktet ist, die Ausgänge der FlipFlops auswerten und den DDR-Datenstrom deserialisieren. Da der Signaltakt asynchron zum Systemtakt ist, müssen diese an einer Stelle im FPGA getrennt werden.

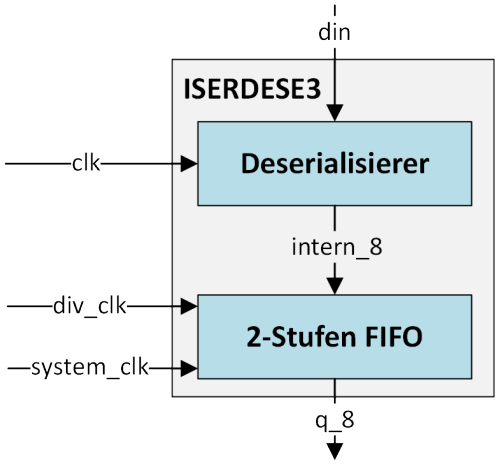

Abbildung 4.3.: ISERDESE3 vereinfachtes Blockschaltbild

Der ISERDESE3 im IO-Block aus Abbildung 4.3 hat gegenüber der Implementierung im FPGA einige Vorteile. Er benötigt dafür drei Takte [27]. Der erste Takt “clk“ ist der Ausgangssignaltakt von Timepix3 mit 160 MHz. Mit diesen werden die Daten “din“ im

DDR-Format intern deserialisiert. Ein zweiter Takt “div_clk“, der synchron zum Signaltakt verläuft, allerdings nur mit 40 MHz taktet, schreibt die deserialisierten Daten in einen internen zweistufigen FIFO. Demzufolge werden mit jedem Takt von “div_clk“ acht Bit aus dem Deserialisierer gelesen und in einen zweistufigen FIFO geschrieben. Mit dem dritten Takt “system_clk“ kann mit 40 MHz im SDR-Format der 8-Bit-Vektor “q_8“ aus dem FIFO gelesen und im FPGA verarbeitet werden. Abbildung 4.4 zeigt in einem Zeitablaufdiagramm, wie die Datenerfassung vom ISERDESE3 durchgeführt wird. Die Primitive gibt das

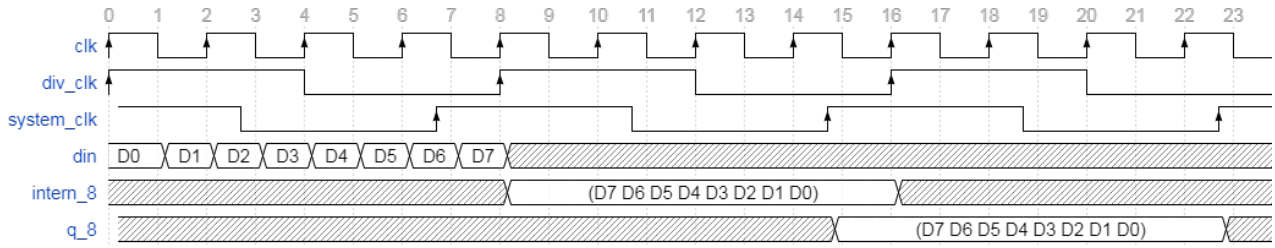

Abbildung 4.4.: ISERDESE3 Zeitablaufdiagramm
Dieses Diagramm ignoriert zur Veranschaulichung die Tiefe des internen FIFOs

zuerst empfangene Bit als LSB von “q_8“ aus. Da Timepix3 die Pakete LSB-First sendet, ist die Anordnung der Bits von “q_8“ automatisch korrekt. Zusätzlich hat die Verwendung der Primitive den großen Vorteil, dass die Daten im SDR-Format ausgegeben werden und die Takttrennung bereits im IO-Block erfolgt. Somit muss keine Takttrennung im FPGA erfolgen. Ein weiterer Vorteil ist, der sehr niedrige Systemtakt mit dem die Daten aus der Primitive gelesen werden können. Hinzu kommt, dass die gleiche Taktdomäne wie bei der Datenwiedergabe verwendet werden kann, da beide Module mit 40 MHz takten. Folglich sind keine zwei Systemtakte in der PL notwendig. Der einzige Nachteil ist, dass der Deserialisierer eine zusätzliche PLL benötigt, die den Signaltakt durch vier teilt und diesen synchron mit dem Signaltakt selbst ausgibt. Dadurch, dass in der PL mehrere PLLs existieren, ist dies kein wesentlicher Nachteil, weshalb sich für den ISERDESE3 im IO-Block entschieden wird.

Jetzt muss entschieden werden, ob die 8B/10B-Codierung in Timepix3 aktiviert wird. Für die 8B/10B-Codierung spricht eine sicherere Übertragung sowie die Verwendung von zusätzlichen Steuer- und Synchronisationszeichen, die im 10B-Format zusätzlich zu den Daten zur Verfügung stehen. Allerdings gibt der ISERDESE3 die Daten in 8-Bit-Vektoren und nicht in 10-Bit-Vektoren aus, was die Implementierung einer zusätzlichen Logik erfordert. In der Logik müssen mehrere 8-Bit-Vektoren zu 10-Bit-Vektoren zusammengefügt werden. Anschließend muss eine Logik die 10-Bit-Vektoren dekodieren. Dafür ist ein weiterer Systemtakt nötig (vgl. Abbildung 4.5), da die Periode eines 10-Bit-Vektors länger ist, als die eines 8-Bit-Vektors, welcher mit 40 MHz aus dem ISERDESE3 ausgelesen wird. Dieser Implementierungsaufwand wird als zu groß eingeschätzt, weswegen die 8B/10B-Codierung deaktiviert wird. Das hat Auswirkungen auf die Erkennung von Paketen im Datenstrom, da folglich die 10-Bit-Steuerzeichen auf 8-Bit-Datenzeichen abgebildet werden. Timepix3 sendet somit im Leerlauf immer das Signal “0xBC“. Welche Probleme sich daraus ergeben, wird im Abschnitt 4.4 erläutert.

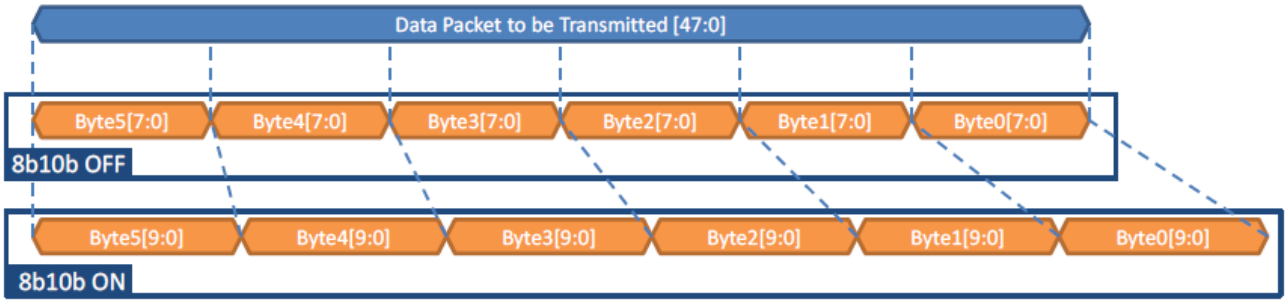

Abbildung 4.5.: 8B/10B-Codierung eines Paketes [2, S. 55]

Der ISERDESE3 stellt nicht sicher, dass die Bytes im korrekten Raster ausgegeben werden. Abbildung 4.6 veranschaulicht an einem Beispiel, zu welchen Problemen das führen kann, wenn der ISERDESE3 zwei Bytes empfängt. Hier ist das Aufnahmeraster des Deseria-

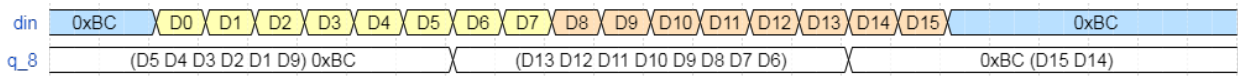

Abbildung 4.6.: ISERDESE3 um zwei Bit verschobener Ausgang

lisierers um zwei Bit verschoben. Folglich werden zwei Bytes auf drei Bytes abgebildet. Das erschwert die nachfolgende Verarbeitung in der PL, da der Ausgang des Deserialisierers nicht direkt einem gesendeten Byte von Timepix3 zugeordnet werden kann. Timepix3 sendet im Leerlauf das Synchronisationssignal “0xBC“. Aufgrund dessen kann der Deserialisierer, wenn Timepix3 sich im Leerlauf befindet, auf das Synchronisationssignal angepasst werden. Diese Aufgabe kann ein VHDL-Automat übernehmen, der den Ausgang vom ISERDESE3 so lange verschiebt, bis das Ausgangssignal der Synchronisationssequenz entspricht. Da es sich hierbei um eine bekannte Problematik handelt, bietet die Firma Xilinx ein BitSlip-VHDL-Modul für den Deserialisierer an, welches die Anpassung übernimmt [28]. Das Modul aus Abbildung 4.7 nimmt als Eingang den Ausgang vom ISERDESE3 entgegen und gibt diesen wieder aus. Allerdings kann über ein Pulseingang mit jedem Puls der Ausgang “daten_8“ um ein Bit

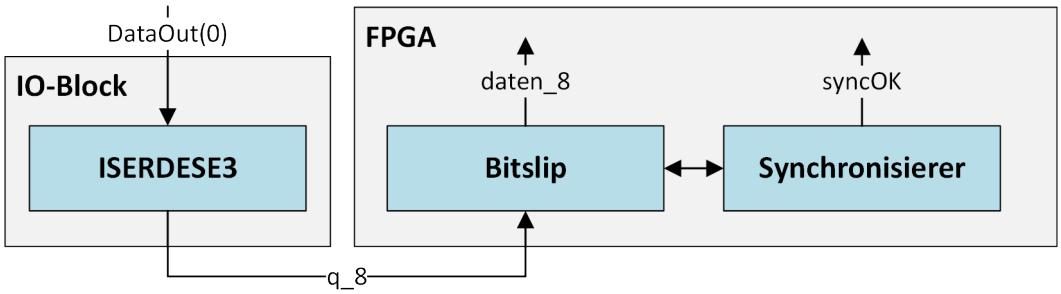

Abbildung 4.7.: Datenaufnahme Konzept

verschoben werden, bis der Ausgang der Synchronisationssequenz entspricht. Somit startet

ein Synchronisierer, der als VHDL-Automat realisiert ist, eine Synchronisation. Zusätzlich kann der Automat ein Signal senden, wenn der Vorgang erfolgreich abgeschlossen wurde.

Demzufolge gibt die Lösung zur Aufnahme der Daten von Timepix3 in der PL mit jeder Systemtaktflanke ein Byte aus. Mit der Zusammensetzung der Pakete aus dem acht Bit breiten Datenstrom wird sich im nächsten Abschnitt beschäftigt.

4.4. Paketerkennung

Der acht Bit breite Datenstrom der Datenaufnahme muss von der Paketerkennung in Timepix3-Pakete formatiert werden. Dazu müssen Synchronisationssequenzen im Datenstrom erkannt

Abbildung 4.8.: Paketerkennung Modul

und herausgefiltert werden. Abbildung 4.8 zeigt dabei die Grundidee der Paketerkennung.

Problemstellung

Ausgangspakete von Timepix3 sind immer 48 Bit breit und haben einen 4-Bit oder 8-Bit breiten Header, um das Paket zu identifizieren. Timepix3 sendet im Leerlauf und nach spätestens acht Timepix3-Paketen zur Synchronisation im 8B/10B-Protokoll das zehn Bit breite Zeichen K.28.5. Da das 8B/10B-Protokoll deaktiviert ist, werden 10-Bit-Steuerzeichen auf 8-Bit-Datenzeichen abgebildet. Das Zeichen K.28.5 wird im 8-Bit-Format als “0xBC“ dargestellt. Folglich sendet Timepix3 im Leerlauf stetig das Signal “0xBC“. In Timepix3-Paketen werden keine Synchronisationssequenzen gesendet. Jedoch sendet Timepix3 während einer längeren Aufnahme zwischen Paketen immer wieder das Synchronisationszeichen. Daher müssen die Pakete von dem Synchronisationszeichen getrennt werden. Die Herausforderung liegt darin, dass anders als im 8B/10B-Protokoll das Synchronisationszeichen “0xBC“ auch in den Daten vorkommen kann. Aus diesem Grund darf nicht automatisch jedes “0xBC“ eliminiert werden, da sonst auch Pakete verfälscht werden können. Dazu kommt, dass Timepix3 die Pakete LSB-First sendet. Demnach werden die acht Bit, die den Header enthalten, als letztes gesendet. Das erschwert die Analyse des Datenstromes, da zuerst Daten, die gleich “0xBC“ sein können, gesendet werden. Um die Problemstellung zu veranschaulichen, werden in Tabelle 4.1 Bezeichnungen und Abkürzungen für die verschiedenen Ereignisse im Datenstrom dargestellt.

Abkürzung	Beschreibung
I	Synchronisationszeichen
ID	Daten, die aussehen wie das Synchronisationszeichen
H	Header eines Paketes
HD	Daten, die aussehen wie der Header eines Paketes
D	Daten, die ungleich HD oder ID sind

Tabelle 4.1.: Bezeichnungen und Abkürzungen der 8-Bit-Daten

Abbildung 4.9 zeigt ein von Timepix3 gesendetes Paket. Dabei wird als letztes der Header des Paketes gesendet. Sobald die Daten ungleich I werden (3. Takt in Abbildung 4.9),

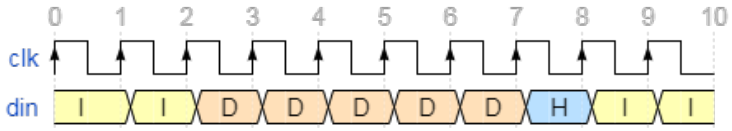

Abbildung 4.9.: Einzelnes Timepix3-Paket mit einfacher Erkennung

erscheint nach genau fünf Takten der Header des Paketes. Demzufolge kann dieses Paket mit einer einfachen Logik erkannt werden. Wie Abbildung 4.10 zeigt, wird es schwieriger bei Paketen mit einem ID am Ende des Paketes, da dieses vorerst nicht von I unterschieden werden kann. In diesem Fall befindet sich nach fünf Takten kein Header, nachdem die Daten ungleich I oder ID werden. Ein mögliches Auftreten von mehreren IDs am Ende des Pake-

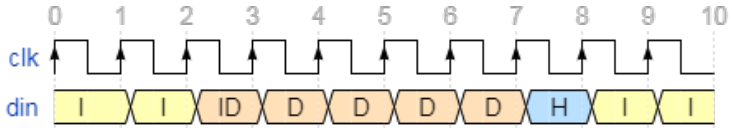

Abbildung 4.10.: Einzelnes Timepix3-Paket mit komplexer Erkennung

tes, erschwert die Paketerkennung zusätzlich. Die Schwierigkeit der Paketerkennung nimmt ebenfalls zu, wenn beispielsweise zwei Pakete wie in Abbildung 4.11 aneinanderliegend gesendet werden. Für eine mögliche fehlerhafte Erkennung muss das erste Paket mit ID und das zweite Paket mit HD enden. Infolgedessen kann fälschlicher Weise angenommen werden,

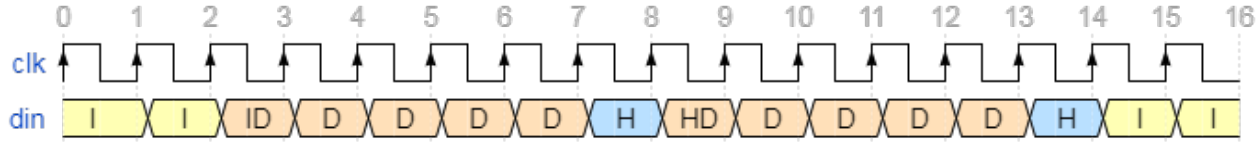

Abbildung 4.11.: Zwei Timepix3-Pakete mit komplexer Erkennung

dass HD ein richtiger Header ist, da das ID im 3. Takt auch als I angenommen werden kann.

Die Datenströme können weitere komplexere Formen annehmen, welche in diesem Abschnitt nicht weiter erläutert werden. Fest steht jedoch, dass es lediglich zu Problemen in der Paketerkennung kommen kann, wenn sich ein ID am Ende eines Paketes befindet. Wird von einer Gleichverteilung der Daten ausgegangen, beinhaltet eines von 256 (2^8) Paketen am Ende ein ID. Daher wird eine Logik, die ID's und HD's ignoriert, mindestens eins von 256 Paketen gar nicht, oder falsch erkennen. Jedoch genügen 255 von 256 erkannte Pakete, um erste Tests mit Timepix3 durchzuführen. Um korrekte Pakete zu erhalten, gilt es dennoch eine Paketerkennung zu konzipieren, die in der Lage ist, alle Pakete korrekt zu erkennen.

Grundkonzept der Paketerkennung

Im Folgenden wird diskutiert, wie der Datenstrom am besten analysiert werden kann. Xilinx oder andere Firmen bieten keine bestehende Firmware für die Lösung eines solchen Problems an. Aus diesem Grund muss ein Automat konzipiert werden, der die Pakete im Datenstrom erkennt. Durch die Komplexität des Problems gibt es viele Lösungsansätze, die alle Vor- und Nachteile aufweisen.

Zuerst muss entschieden werden, ob der Datenstrom von LSB nach MSB oder umgekehrt analysiert werden soll. Wie in Abbildung 4.12 zu sehen ist, entspricht die Analyse von LSB nach MSB der Reihenfolge, wie die Daten aus der Datenaufnahme gesendet werden. Den

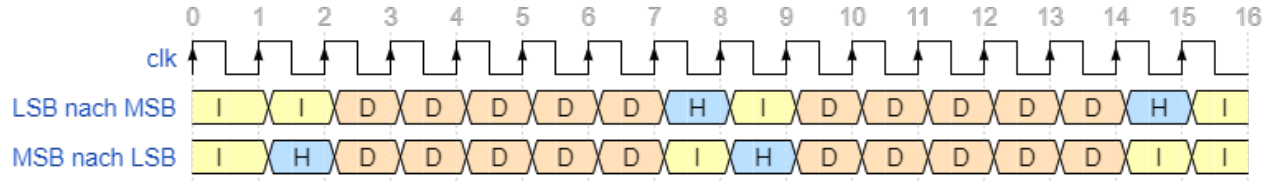

Abbildung 4.12.: Richtung der Datenstromanalyse

Datenstrom von LSB nach MSB einzulesen hat demnach den Vorteil, dass wenig oder gar kein Speicher beansprucht werden muss, da der Datenstrom direkt analysiert werden kann. Es hat jedoch den Nachteil, dass das LSB eines Paketes zuerst eingelesen wird, was dazu führt, dass eine Logik die IDs am Ende eines Paketes erkennen muss. Wird der Datenstrom jedoch von MSB nach LSB analysiert, muss dieser erst zwischengespeichert werden, da die Daten, die zuerst ausgegeben werden, als letztes analysiert werden müssen. Allerdings hat dieser Ansatz den Vorteil, dass beim Einlesen des Datenstromes zuerst die Header der Pakete eingelesen werden. Demzufolge können IDs am Ende eines Paketes ignoriert werden, was das Erkennen der Pakete erleichtert. Jedoch ist der Speicherbereich auf dem Ausleseboard auf 1 GB begrenzt. Wie Formel 4.1 zeigt, kann daher bei 32 aktiven Ausgangskanälen mit maximaler Datenrate der Speicher nach einer halben Sekunde überlaufen.

$$t = \frac{Speicher}{Ausgänge \cdot Datenrate} = \frac{1\ GByte}{32 \cdot 640\ MBit/s} = 0.4\ s \tag{4.1}$$

Demnach dürfen nur Teile der Aufnahme zwischengespeichert werden, welche beispielsweise über Double Buffering abwechselnd analysiert und eingelesen werden. Folglich muss der

Datenstrom an einer bestimmten Steller zur Analyse unterbrochen werden. Dadurch entsteht jedoch das gleiche Problem wie bei der Analyse des Datenstromes von LSB nach MSB. Aus diesem Grund muss wieder auf IDs geachtet werden, um den Datenstrom nicht mitten in einem Paket zu trennen. Ein weiterer Nachteil ist, dass die erkannten Pakete nach der Erkennung in der falschen Reihenfolge vorliegen. Dies kann zu extra Dekodierungsarbeit im Datenerfassungssystem führen. Daher wird sich dafür entschieden, den Datenstrom von LSB nach MSB zu analysieren und mit einem VHDL-Modul die IDs am Ende der Pakete zu detektieren.

Um die Pakete im Datenstrom zu erkennen, gibt es verschiedene Ansätze. Zum einen kann eine Art Kreuzkorrelation konzipiert werden, die eine Paketform mit dem Datenstrom vergleicht. Zum anderen ist eine Lösung mit einem Zähler möglich, der zu den angenommenen Positionen der Header zählt. Hier ist allerdings noch eine Fehlerkorrektur nötig, die eingreift, wenn sich an der Position kein Header befindet. In Abbildung 4.13 wird dieser Lösungsansatz in Form eines Automatengraphen dargestellt. Im Zustand “Leerlauf“ wartet der Automat,

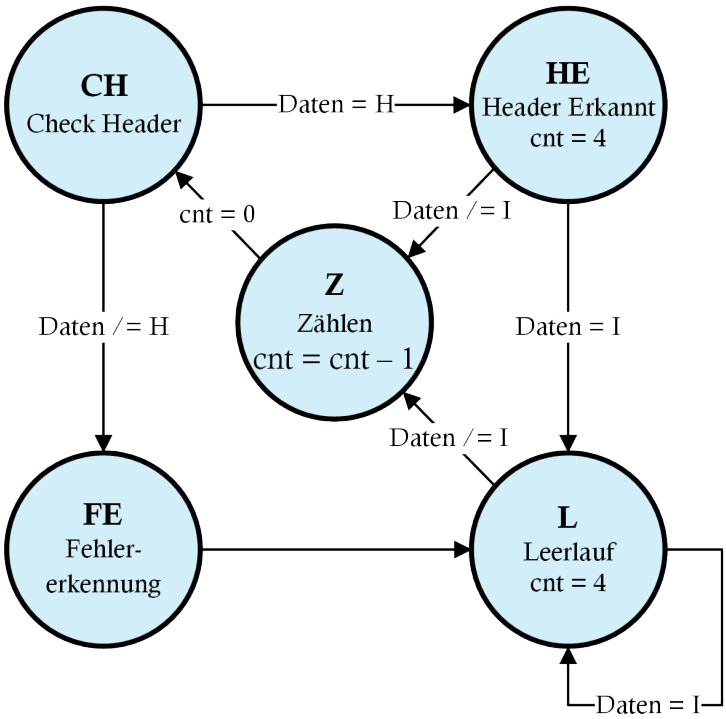

Abbildung 4.13.: Lösungsansatz Zähler

bis die Daten ungleich I oder ID sind. Danach zählt der Automat im Zustand “Zählen“ vier Takte bis zum Header und springt im fünften Takt in den Zustand “Check Header“. Befindet sich in diesem Takt ein Header im Datenstrom, ist ein Paket erkannt. Ist dies nicht der Fall, muss im Zustand “Fehlererkennung“ der Fehler korrigiert werden. Ein weiterer Lösungsansatz ist die Nutzung von bereits existierenden Mustererkennungsalgorithmen. Allerdings ist die Überführung in VHDL sowie die Anpassung des Algorithmus’ an das Problem und die dadurch verbundene Einarbeitung in das Thema mit großem Zeitaufwand verbunden. Aus

diesen Gründen wird dieser Lösungsansatz nicht gewählt. Bei einer Kreuzkorrelation handelt es sich ebenso um eine Art Mustererkennung. Da durch HDs und IDs im Datenstrom mehrere mögliche Pakete entstehen können, benötigt eine Kreuzkorrelation genauso wie der Zählautomat eine zusätzliche Logik, die den Ausgang der Korrelation auswertet. Wird von einer Gleichverteilung der Daten ausgegangen, erkennt der Zählautomat auch ohne Fehlererkennung 255 von 256 Pakete. Da mit dieser Fehlerquote bereits erste Tests mit Timepix3 durchgeführt werden können, wird sich für den Zählautomaten entschieden.

Um jedes Paket zu erkennen, soll dennoch eine Fehlererkennung konzipiert werden. Dafür wird vorerst der Fehlerfall im Zählautomat genauer betrachtet. Wie in Abbildung 4.14 zu sehen ist, muss für eine korrekte Fehlererkennung der Datenstrom zusammenhängend analysiert werden. Andernfalls können Pakete nicht sicher im Datenstrom erkannt werden.

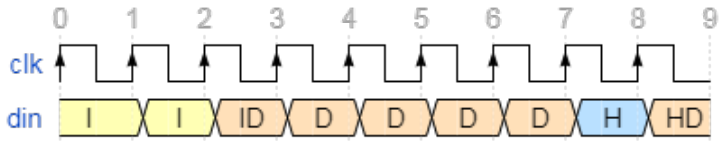

Abbildung 4.14.: Paketerkennung nicht möglich

Alle fehlerhaften Erkennungen des Automaten haben gemeinsam, dass mindestens in einem der vorangegangenen Pakete mindestens ein ID am Ende vorgekommen ist. Wird solch ein Paket nicht empfangen, kann es über den Zählautomaten nicht zur fehlerhaften Erkennung kommen, da sonst im Zustand “Check Header“ immer ein Header vorliegt. Liegt dem Automaten im Zustand “Check Header“ allerdings kein Header vor, bedeutet dies, dass eines der vorangegangenen Pakete mindestens ein ID am Ende beinhalten muss.

Wie Abbildung 4.15 zeigt, ist es bei der Fehleranalyse außerdem wichtig, wie viele Pakete zuvor zusammenhängend empfangen werden. Im ersten Beispiel wird ein Paket mit einem

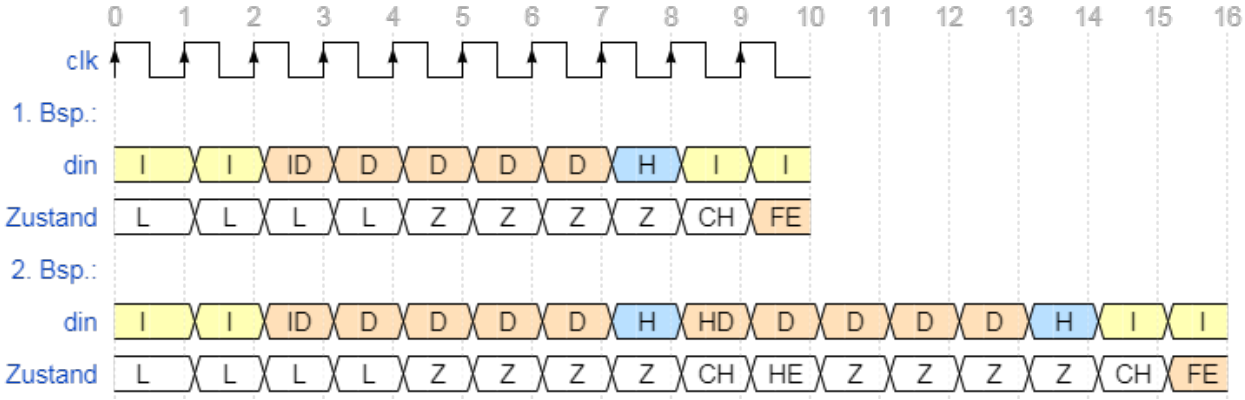

Abbildung 4.15.: Fehlerfall Zählautomat einfach

ID am Ende empfangen. Da der Fehlerfall nach einem Paket auftritt, muss sich am Ende des Paketes mindestens ein ID befinden. Im zweiten Beispiel werden zwei Pakete direkt hintereinander empfangen. Der Automat erkennt das erste Paket durch das HD falsch und

bemerkt den Fehler erst nachdem zweiten Paket. In diesem Fehlerfall muss das erste Paket mindestens ein ID am Ende beinhalten.

Wie Abbildung 4.16 zeigt, kann es sich mit geringerer Wahrscheinlichkeit auch um ein komplexeres Problem handeln. Dort erkennt der Automat zuerst den falschen Header HD

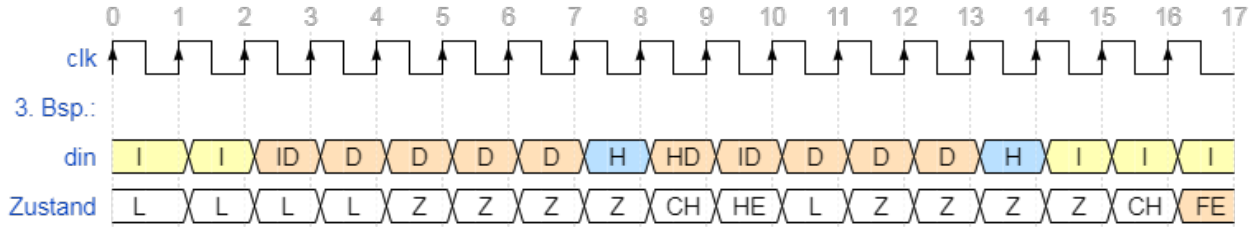

Abbildung 4.16.: Fehlerfall Zählautomat komplex

und nimmt durch das darauffolgende ID an, dass eine Leerlaufsequenz gesendet wird. Der Automat springt daher einen Takt später in den Zustand “Zählen“. Folglich wird der Fehlerfall einen Takt später festgestellt. Die Komplexität mit jedem weiteren zusammenhängenden Paket nimmt zu. Jedoch nimmt die Wahrscheinlichkeit, dass so ein Fehlerfall auftritt gleichzeitig ab, da sich immer mehr ID’s oder HD’s an bestimmten Stellen im Datenstrom befinden müssen.

Für die Fehlererkennung gibt es viele Lösungsansätze. Im Rahmen dieser Arbeit werden lediglich zwei von ihnen vorgestellt. Zum einen kann der Datenstrom bei einem Fehler, abhängig von der Anzahl der zusammenhängenden Pakete, mit Vergleichspaketen verglichen werden. Zum anderen kann bei einem Fehler der fehlerhafte Datenstrom separat abgespeichert werden und anschließend von einem anderen Automaten von MSB nach LSB analysiert werden, um die Header zu identifizieren. In Abbildung 4.17 wird der Lösungsansatz mit Vergleichspaketen für ein falsch erkanntes Paket dargestellt. Dort muss der Datenstrom mit zwei

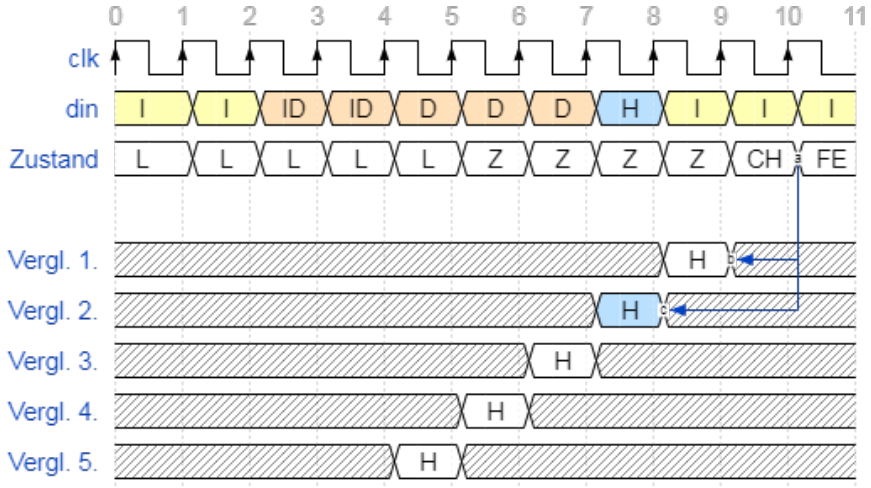

Abbildung 4.17.: Vergleichspakete

von fünf möglichen Paketen verglichen werden, bis das korrekte Paket erkannt wird. Falls

durch HDs mehrere mögliche Pakete existieren, wird das Vergleichspaket mit der höchsten Wahrscheinlichkeit gewählt. Für diesen Lösungsansatz spricht, dass die Fehler sofort erkannt werden und somit kein zusätzlicher Speicher nötig ist. Ein Nachteil ist allerdings, dass es bei Fehlern mit vielen zusammenhängenden Paketen, zu einer großen Anzahl an möglichen Vergleichspaketen kommt. Beim zweiten Lösungsansatz muss der Teil des Datenstromes, der die falsch erkannten Pakete enthält, abgespeichert werden und von MSB nach LSB eingelesen werden. Das hat den Vorteil, dass der Datenstrom beim Fehlerfall sofort analysiert werden kann und nicht mit Vergleichspaketen verglichen werden muss. Allerdings kann es bei diesem Lösungsansatz dazu kommen, dass Pakete durch falsches Abspeichern aufgetrennt und somit beschädigt werden. In Abbildung 4.18 ist dieser Fall genauer dargestellt. Hier wird

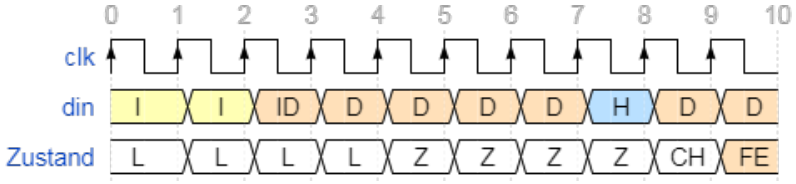

Abbildung 4.18.: Fehler bei Datenstromauftrennung

der Fehler in einem neuen Paket erkannt. Aufgrund dessen muss der Datenstrom in einem Paket aufgetrennt werden, was zu Fehlern führen kann. Des Weiteren ist bei diesem Lösungsansatz zusätzlicher Speicher nötig. Aus diesen Gründen wird sich für die Lösung mit Vergleichspaketen entschieden.

Die Fehlererkennung benötigt durchgehend Zugriff auf den Datenstrom, um diesen im Fehlerfall mit Vergleichspaketen zu vergleichen. Daher muss ein Konzept entwickelt werden, das den Datenstrom abspeichert und diesen der Fehlererkennung zur Verfügung stellt. Speichermöglichkeiten, die direkt mit externen oder internen RAM des Zynqs kommunizieren, haben den Nachteil, dass die Daten erst ein bis zwei Takte nach der Anfrage zur Verfügung stehen [29, S. 44]. Demnach ist es schwierig eine schnelle Fehlererkennung mit RAM zu realisieren. Eine weitere Möglichkeit ist, den Speicher im FPGA durch FlipFlops zu realisieren. FlipFlops im FPGA haben den Vorteil, dass diese direkt abgefragt werden können und somit eine schnelle Fehlererkennung ermöglichen. Allerdings beansprucht dieser Lösungsansatz viele Ressourcen. Wie Formel 4.2 zeigt, sind beispielsweise 240 FlipFlops für die Speicherung von fünf rückwirkend betrachteten Paketen nötig.

$$FlipFlops = Paketanzahl \cdot Paketbreite = 5 \cdot 48 = 240 \tag{4.2}$$

Jedoch vergleicht die Fehlererkennung im Fehlerfall den Datenstrom nur mit Headern. Die Fehlererkennung benötigt daher lediglich die Information, ob ein Byte des Datenstroms gleich H oder HD ist. Somit ist der Informationsgehalt, auf welchen direkt zugegriffen werden muss, anstatt acht Bit ein Bit breit. Aus diesem Grund werden hier beide Lösungsansätze kombiniert. Folglich wird der Datenstrom im RAM abgespeichert und die Information, ob es sich bei den Daten um einen möglichen Header handelt, wird in FlipFlops abgespeichert. Dieser Lösungsansatz verringert die benötigten FlipFlops für fünf Pakete von 240 auf 30.

Des Weiteren muss ein Verfahren ausgearbeitet werden, das die Pakete im Datenstrom als erkannt markiert und anschließend ausgibt. Dafür kann der Speicher mit Ringbuffern oder Schieberegistern realisiert werden. Abbildung 4.19 zeigt, wie die Realisierung mit Schieberegistern erfolgen kann. Dafür wird der komplette Speicher im Modul mit Schieberegistern realisiert. Somit läuft das Schieberegister, das den Datenstrom speichert, parallel zum Schieberegister, das mögliche Header speichert. Ein zusätzliches Schieberegister, das ebenfalls

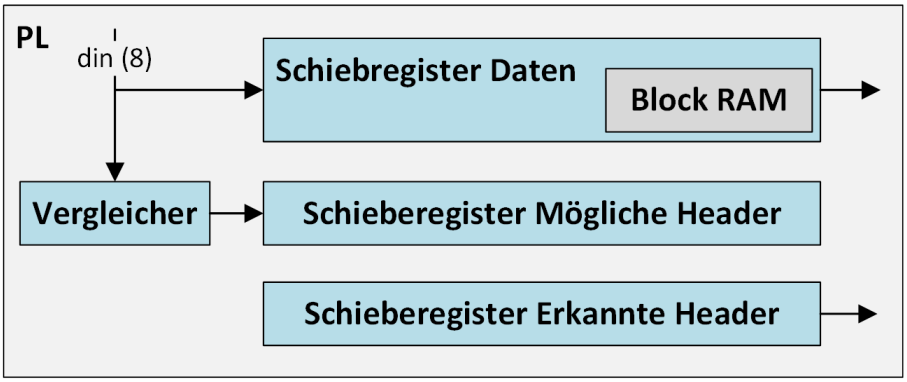

Abbildung 4.19.: Speicherung mit Schieberegistern

parallel zum Datenstrom verläuft, kann auf diese Weise korrekt erkannte Header anzeigen. Es muss allerdings möglich sein, dieses parallel zu überschreiben, um falsch erkannte Header zu korrigieren. Durch die Parallelität der Register entsprechen die letzten sechs Byte des Daten Schieberegisters einem Paket, wenn das "Erkannte Header"-Schieberegister eine Eins ausgibt. Ein weiterer Lösungsansatz ist, den Speicher als Ringbuffer zu realisieren. Dabei können Register Indices abspeichern, an welchen sich mögliche Header im Ringbuffer befinden. Beide Lösungsansätze sind ähnlich und gleich gut. Allerdings bietet Xilinx bereits einen IP-Core für ein Schieberegister an, der mit dem internen Block-RAM des Zynqs kommuniziert [30]. Daher ist das Schieberegister einfacher zu implementieren, weshalb sich gegen den Ringbuffer und für die Realisierung mit Schieberegistern entschieden wird.

Als letztes muss festgelegt werden, wie lange der Datenstrom rückwirkend analysiert werden muss, bevor ein Paket als Erkannt markiert wird. Ein Fehler kann sich theoretisch bis in das Unendliche ausbreiten, wenn immer wieder Pakete mit einem HD am Ende auftreten. Daher muss eine sinnvolle Anzahl an zu analysierenden Paketen gewählt werden. Jedoch ist das Ziel, alle Pakete zu erkennen, mit den vorgestellten Lösungsansätzen gar nicht bzw. nur sehr schwierig möglich. Das hängt damit zusammen, dass der Datenstrom immer zusammenhängend analysiert werden muss, der FPGA jedoch begrenzte Ressourcen besitzt. Zudem ist es durch die vielen möglichen Fehlerarten schwierig, die Fehlerrate der Paketerkennung mathematisch zu bestimmen. Daher muss, eine Simulation entwickelt werden, die einen Datenstrom von Timepix3 möglichst genau simuliert und die Fehlerrate von der Paketerkennung bestimmt. Zusätzlich soll es in der Paketerkennung möglich sein, über Generics die Länge des zu analysierenden Datenstromes einzustellen. Auf diesem Weg kann die Fehlerrate der Paketerkennung in Abhängigkeit von der Länge des abgespeicherten Datenstromes simuliert werden. Ein weiterer Vorteil ist, dass benötigte Ressourcen im FPGA mit der Feh-

lerrate verglichen werden können. Somit kann nach der Implementation im Kapitel 6 eine sinnvolle Länge des abgespeicherten Datenstromes bestimmt werden.

4.5. Speichermanagement und Ethernet

Die von der Paketerkennung erkannten Pakete müssen über Ethernet zum Datenerfassungssystem übertragen und dort abgespeichert werden. Dabei werden die Pakete mit einem "valid"-Signal in der PL ausgegeben. Durch die gegebene Hardware ist die Ethernetkommunikation nur über das PS möglich. Daher müssen die Pakete vom PL zum PS gesendet werden, bevor diese über Ethernet verschickt werden können. Um auf der Kommunikationsstrecke zwischen PL und Datenerfassungssystem keine Pakete durch Speicherüberlastung zu verlieren, muss ein Speichermanagement konzipiert werden.

Dafür wird zuerst die Kommunikationsschnittstelle zwischen PL und PS festgelegt. Zur Auswahl stehen ein FIFO und ein DMA-Controller. Mit dem DMA-Controller ist der Prozessor in der Lage, die Daten schneller zu verarbeiten, da komplette Speicherbereiche mit Hilfe von Adressen aus dem Speicher ausgelesen werden können. Die Adressen können anschließend dem Ethernet-Controller übergeben werden, damit dieser den Speicherbereich zum Datenerfassungssystem senden kann. Demzufolge werden weniger Rechenressourcen vom Prozessor beansprucht und es können größere Datenmengen schneller als mit einem FIFO zwischen PL und PS übertragen werden. Da die Daten von Timepix3 mit einem aktiven Kanal mit 320 MBit/s empfangen werden, ist eine hohe Übertragungsgeschwindigkeit zwingend notwendig. Aus diesem Grund wird trotz des größeren Ressourcen- und Implementierungsaufwands die Kommunikationsschnittstelle mit einem DMA-Controller realisiert.

Die Datenbreite des DMA-Controllers beträgt 32 Bit. Da 48-Bit-Pakete empfangen werden, muss eine Lösung konzipiert werden, welche die Pakete in 32-Bit-Vektoren verpackt. Dabei gibt es zwei sinnvolle Ansätze. Das obere Beispiel aus Abbildung 4.20 zeigt, dass zwei 32-

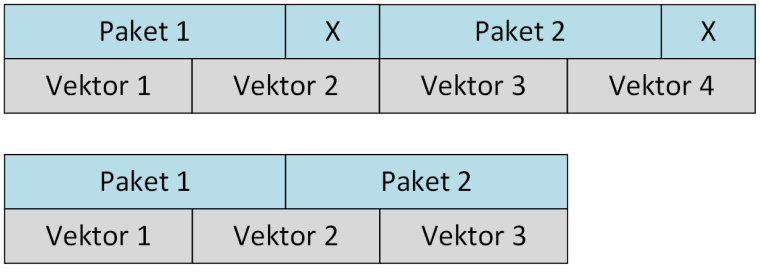

Abbildung 4.20.: Mögliche Anordnung von Paketen im Speicher

Bit-Vektoren immer genau ein 48-Bit-Paket darstellen. Das untere Beispiel zeigt, dass immer zwei Pakete in drei 32-Bit-Vektoren gespeichert werden. Für das obere Beispiel spricht, dass es für das Datenerfassungssystem einfacher ist, einen 64-Bit-Datentyp direkt zu analysieren. Das zweite Beispiel hingegen ist schwieriger zu analysieren, da keine standardisierten 48-Bit-Datentypen auf dem Datenerfassungssystem existieren. Um ein Paket zu erfassen, sind daher mehr Rechenschritte als beim ersten Beispiel nötig. Jedoch nutzt das untere Beispiel

den kompletten Speicher, während das obere Beispiel lediglich 75% des Speichers nutzt. Dadurch gehen 25% der Übertragungsbandbreite verloren, weswegen sich für das untere Beispiel entschieden wird. Wie Abbildung 4.21 zeigt, wird die Formatierung von 48 auf 32 Bit von einem "Paket zu Stream"-Automaten durchgeführt. Der Automat muss dem DMA-

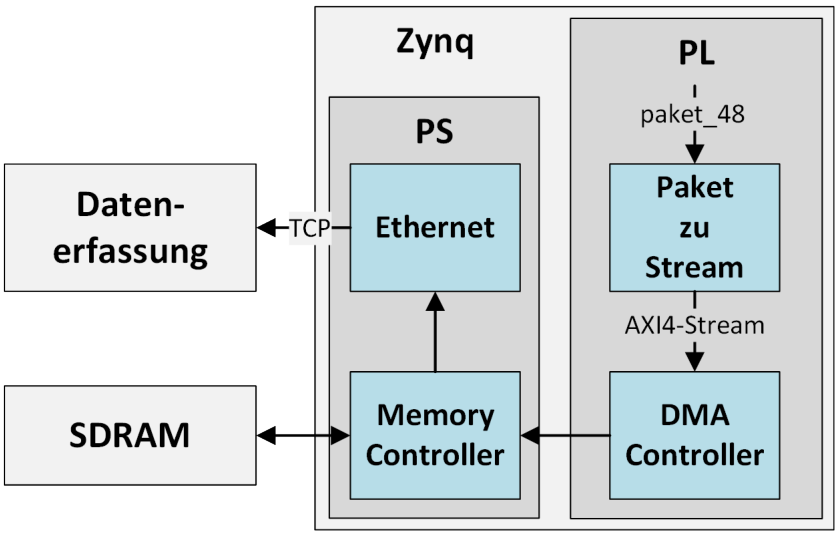

Abbildung 4.21.: Speicherkonzept
Die Kommunikation des Prozessors mit DMA-, Memory-Controller und Ethernet ist in dieser Abbildung nicht dargestellt.

Controller im AXI4-Stream-Protokoll die Pakete in 32-Bit-Vektoren senden.

Um die Übertragungsstrecke abzuschließen, fehlt noch die Definition des Protokolls der Ethernet-Übertragungsstrecke vom Zynq zum Datenerfassungssystem. Generell kommen das TCP/IP-Protkoll oder das UDP/IP-Protkoll für die Ethernet-Übertragungsschnittstelle in Frage. Das UDP/IP-Protkoll bietet durch die fehlende Fehlerbehandlung eine höhere Nutzbandbreite, jedoch ist nicht sichergestellt, dass die Daten korrekt übertragen werden [31, S. 12]. Da nur eine Verbindung zwischen Datenerfassungssystem und Zynq existiert, müssen auch Anweisungen wie das Starten einer Aufnahme über die Ethernetverbindung gesendet werden. Des Weiteren sollen alle von Timepix3 gesendeten Pakete korrekt abgespeichert werden. Aus diesem Grund wird das TCP/IP-Protkoll verwendet, das durch die integrierte Fehlerbehandlung die Korrektheit der Übertragung sicherstellt. Abbildung 4.21 zeigt den kompletten Kommunikationskanal von der Paketerkennung zum Datenerfassungssystem. Das Kommunikationsmodul empfängt 48-Bit-Pakete vom Paketerkennungsmodul und schreibt diese als 32-Bit-Vektoren über dem DMA-Controller in den externen SDRAM. Der Prozessor kann anschließend die gespeicherten Pakete über TCP/IP zum Datenerfassungssystem senden. Im nächsten Abschnitt wird die Systemkontrolle konzipiert, die den kompletten Operationsablauf steuert.

4.6. Systemkontrolle

Es muss ein Konzept entwickelt werden, das den Operationsablauf von Timepix3 steuert und überwacht. Dafür muss mit den bereits konzipierten Modulen kommuniziert werden. In Abbildung 4.22 ist ein kompletter Kommunikationsablauf zeitlich dargestellt. Dabei wird sich an einen typischen Operationsablauf von Timepix3 gehalten (vgl. Kapitel 2.2.4). Die

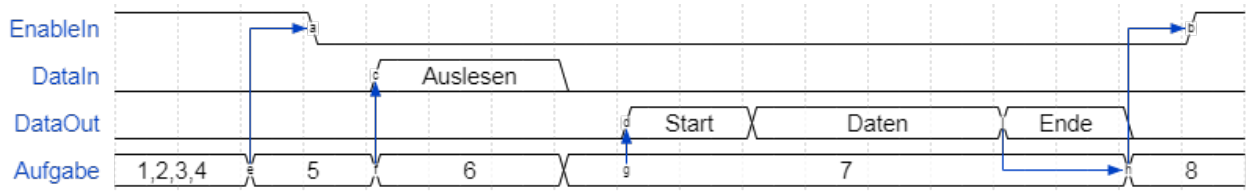

Abbildung 4.22.: Zeitablaufdiagramm der Systemkontrolle einer Timepix3 Operation

folgende Auflistung zeigt die zu den Nummer aus der Abbildung gehörenden Aufgaben, die getätigt werden müssen, um eine Operation mit Timepix3 erfolgreich durchzuführen.

1. Abfragen ob die Paketerkennung sowie das “Paket zu Stream“-Modul bereit für eine neue Operation sind.
2. Die Datenaufnahme synchronisieren.
3. Für die Operation spezifische Header in die Paketerkennung laden.
4. Der Paketerkennung signalisieren, dass Daten erwartet werden.
5. Timepix3-Steuersignale wie zum Beispiel “EnableIn“ oder “Shutter“ setzen.
6. Daten, welche die Operation einleiten, zu Timepix3 senden.
7. Empfangene Pakete aus dem DMA auslesen und zum Datenerfassungssystem senden.
8. Die Operation beenden, wenn die Paketerkennung das letzte Paket erkannt hat.

Für die Realisierung der aufgeführten Aufgaben gibt es mehrere Lösungsansätze. Dabei wurde bereits in Abschnitt 4.2 festgelegt, dass die Befehlsauswahl im PS stattfindet. Die anderen Aufgaben können in der PL oder in der PS des Zynqs abgearbeitet werden. Für die Konzeptionierung in der PL spricht, dass Teile der Aufgaben parallel abgearbeitet werden können. Für die Realisierung im PS spricht, dass bereits die Pakete von dort gesendet werden. Das hat den Vorteil, dass eine Funktion geschrieben werden kann, die den kompletten Ablauf steuert. Dies bietet eine übersichtliche und in der Konfiguration von Timepix3 flexible Möglichkeit, die Aufgaben abzuarbeiten. Hier wird sich dafür entschieden, beide Lösungsansätze zu kombinieren und die Abarbeitung der Aufgaben in der PL sowie im PS zu realisieren. Zwar ist bei diesen Lösungsansatz auch eine zusätzliche Kommunikationsschnittstelle zwischen PS und PL nötig. Allerdings kann diese kleiner gehalten werden, da das Modul in der PL viele Teilaufgaben übernimmt. Ein AXI4-Lite-Interface, das Zugriff auf Register in

der PL besitzt, kann für die Kommunikation zwischen PS und PL implementiert werden. Das ermöglicht eine flexible Konfiguration von Timepix3 über das PS, während Teile der Aufgaben parallel in der PL abgearbeitet werden können. Wie in Abbildung 4.23 zu sehen ist, wird dafür eine Systemkontrolle in der PL konzipiert, die nach einem Befehl vom PS bestimmte Aufgaben abarbeitet.

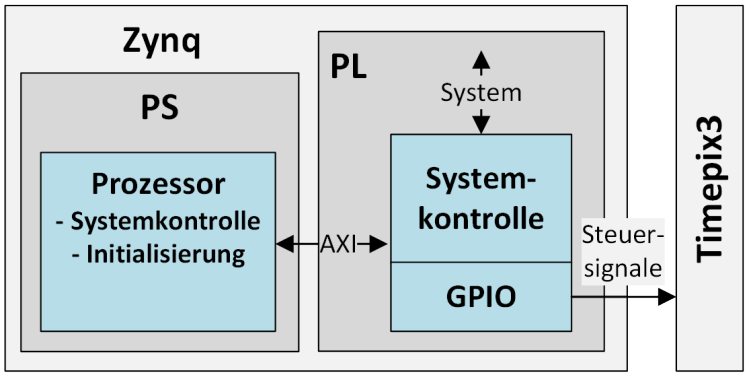

Abbildung 4.23.: Systemkontrolle Blockdiagramm

Aufgaben 1. bis 4. müssen dabei unabhängig von der Operationsart immer gleich ausgeführt werden, weswegen sie in der PL realisiert werden. Lediglich die Header aus Aufgabe 3. variieren mit der Operationsart, weswegen diese im PS ausgewählt werden. Beim Start einer Operation kann somit eine Funktion im PS der PL signalisieren, dass Aufgaben 1. bis 4. ausgeführt werden sollen. Dabei übergibt die Funktion der PL gleichzeitig die erwarteten Header. Aufgabe 5. muss flexibel konfigurierbar sein, da abhängig von der Operation Shutter, EnableIn oder andere Steuersignale zu verschiedenen Zeitpunkten gesetzt werden müssen. Aufgrund dessen wird dieser Aufgabenteil in der PS durchgeführt. Allerdings sind die Steuersignale mit der PL des Zynqs verbunden. Demzufolge muss ein General-Purpose Input/Output (GPIO)-Block, der über eine AXI4-Lite-Schnittstelle mit dem PS verbunden ist, in der PL implementiert werden [32]. Sind Aufgaben 1. bis 5. abgeschlossen, kann die Funktion Aufgabe 6. ausführen und die Befehle zur Datenwiedergabe in den FIFO schreiben. Jetzt muss die Funktion Aufgabe 7. ausführen. Diese kann abhängig von der Operationsart unterschiedlich lange andauern. Ist die Operation abgeschlossen, muss gewartet werden, bis die restlichen Pakete aus dem Speicher zum Datenerfassungssystem gesendet worden sind. Anschließend kann die Operation beendet werden, indem das PS Aufgabe 8. ausführt.

4.7. Speicherung und Visualisierung der Daten

Die Speicherung und Visualisierung der Pakete findet im Datenerfassungssystem statt. Hier muss ein Programm, das auf einem Betriebssystem operiert, die Pakete entgegennehmen und abspeichern. Des Weiteren muss es möglich sein, über das Programm eine Aufnahme mit definierbarer Zeit einzuleiten. Optional sollen die empfangenen Pakete dekodiert und visualisiert werden. Im Rahmen dieser Arbeit wird die Abspeicherung und Visualisierung der Daten lediglich kurz behandelt.

Das Programm kann in verschiedenen Hochsprachen, wie C++ oder Python entwickelt werden. C++ bietet durch die Verwendung von Zeigern gute Möglichkeiten, die Pakete zu dekodieren. Allerdings wird in der Abteilung, in der diese Arbeit verfasst wird, primär mit Python gearbeitet. Zusätzlich ist die Visualisierung eines zweidimensionalen Pixelarrays einfacher in Python als in C++, da Python einen höheren Abstraktionsgrad hat. Aus diesen Gründen wird sich für die Verwendung von Python entschieden.

Damit die Hardware des Datenerfassungssystems optimal ausgenutzt wird, muss ein Konzept für die Aufgabenaufteilung ausgearbeitet werden. Zum einen kann das komplette Programm auf einem Kern des Prozessors realisiert werden. Zum anderen können mehrere Kerne parallel die Aufgaben des Programmes abarbeiten. Die Implementierung auf einen Kern hat den Vorteil, dass das Programm leichter implementiert werden kann. Jedoch ist ein einzelner Kern mit der gleichzeitigen Datenaufnahme, dem Dekodieren und der Verarbeitung stark ausgelastet. Das kann dazu führen, dass das Datenerfassungssystem nicht genug Daten entgegennehmen kann, was zur Folge hat, das Pakete im Zynq verloren gehen können. Aus diesem Grund werden die Aufgaben, wie in Abbildung 4.24 zu sehen ist, auf mehrere Kerne des Prozessors aufgeteilt.

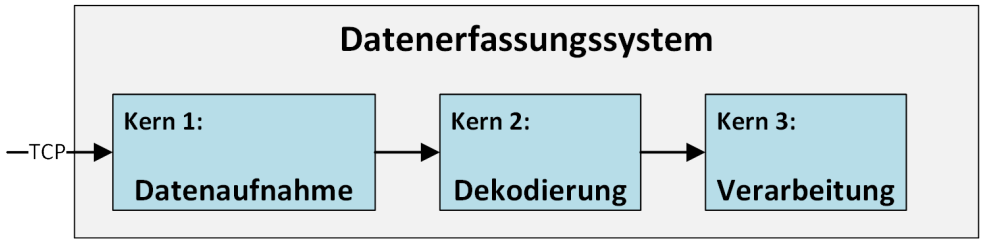

Abbildung 4.24.: Blockschaltbild Datenerfassung

Der erste Kern ist dabei für die Datenaufnahme verantwortlich. Pakete werden über TCP/IP an den Kern übertragen und von dort in den Arbeitsspeicher geschrieben. Der zweite Kern dekodiert die Pakete. Dabei werden aus der 16 Bit breiten Adresse der Pakete X- und Y-Koordinaten berechnet. Die X- und Y-Koordinaten repräsentieren Pixel auf dem zweidimensionalen Sensor, zu welchem die Daten aus dem Paketen gehören. Die meisten Daten wie die Anzahl der eingetroffenen Teilchen auf einem Pixel sind mit linear rückgekoppelten Schieberegistern codiert. Aufgrund des hohen Arbeitsspeichers im Datenerfassungssystem und um Rechenleistung zu sparen, werden die Schieberegister über Lookup-Tabellen dekodiert. Folglich repräsentieren die Werte der linear rückgekoppelten Schieberegister Indices in Lookup-Tabellen, in denen die dekodierten Werte hinterlegt sind. Die Lookup-Tabellen können mit einem Tabellenprogramm erstellt werden. Nach der Dekodierung werden die Koordinaten mit den dekodierten Daten zurück in den Arbeitsspeicher geschrieben, von wo sie vom 3. Kern zur Verarbeitung eingelesen werden. In der Verarbeitung können aus den dekodierten Daten und deren X- und Y-Koordinaten zweidimensionale Bilder, die den Sensor repräsentieren, erzeugt werden.

Die empfangenen Pakete können auch ohne Dekodierung abgespeichert werden. In dem Fall wird die Dekodierung übersprungen und der dritte Kern schreibt die Pakete vom Ar-

beitsspeicher in den Massenspeicher. Somit wird die Arbeit auf mehrere Kerne aufgeteilt und es können höhere Datenraten vom Ausleseboard empfangen werden.

Des Weiteren muss es möglich sein, nach Aufforderung des Nutzers eine Aufnahme bestimmter Zeit zu starten. Dies kann über ein grafische Benutzeroberfläche oder über die Konsole erfolgen. Da das Programm vorerst nur zu Entwicklungszwecken dient und keine komplexen Nutzereingaben empfangen werden, wird sich für eine Konsole entschieden. Aufgrund der im ersten Kern vorhandenen TCP/IP-Kommunikation mit dem Ausleseboard, findet die Eingabeaufforderung ebenso im ersten Kern statt. Des Weiteren werden während einer Aufnahme mit Timepix3 keine Eingaben vom Nutzer erwartet, weswegen die Datenaufnahme durch die Eingabeaufforderung nicht zusätzlich belastet wird.

5. Implementierung

In diesem Kapitel wird das zuvor erarbeitete Konzept implementiert. Dafür wird im ersten Abschnitt die Implementierung der Datenwiedergabe durchgeführt. Anschließend wird im Abschnitt 5.2 die Datenaufnahme implementiert. Die daraus resultierenden Daten werden von der Paketerkennung in Abschnitt 5.3 analysiert. Im Abschnitt 5.4 wird das Speichermanagement und die Ethernet-Kommunikation mit dem Datenerfassungssystem durchgeführt. Als letztes wird die Implementierung der Systemkontrolle in Abschnitt 5.5 behandelt. Im Rahmen dieser Arbeit wird nicht weiter auf die Software des Datenerfassungssystems eingegangen.

5.1. Datenwiedergabe

Um die Pakete vom PS zu Timepix3 zu senden, wird ein FIFO und ein VHDL-Serialisierer verwendet. Dabei wird der Serialisierer als Schieberegister implementiert. Das Blockschaltbild für die Ausgabe der Eingangsdaten für Timepix3 ist dabei in Abbildung 5.1 dargestellt. Im Folgenden wird auf die einzelnen Komponenten der Datenausgabe eingegangen.

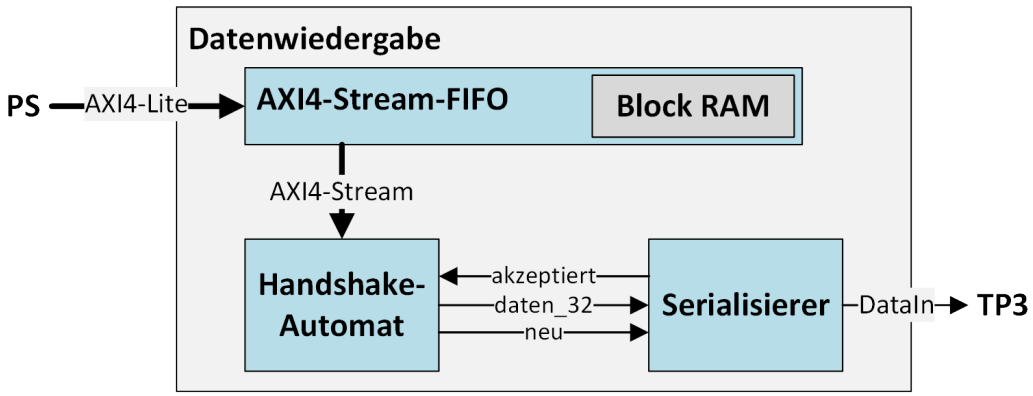

Abbildung 5.1.: Blockschaltbild Datenwiedergabe

Für den FIFO wird ein AXI4-Stream-FIFO von Xilinx verwendet [3]. Der IP-Core gibt 32-Bit-Vektoren im AXI4-Stream-Protokoll in der PL aus. Das bedeutet, dass im PS eine Funktion die erzeugten Pakete in 32-Bit-Datentypen formatieren muss, um diese in den FIFO zu schreiben. Ein erneutes Zusammensetzen der Pakete in der PL ist nicht nötig, da diese dort lediglich versendet und nicht betrachtet werden. Das Schieberegister des Serialisieres muss folglich nur 32 Bit lang sein. Da Timepix3-Pakete über 32 Bit lang sind, muss sichergestellt werden, dass das Schieberegister die Vektoren aus dem FIFO zusammenhängend

ausgibt. Andernfalls kommt es zu Lücken in den Paketen und demzufolge zu Fehlern in der Übertragung. Um die Daten aus dem FIFO auszulesen und in den Serialisierer zu schreiben, wird ein Handshake-Automat implementiert.

Der Handshake-Automat und der Serialisierer sind in Abbildung 5.2 dargestellt. Der

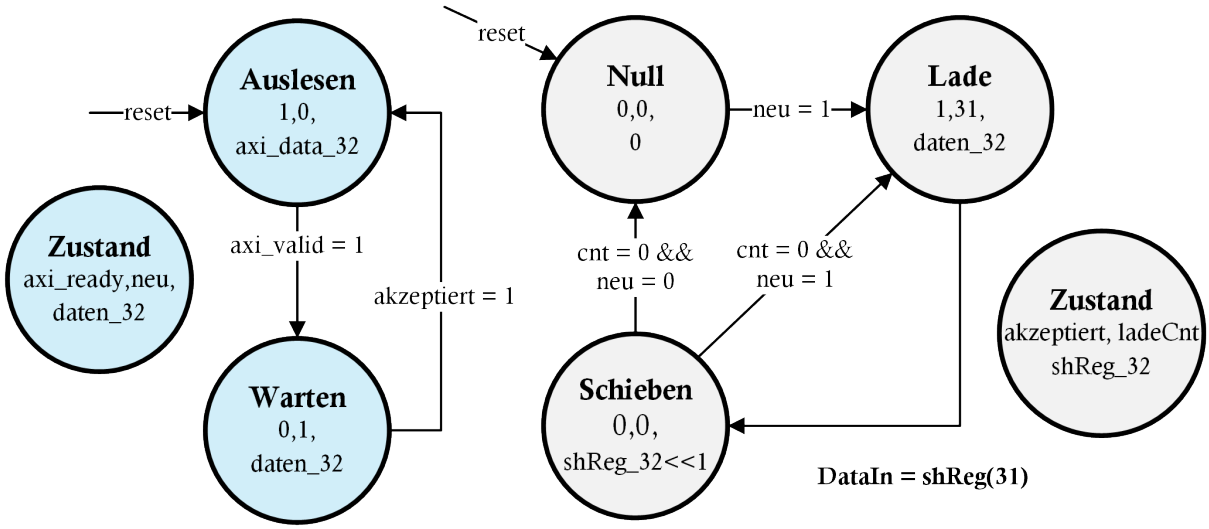

Abbildung 5.2.: Zustandsautomaten Datenwiedergabe
Links: Handshake-Automat, Rechts: Serialisierer mit Schieberegister

Handshake-Automat befindet sich nach dem Reset im Zustand “Auslesen“. In diesem Zustand wird “axi_data_32“ in ein Register des Handshake-Automaten geschrieben, welches der Serialisierer über das Signal “daten_32“ auslesen kann. Signalisiert der FIFO durch das Signal “axi_valid“, dass “axi_data_32“ neue Daten enthält, setzt der Handshake-Automat im Zustand “Warten“ das Signal “neu“ auf Eins. Der Serialisierer lädt daraufhin “daten_32“ in das Schieberegister und signalisiert mit dem Signal “akzeptiert“, dass der Vektor erfolgreich aus dem Register ausgelesen wurde. Gleichzeitig wird das erste Bit des neuen Vektors an den “DataIn“-Ausgang gesendet. Im nächsten Takt springt der Serialisierer in den Zustand “Schieben“ und schiebt den 32-Bit-Vektor zum “DataIn“-Ausgang. Der Handshake-Automat lädt in dieser Zeit den nächsten 32-Bit-Vektor aus dem FIFO. Durch die Verwendung von insgesamt zwei 32-Bit-Registern kann der Handshake-Automat einen neuen Vektor aus dem FIFO laden, bevor der Serialisierer den alten Vektor vollständig verarbeitet hat. Das hat den Vorteil, dass auf das “axi_valid“-Signal theoretisch mehrere Takte gewartet werden kann, ohne dass es zu Lücken in der Übertragung kommt. Hat der Serialisierer den ersten 32-Bit-Vektor komplett ausgegeben und das Signal “neu“ ist Eins, lädt es direkt den nächsten Vektor vom Handshake-Automaten in das Schieberegister. Sollten keine neuen Daten bereitstehen, beschreibt der Serialisierer selbständig sein Schieberegister mit Nullen. Geht das Signal “neu“ wieder auf Eins, wird das Schieberegister erneut mit “daten_32“ beschrieben.

Das serielle Ausgangssignal “DataIn“ des Serialisierers muss synchron mit 40MHz getaktet sein. Wie in Abbildung 5.3 zu sehen ist, muss sich dafür das letzte Flip-Flop des Ausgangssignales im IO-Block befinden. Andernfalls platziert das Implementierungswerkzeug

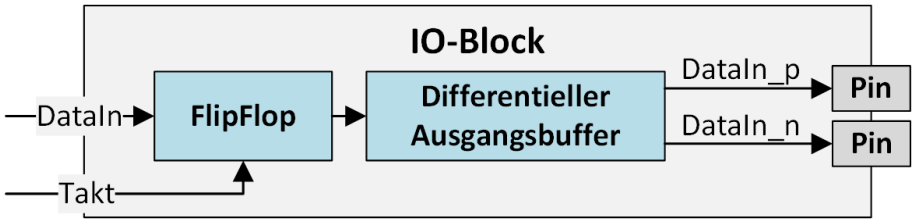

Abbildung 5.3.: Verarbeitung von DataIn im IO-Block

das Flip-Flop unabhängig von der Region des Pins vom Zynq. Dadurch hätte das Signal zwischen Ausgangs-FlipFlop und Pin eine unvorhersehbare Laufzeit und die Synchronität am Pin des Zynqs wäre nicht gewährleistet. Um ein FlipFlop im IO-Block zu platzieren, wird ein zusätzliches FlipFlop am Ausgang des Schieberegisters in VHDL implementiert. Damit das Implementierungswerkzeug das FlipFlop im IO-Block platziert, muss dieses über Constraints als solches markiert werden. Zusätzlich müssen die Daten differentiell ausgegeben werden. Dafür wird hinter dem FlipFlop ein differentieller Ausgangsbuffer im IO-Block implementiert. Über Constraints werden den Ausgangssignalen des Ausgangsbuffers die korrekten Pins sowie der LVDS-IO-Standard zugeordnet. Nach der Festlegung des Pins wird das Ausgangs-FlipFlop automatisch dem korrekten IO-Block zugewiesen.

Um eine korrekte Übertragung sicherzustellen, müssen die Setup- und Hold-Zeiten für das FlipFlop in den Constraints festgelegt werden. Wie in Abbildung 5.4 zu sehen ist, legen diese fest, in welchem Zeitraum die Daten stabil am Pin ausgegeben werden [33]. Da die

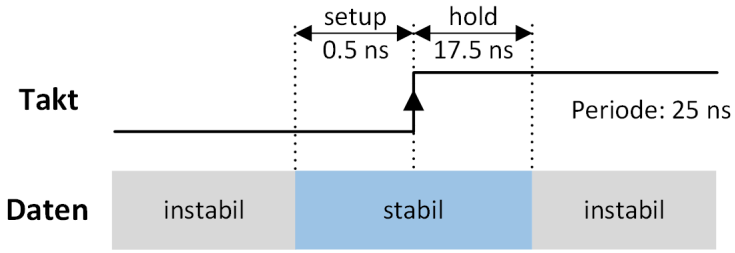

Abbildung 5.4.: Setup- und Hold-Zeiten des Pins

Signalleitungen vom Zynq zu Timepix3 in inneren Lagen der Platine verlaufen, ist es nicht möglich, die benötigten Setup- und Hold-Zeiten zu messen, ohne die Hardware zu beschädigen. Des Weiteren sind die internen Signallaufzeiten von Timepix3 nicht bekannt, was eine Messung zusätzlich erschwert. Daher werden die Zeiten so eingestellt, dass bei der gegebenen Übertragungsstrecke genug Reserven vorhanden sind. Infolgedessen wird im Anschluss getestet, ob es mit den gewählten Constraints möglich ist, mit Timepix3 zu kommunizieren. Dafür werden Register beschrieben, die beispielsweise den Ausgangstakt von Timepix3 ein und ausschalten. Wie Abbildung 5.4 zeigt, haben sich 0.5 ns für die Setup- und 17.5 ns für die Hold-Zeit als stabile Werte für die Übertragung herausgestellt.

Um das Modul zu verifizieren, wird es zuerst über eine Testbench simuliert. Da es sehr aufwendig ist, PS und PL zusammen zu simulieren, werden die Befehlsauswahl und der Se-

rialisierer getrennt getestet. In der ersten Simulation werden über das PS Pakete in den FIFO geschrieben, die über einen zweiten FIFO zurück in die PS gesendet werden und dort überprüft werden. In der zweiten Simulation wird der Handshake-Automat und der Serialisierer über eine Testbench getestet. Dabei wird der FIFO so simuliert, dass dieser 32-Bit-Vektoren im AXI4-Stream Protokoll dem Handshake-Automaten übergibt. Das Ausgangssignal des Modules wird anschließend in der Testbench überprüft. Um die Implementation zu testen, wird zusätzlich ein Test auf dem Auslesesystem mit Timepix3 durchgeführt. Dabei wird eine Timepix3-Peripherieoperation ausgeführt, die den Ausgangstakt von Timepix3 ein und ausschaltet. Der Ausgangstakt sowie das Signal "axi_data_32" des FIFOs wird im Zynq mit einem Logic-Analyzer überwacht. Der Test wurde erfolgreich durchgeführt, was auf korrekt gesendete Daten schließen lässt.

5.2. Datenaufnahme

Die Ausgangsdaten von Timepix3 sollen mit einem Deserialisierer im IO-Block aufgenommen werden. Dabei wird dieser über eine PLL mit dem Ausgangstakt von Timepix3 sowie mit einem Viertel des Ausgangstaktes versorgt. Ein Bit-Slip-Modul von Xilinx soll mit Hilfe eines Synchronisierers die Daten im korrekten Raster ausgeben. Das Blockschaltbild der Datenaufnahme ist dabei in Abbildung 5.5 dargestellt. Im Folgenden wird auf die einzelnen

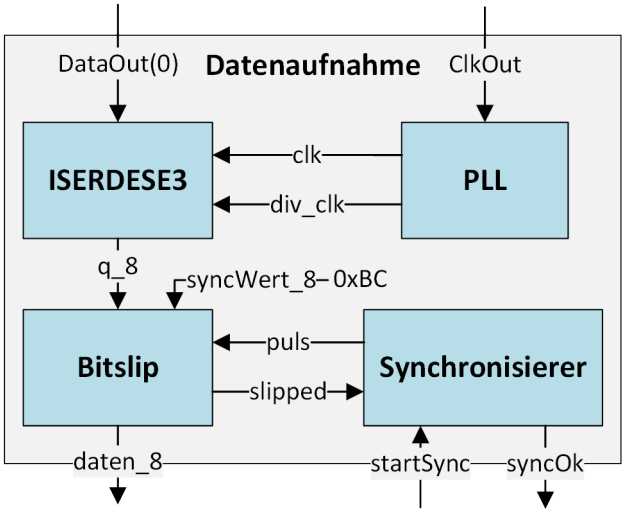

Abbildung 5.5.: Blockschaltbild Datenaufnahme

Komponenten der Datenaufnahme eingegangen.

Die Funktion von ISERDESE3 wurde bereits im Konzeptkapitel dargestellt. Die Primitive muss in VHDL instanziiert und dort über Generics konfiguriert werden. Bei der Konfiguration einer Primitive ist darauf zu achten, dass diese bereits in Hardware fest eingebettet ist und daher nur bestimmte Konfigurationskombinationen akzeptiert. Da die genaue Konfiguration zu tief ins Detail gehen würde, wird diese hier nicht weiter behandelt. Für die Implementation der PLL wird der Xilinx IP-Core "Clocking Wizard" verwendet [34]. In dem IP-Core müssen

die Taktgeschwindigkeiten der Ein- und Ausgänge der PLL definiert werden. Des Weiteren werden die Ein- und Ausgänge so eingestellt, dass sie phasengleich zueinander laufen. Dies ist wichtig, damit die Daten von Timepix3 nicht asynchron abgetastet werden.

Die Bitslip Logik von Xilinx wird im “Compare“-Modus betrieben. In diesem Modus nimmt die Bitslip-Logik den Datenstrom von ISERDESE3 sowie den konstanten Synchronisationswert “0xBC“ entgegen. Ist der Ausgang der Bitslip Logik gleich dem Synchronisationswert, gibt die Logik über das Signal “slipped“ eine Eins aus, andernfalls eine Null. Über das Signal “puls“ kann ein Puls gesendet werden, der den Ausgang “daten_8“ um ein Bit verschiebt. Nach dem Puls benötigt die Bitslip-Logik sechs Takte, um das Signal “slipped“ zu aktualisieren. Wie im Konzeptkapitel bereits erwähnt, muss also ein zusätzlicher VHDL-Automat mit der Bitslip-Logik interagieren. Folglich soll ein Synchronisierer auf Befehl eine Synchronisation durchführen und anzeigen, wenn diese erfolgreich durchgeführt wurde.

Wie dies funktioniert, zeigt der Zustandsautomat des Synchronisierers in Abbildung 5.6. Im Zustand “leerlauf“ wird, sofern der ISERDESE3-Ausgang bereits der Synchronisations-

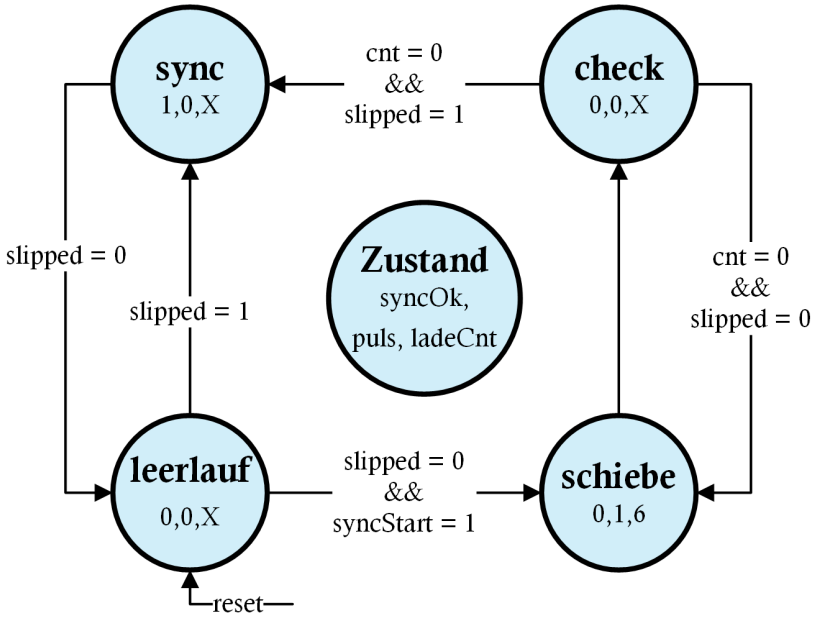

Abbildung 5.6.: Zustandsautomat Synchronisierer

sequenz entspricht, in den Zustand “sync“ gesprungen. Im Zustand “sync“ wird ausgegeben, dass die Ausgangsdaten der Datenaufnahme synchronisiert sind und somit eine Operation mit Timepix3 gestartet werden kann. Ist der ISERDESE3 nicht synchronisiert und das Signal “syncStart“ ist Eins, springt der Automat vom Zustand “leerlauf“ in den Zustand “schiebe“. Im Zustand “schiebe“ wird der Puls an die Bitslip-Logik gesendet. Nach sechs Takten überprüft der Automat im Zustand “check“, ob der Datenausgang synchronisiert ist. Ist dies der Fall, wird in den Zustand “sync“ gesprungen. Ist dies nicht der Fall, wird erneut in den Zustand “schiebe“ gesprungen und der Vorgang wird wiederholt. Werden nach der erfolgreichen Synchronisation Daten von Timepix3 empfangen, entspricht der Ausgang nicht

mehr dem Synchronisationswert. Folglich zeigt die Slip-Control an, dass der Ausgang nicht synchronisiert ist. Ist die Datenaufnahme beendet und die Synchronisation noch korrekt, springt der Automat automatisch zurück in den Zustand “sync“ und zeigt an, dass die Ausgangsdaten synchronisiert sind. Dadurch, dass die Ausgangsdaten maximal um sieben Bits verschoben sein können, benötigt der Synchronisationsvorgang maximal 42 Takte.

Anschließend müssen die Constraints für die Ausgangssignale von Timepix3 definiert werden. Das Ausgangssignal “DataOut“ von Timepix3 wird dabei auf einen differentiellen Eingansbuffer gegeben, der sich vor dem ISERDESE3 befindet. Der Eingangsbuffer arbeitet im SLVS-IO-Standard und ist mit 100 Ohm terminiert.

Die Verifikation der Datenaufnahme wird zuerst über eine Testbench durchgeführt. Die Testbench serialisiert 32-Bit-Vektoren und sendet diese an den Eingang des ISERDESE3 im DDR-Format. Anschließend wird der Ausgang der Datenaufnahme mit den gesendeten Vektoren verglichen. Um die Implementation zu testen, wird eine Timepix3-Peripherieoperation auf dem Auslesesystem durchgeführt und somit der Wert des OutputBlockConfig-Registers ausgelesen. Zu Beginn der Operation wird eine Synchronisation durchgeführt und anschließend der benötigte Befehl an Timepix3 gesendet. Ein Logic-Analyser überwacht den Ausgang der Datenaufnahme sowie die Signale “syncOk“ und “startSync“. Infolgedessen kann über den Analyser der Registerinhalt vom OutputBlockConfig-Register abgelesen und überprüft werden. Der Test wurde erfolgreich durchgeführt, was auf korrekt empfangene Daten schließen lässt.

5.3. Paketerkennung

Im Folgenden muss das Konzept der Paketerkennung implementiert werden. Dafür wird zuerst das Blockschaltbild für die Paketerkennung erklärt, welches in Abbildung 5.7 gezeigt ist. Der acht Bit breite Datenstrom der Datenaufnahme wird direkt in das RAM-Schieberegister geschrieben. Gleichzeitig wird der Datenstrom auf Header und Synchronisationssequenzen im Vergleicher überprüft. Die meisten Timempix3-Operationen werden mit drei unterschiedlichen Headern gesendet. Der Start-Header wird im ersten und der Ende-Header im letzten Paket einer Operation gesendet. Start- und Ende-Header sind beide acht Bit breit. Alle Pakete zwischen dem Start- und dem Ende-Paket haben als Header einen Daten-Header, der vier Bit breit ist. Da jede Timepix3-Operation unterschiedliche Header verwendet, können die Header vor jeder Operation über das Signal “header_20“ in den Vergleicher geladen werden. Sofern der Datenstrom mit einem Header übereinstimmt, gibt der Vergleicher am entsprechenden Ausgang eine Eins aus. Dabei werden die Paket-Header in ein Schieberegister geschrieben, welches die möglichen Header im Datenstrom repräsentiert. Die Start- und Ende-Header können zur Erkennung einzeln an den Analysierer gesendet werden, da der Paketinhalt von Start- und Ende-Paketen nie ein ID am Ende beinhalten kann.

Im Blockschaltbild werden drei Pakete rückwirkend überprüft, wodurch die Schieberegister, welche den Datenstrom repräsentieren, 18 Bit lang sind. Vor der Synthese ist die Anzahl der rückwirkend überprüften Pakete jedoch über Generics konfigurierbar. Zusätzlich wird ein Idle-Schieberegister implementiert, das lediglich fünf Bit tief ist. Warum der Analysierer dieses Register benötigt, wird später erläutert. Der Analysierer ist ein Zustandsautomat, der

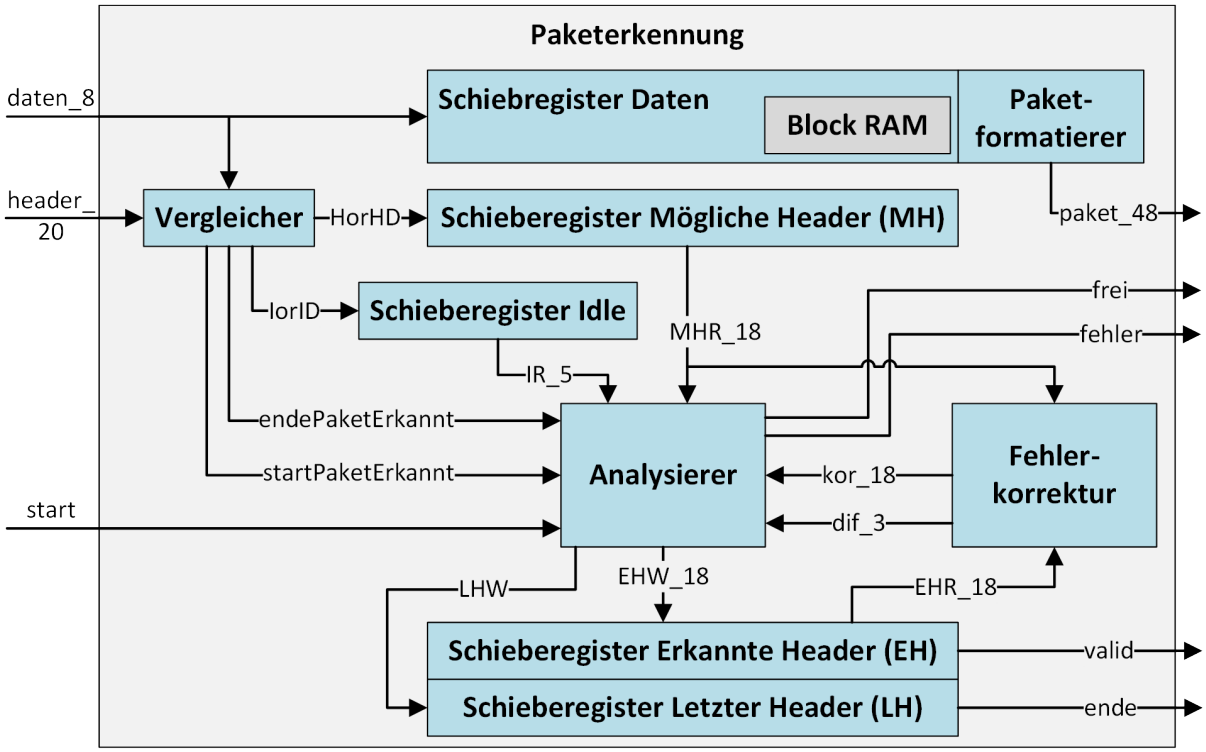

Abbildung 5.7.: Blockschaltbild Paketerkennung

das Zählen bis zu einem Header übernimmt. Über das Signal “start“ kann eine Operation mit dem Analysierer gestartet werden. Befindet sich der Analysierer nicht in einer Operation, wird das Signal “frei“ auf Eins gesetzt. Sollten bei einer Operation längere Zeit keine Daten von Timepix3 empfangen werden, wird das Signal “fehler“ ausgegeben. Um erkannte Pakete im Datenstrom zu markieren, beschreibt der Analysierer das EH-Schieberegister mit dem Signal “EHW_18“. Das Signal “valid“ zeigt an, wenn eine Eins aus dem EH-Schieberegister geschoben wird. In diesem Fall entsprechen die letzten sechs Byte des Daten-Schieberegisters einem erkannten Paket. Da das Daten-Schieberegister nur acht Bit breit ist, stellt ein Paketformatierer über das Signal “paket_48“ die letzten sechs Byte des Daten-Schieberegisters dar. Über das Signal “ende“ wird signalisiert, dass der Paketformatierer das Ende-Paket ausgibt und somit die Operation erfolgreich abgeschlossen ist. Die Fehlerkorrektur erkennt mit den Signalen “MHR_18“ und “EHR_18“ falsch erkannte Pakete im Datenstrom. Erkennt der Analysierer einen Fehler, ließt dieser im nächsten Takt den korrigierten Fehler “kor_18“ von der Fehlerkorrektur aus und lädt diesen in das EH-Schieberegister. Dabei wird zusätzlich die Differenz “dif_3“ übertragen, welche die Stellen repräsentiert, um die das Paket falsch erkannt wurde.

Zuerst wird der Analysierer implementiert. Dafür muss der Zählautomat aus dem Konzeptkapitel erweitert werden. Abbildung 5.8 zeigt den für den Analysierer implementierten Zustandsautomaten. Damit der Analysierer bei Synchronisationsvorgängen der Datenaufnahme nicht versucht, Pakete zu erkennen, befindet sich der Automat nach dem Reset im

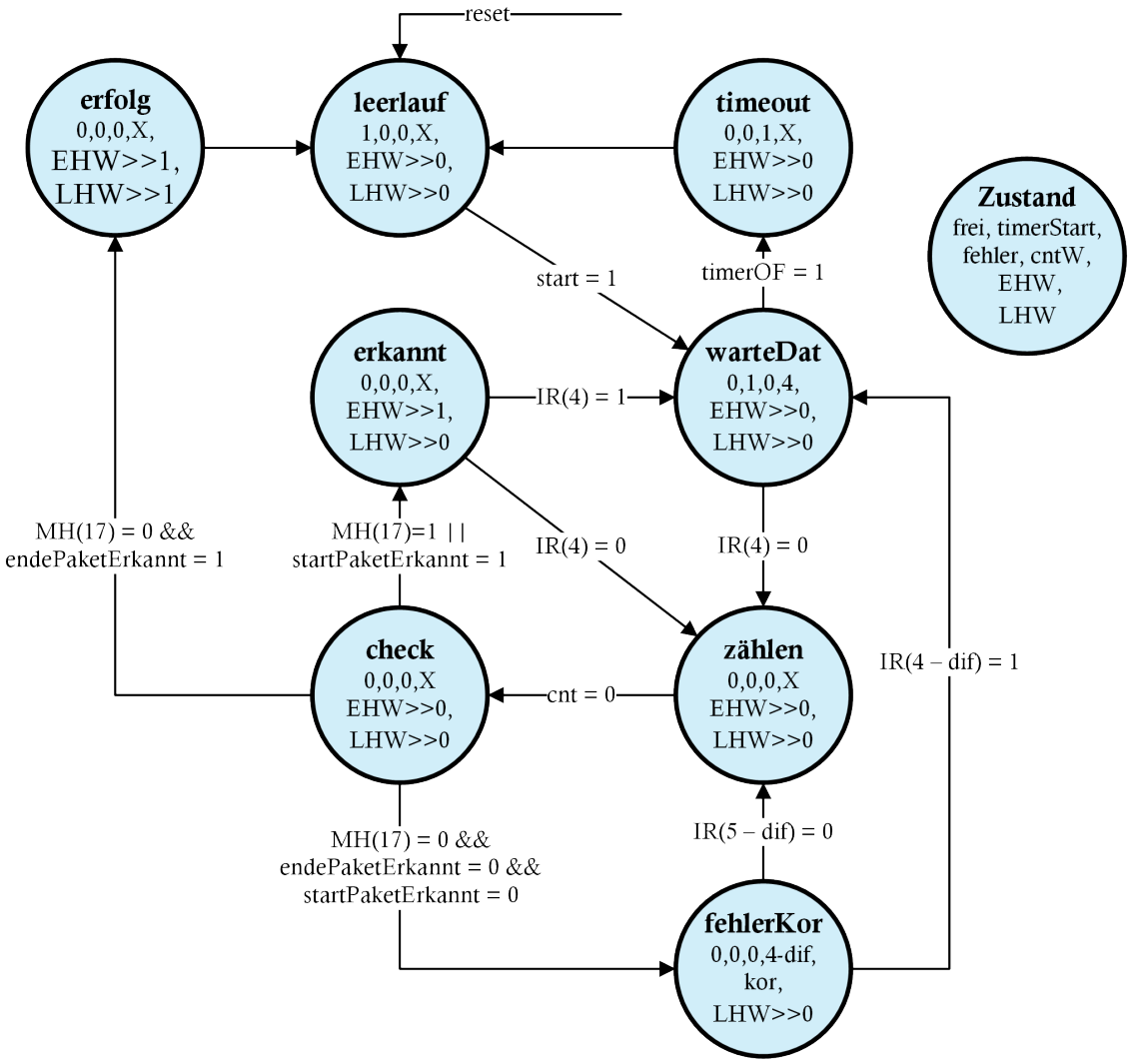

Abbildung 5.8.: Zustandsautomat Analysierer

Zustand “leerlauf“ und ist nicht empfänglich für Daten. Im Zustand “leerlauf“ signalisiert der Analysierer über das Signal “frei“, dass der Automat für eine neue Timepix3-Operation bereit ist. Wird dem Automat das Signal “start“ übergeben, springt dieser in den Zustand “warteDat“. In diesem Zustand wartet der Automat, dass die Daten ungleich I oder ID werden. Damit der Automat nicht im Zustand “warteDat“ hängen bleibt, wenn Timepix3 keine Daten sendet, wird gleichzeitig ein Timer gestartet. Der Timer gibt nach ungefähr einer Sekunde das Signal “timerOF“. Infolgedessen springt der Automat in den Zustand “timeout“, um die Operation kontrolliert zu beenden und das Signal “fehler“ zu senden. Werden jedoch die Daten im Zustand “warteDat“ ungleich I oder ID, wartet der Automat im Zustand “zählen“ vier Takte. Anschließend wird automatisch in den Zustand “check“ gesprungen, indem der Header überprüft wird. Im Zustand “check“ differenziert der Analysierer zwischen Start-, Ende- oder Paket-Header. Erkennt der Automat ein Start- oder Paket-Header, springt dieser

in den Zustand “erkannt“. In diesem Zustand wird eine Eins in das “EH“-Schieberegister geschoben. Sollte nach dem erkannten Header eine Synchronisationssequenz folgen, springt der Automat wieder in den Zustand “warteDat“. Andernfalls wird im Anschluss das nächste Paket empfangen und im Zustand “zählen“ erneut zum nächsten Header gezählt. Erkennt der Analysierer im Zustand “check“ keinen Header, wird in den Zustand “fehlerKor“ gesprungen. Hier wird der bereits korrigierte Fehler aus der Fehlerkorrektur geladen und mit diesen das gesamte “EH“-Schieberegister überschrieben. Zusätzlich überprüft der Automat mit Hilfe der Fehlerdifferenz, ob sich hinter dem richtigen Header eine Synchronisationssequenz befindet. Wird ein Paket fehlerhaft erkannt, nachdem eine Synchronisationssequenz gesendet wird, muss der Automat in den Zustand “warteDat“ springen. Wird jedoch nach dem falsch erkannten Paket direkt ein zweites Paket gesendet, muss der Automat nach der Fehlerkorrektur im Zustand “zählen“ zum nächsten Header zählen. Allerdings ist der Startzählwert abhängig von der vorherigen Fehlerdifferenz “dif“. Ansonsten kann es passieren, dass der Analysierer im falschen Raster weiterzählt. Die Operation ist erfolgreich beendet, wenn der Automat im Zustand “check“ das Signal “endePaketErkannt“ empfängt. Folglich wird im Zustand “erfolg“ eine Eins in das LH- und EH-Schieberegister geschoben.

Im Folgenden wird die Fehlerkorrektur implementiert. Die Fehlerkorrektur muss innerhalb eines Taktes den Fehler über die Schieberegister “MH“ und “EH“ erkennen und über das Signal “kor_18“ korrigiert ausgeben. Dabei soll der Datenstrom in einem Fehlerfall mit Vergleichsvektoren verglichen werden. Wie in Abbildung 5.9 zu sehen ist, wird die Fehlerkorrektur mit mehreren kombinatorischen Blöcken realisiert. Um einen Fehler zu korrigieren,

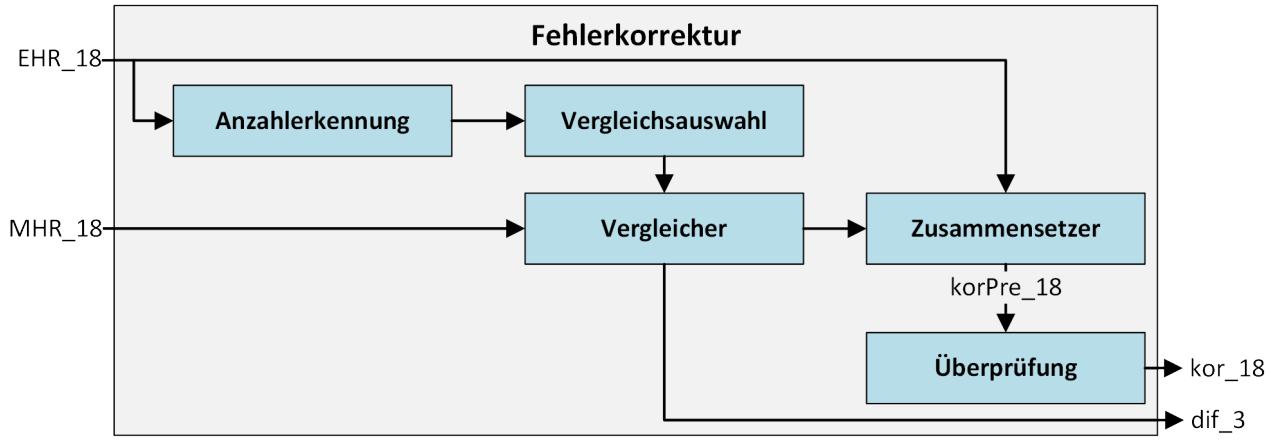

Abbildung 5.9.: Blockschaltbild Fehlererkennung

muss zuerst die Anzahl der zusammenhängenden fehlerhaften Pakete bekannt sein. Dafür wird der Moment des Fehlers in Abbildung 5.10 genauer betrachtet. Die Signale “EHR_18“, “korPre_18“ und “kor_18“ werden dafür im 0. Takt (T0), indem der Fehler erkannt wird, dargestellt. In dem Beispiel sendet Timepix3 drei zusammenhängende Pakete, die alle drei falsch erkannt werden. Dass es sich um drei zusammenhängende Pakete handelt, kann daran erkannt werden, dass das “EH“-Schieberegister ausgehend vom Fehlerfall genau alle sechs Bits eine Eins gespeichert hat. Demzufolge kann aus den Positionen der erkannten Header im

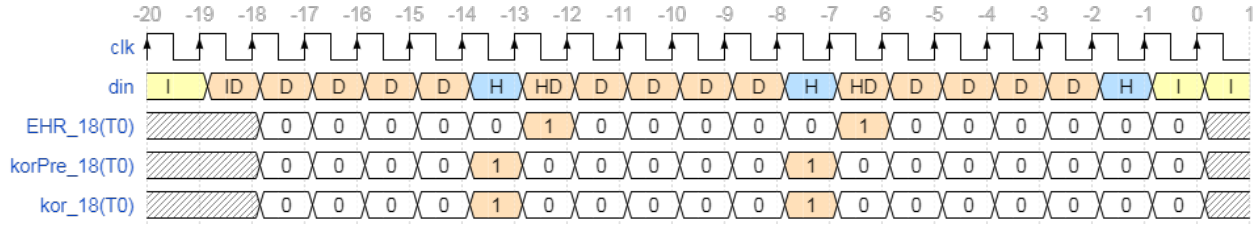

Abbildung 5.10.: Anzahlerkennung korrekt: Register im Fehlerfall

“EH“-Schieberegister die Anzahl der zusammenhängenden Pakete bestimmt werden. Diese Aufgabe übernimmt die Logik “Anzahl-Erkennung“.

Ist die Anzahl von falsch erkannten Paketen bekannt, gibt es eine begrenzte Anzahl an möglichen Fehlern, die aufgetreten sein können. Die “Vergleichsauswahl“ sendet die möglichen Fehler in Form von Fehlervektoren zum “Vergleicher“. Im “Vergleicher“ werden die Fehlervektoren mit dem falschen Teil des “MH“-Schieberegisters verglichen. Dabei wird der Vergleich priorisiert durchgeführt, sodass der Fehler mit der höheren Wahrscheinlichkeit den Vergleich gewinnt. Aus der “Vergleicher“-Logik kann ebenso die Fehlerdifferenz “dif_3“ für den Analysierer bestimmt werden. Im Beispiel aus Abbildung 5.10 beträgt die Fehlerdifferenz 1. Der korrekte Fehlervektor wird in der Logik “Zusammensetzer“ mit dem nicht fehlerhaften Teil des “EH“-Schieberegisters verodert und als “korPre_18“ ausgegeben. Allerdings werden bis zu diesem Block nur einfache Fehler korrigiert.

Die Logik “Überprüfung“ erkennt aus “korPre_18“ komplexe Fehler und korrigiert diese. Was bei einem komplexen Fehler passiert, ist in Abbildung 5.11 dargestellt. Wie bereits im

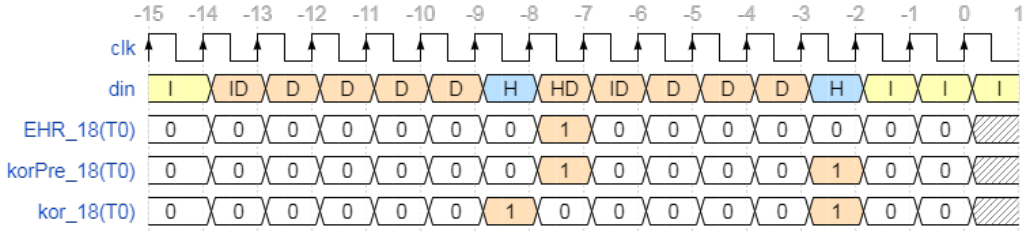

Abbildung 5.11.: Anzahlerkennung falsch: Register im Fehlerfall

Konzeptkapitel erklärt wurde, springt der Analysierer bei komplexen Fehlerfällen durch das ID im zweiten Paket für einen Takt in den Zustand “warteDat“. Aus diesem Grund wird der Fehler einen Takt später als üblich erkannt und der erkannte Header befindet sich sieben Bits hinter dem Fehlerfall. Daher befindet sich im sechsten Bit hinter dem Fehlerfall kein erkannter Header. Aufgrund dessen nimmt die Anzahlerkennung an, dass lediglich ein Paket falsch erkannt wurde. Folglich wird nur das zweite Paket korrigiert, während das erste Paket weiterhin fehlerhaft von “korPre_18“ ausgegeben wird. Dieser Fehler kann jedoch sehr einfach erkannt werden, da in “korPre_18“ der Header vom ersten Paket in den Daten vom zweiten Paket liegt. Die Überprüfung verschiebt daraufhin den zu erst erkannten Header soweit, dass die Differenz zwischen beiden Headern wieder sechs ist. Auch der komplexe Fehler

wird somit korrekt korrigiert. Diese sind äußerst selten, sodass die “Überprüfung“ in den meisten Fällen den Wert vom “Zusammensetzer“ übernimmt und ausgibt. Komplexe Fehler werden aufgrund des Ressourcenaufwandes nur bis zu drei zusammenhängenden Paketen erkannt. Die Verifikation der Paketerkennung ist Teil des Kapitels 6.

5.4. Speichermanagement und Ethernet

Um die erkannten Pakete von der PL zum Datenerfassungssystem zu senden, soll ein “Paket zu Stream“-Modul die Pakete in 32-Bit-Vektoren formatieren und dem DMA-Controller übergeben. Der Prozessor überträgt anschließend die Pakete, die sich im Speicherbereich befinden, unter Nutzung des TCP/IP-Protokolls zum Datenerfassungssystem.

Zuerst werden die Speicherbereiche des DMA-Controllers festgelegt. Damit der Prozessor die Speicherbereiche über Ethernet versenden kann, werden diese an die Datenkapazität eines Ethernet TCP/IP-Rahmens angepasst. Bei der TCP/IP-Standardrahmengröße können 1500 Bytes übertragen werden [35, S. 47]. Dazu soll ein Rahmen keine abgebrochenen Pakete enthalten, da es die Datenverarbeitung im Datenerfassungssystem erschwert. Werden 240 Timepix3-Pakete in einen Speicherbereich abgespeichert, entspricht das einem Datenbereich von 1440 Byte oder 360 32-Bit-Vektoren. Den Speicherbereichen des DMA-Controllers werden daher 1440 Byte zugewiesen. Die Anzahl der Speicherbereiche wird auf 1024 begrenzt, sodass dem DMA-Controller eine Gesamtspeicherkapazität von 1.5 MByte zur Verfügung steht. Nimmt das Datenerfassungssystem keine Daten entgegen, können diese nach Formel 5.1 über 30 ms ohne Paketverluste auf dem Ausleseboard zwischengespeichert werden.

$$t = \frac{Speicher}{Datenrate} = \frac{1.5\ MByte}{320\ MBit/s} = 37.5ms \tag{5.1}$$

Im Folgenden soll das “Paket zu Stream“-Modul aus Abbildung 5.12 implementiert werden. Das Modul kommuniziert mit der Paketerkennung über die Signale “valid“, “ende“ und

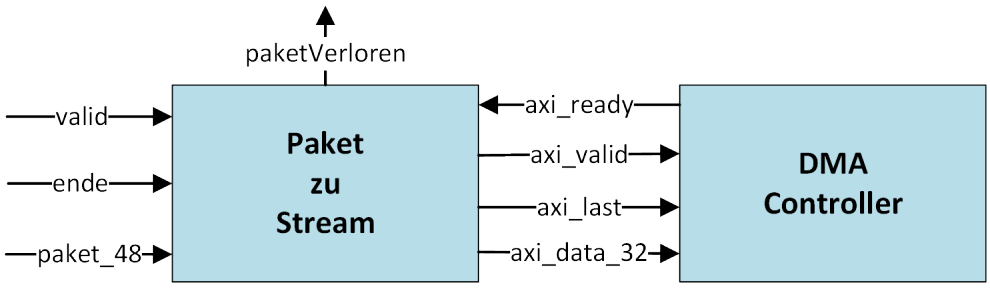

Abbildung 5.12.: Kommunikation mit dem DMA-Controller

“paket_48“. Das “Paket zu Stream“-Modul sendet kein “ready“-Signal an die Paketerkennung. Demnach ist es der Paketerkennung durchgehend gestattet, neue Pakete zu senden. Da das System allerdings mit 40 MHz taktet, kann höchstens alle sechste Takte ein neues Paket gesendet werden. Kommt es dennoch dazu, dass der Speicher überlastet ist und der

DMA-Controller das Signal “axi_ready“ auf Null setzt, werden empfangene Pakete verworfen. In diesem Fall wird das Signal “paketVerloren“ gesendet und von der Systemkontrolle ausgewertet.

Das Signal “axi_last“ hat in der Kommunikation mit dem DMA-Controller eine wichtige Funktion. Es zeigt dem Controller, dass ein Datenabschnitt abgeschlossen ist und die Daten zur weiteren Verarbeitung freigegeben sind. Das PS kann daher erst einen Datenabschnitt über Ethernet versenden, wenn dem Controller das “axi_last“-Signal übergeben wird. Es bedeutet aber nicht, dass keine Daten mehr gesendet werden, sondern lediglich das ein definiert großer Abschnitt der Daten fertiggestellt ist. In diesem System wird das “axi_last“-Signal nach 360 32-Bit-Vektoren gesendet. Das entspricht genau 1440 Byte, womit genau ein Speicherbereich des DMA-Controller vollständig beschrieben wird. Infolgedessen wird der Prozessor benachrichtigt, wenn ein Speicherbereich des DMA-Controllers abgeschlossen ist. Die Speicheradresse, welche vom Prozessor ausgelesen wird, kann anschließend dem Ethernet-Controller übergeben werden.

Ebenso muss das “axi_last“-Signal beim letzten Paket gesendet werden. Die Paketerkennung signalisiert das letzte Paket einer Operation durch das “ende“-Signal. Dadurch, dass Timepix3 bei einer Aufnahme eine unbestimmte Anzahl von Paketen sendet, kann es dazu kommen, dass ein Speicherbereich des DMA-Controllers nicht vollständig genutzt wird. Dafür ist in den Deskriptoren zusätzlich die beschriebene Datenlänge eines Speicherbereichs hinterlegt. Auf diesem Weg ist es möglich, nur ein Teil des Speicherbereiches über Ethernet zum Datenerfassungssystem zu senden. Das “Paket zu Stream“-Modul ist als Zustandsautomat implementiert, der hier nicht weiter erläutert wird.

Um die Übertragungsstrecke vom Eingang des DMA-Controllers bis zum Datenerfassungssystem zu verifizieren, werden mehrere separate Tests durchgeführt. In den Tests werden das “Paket zu Stream“-Modul, der DMA-Controller und die TCP/IP-Übertragung einzeln verifiziert. Anschließend wird die komplette Übertragungsstrecke auf dem Auslesesystem getestet, indem bekannte 48-Bit-Vektoren an das “Paket zu Stream“-Modul gesendet werden. Dafür wird ein 48-Bit-Zähler verwendet, der alle sechs Takte einen um Eins inkrementierten Vektor in das “Paket zu Stream“-Modul schreibt. Das entspricht einer Datenrate von 320 MBit/s. Die Vektoren werden in den SDRAM geschrieben, zum Datenerfassungssystem gesendet und dort nacheinander subtrahiert. Ist die Differenz nicht Eins, liegt ein Fehler in der Übertragungsstrecke vor. Ist die Differenz jedoch konstant Eins, konnten die 48-Bit-Vektoren mit 320 MBit/s korrekt zum Datenerfassungssystem gesendet werden. Der Test wurde erfolgreich durchgeführt. Anschließend wird der 48-Bit-Zähler durch die Paketerkennung ersetzt.

5.5. Systemkontrolle

Es soll eine Systemkontrolle in der PL implementiert werden, welche in Kommunikation mit dem PS steht und die Steuerung einer Operation durchführt. Wie in Abbildung 5.13 zu sehen ist, werden zwei 32-Bit-Register implementiert, um die Kommunikation zwischen PS und PL zu realisieren. Das Kontrollregister dient als Kommunikationsschnittstelle vom PS zur Systemkontrolle. Über dieses Register ist es dem PS möglich, die Start-, Ende- und Paket-Header für die nächste Timepix3-Operation zu übergeben. Des Weiteren kann über das Bit

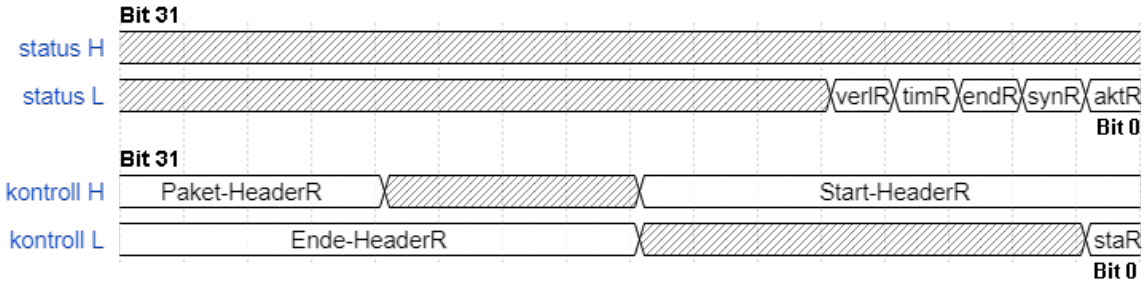

Abbildung 5.13.: Registersatz Systemkontrolle

“staR“ eine Timepix3-Operation gestartet werden. Das Statusregister dient als Kommunikationsschnittstelle von der Systemkontrolle zum PS. In diesem Register wird der Zustand der Operation signalisiert. Dabei werden dem PS über einzelne Bits des Registers Informationen mitgeteilt, die in Tabelle 5.1 näher erklärt sind.

Bit	**Beschreibung**
aktR	Ist Eins, wenn sich das System in einer Operation befindet.
synR	Ist Eins, wenn die Datenaufnahme synchronisiert wird.
endR	Ist Eins, wenn eine Operation abgeschlossen ist.
timR	Ist Eins, wenn Timepix3, obwohl erwartet, keine Daten sendet.
verlR	Ist Eins, wenn ein Paket verloren wurde.

Tabelle 5.1.: Statusregister Beschreibung

Wie in Abbildung 5.14 zu sehen ist, steuert ein Automat in der PL mit beiden Registern das gesamte System. Nach dem Reset befindet sich die Systemkontrolle im Zustand “leerlauf“. Dort wartet der Automat auf das Signal “staR“ aus dem Kontrollregister. Startet das PS eine Operation, wird abhängig vom Synchronisationszustand der Datenaufnahme eine Synchronisation eingeleitet oder direkt in den Zustand “ladeHeader“ gesprungen. Zur Sicherheit wird zusätzlich über das Signal “frei“ abgefragt, ob die Paketerkennung bereit für eine Operation ist. Durch verschiedene Verriegelungen sollte die Paketerkennung im Zustand “leerlauf“ allerdings immer bereit für eine Operation sein. Im Zustand “ladeHeader“ werden die Header aus dem Kontrollregister in die Paketerkennung geladen. Anschließend wird im Zustand “OPstart“ das Signal “start“ an die Paketerkennung gesendet. Hat die Paketerkennung das Signal “start“ akzeptiert, wird das Signal “frei“ Null und der Automat springt in den Zustand “OPaktiv“. In diesem Zustand wird im Statusregister das Signal “aktR“ auf Eins gesetzt. Daraufhin sendet das PS die Befehle an Timepix3, welche die gewünschte Operation ausführen. Folglich sendet Timepix3 je nach Operation Pakete, die anschließend von der Paketerkennung über den DMA-Controller in den Speicher geschrieben werden. Das PS versendet die Pakete gleichzeitig über Ethernet. Kommt es während der Operation zu Fehlerfällen in den Modulen, werden diese im Zustand “OPaktiv“ dem PS mitgeteilt, indem das Statusregister dementsprechend gesetzt wird. Geht das Signal “frei“ von der Paketerkennung wieder auf Eins, wurde das letzte Paket in den Speicher geschrieben und die Operation ist

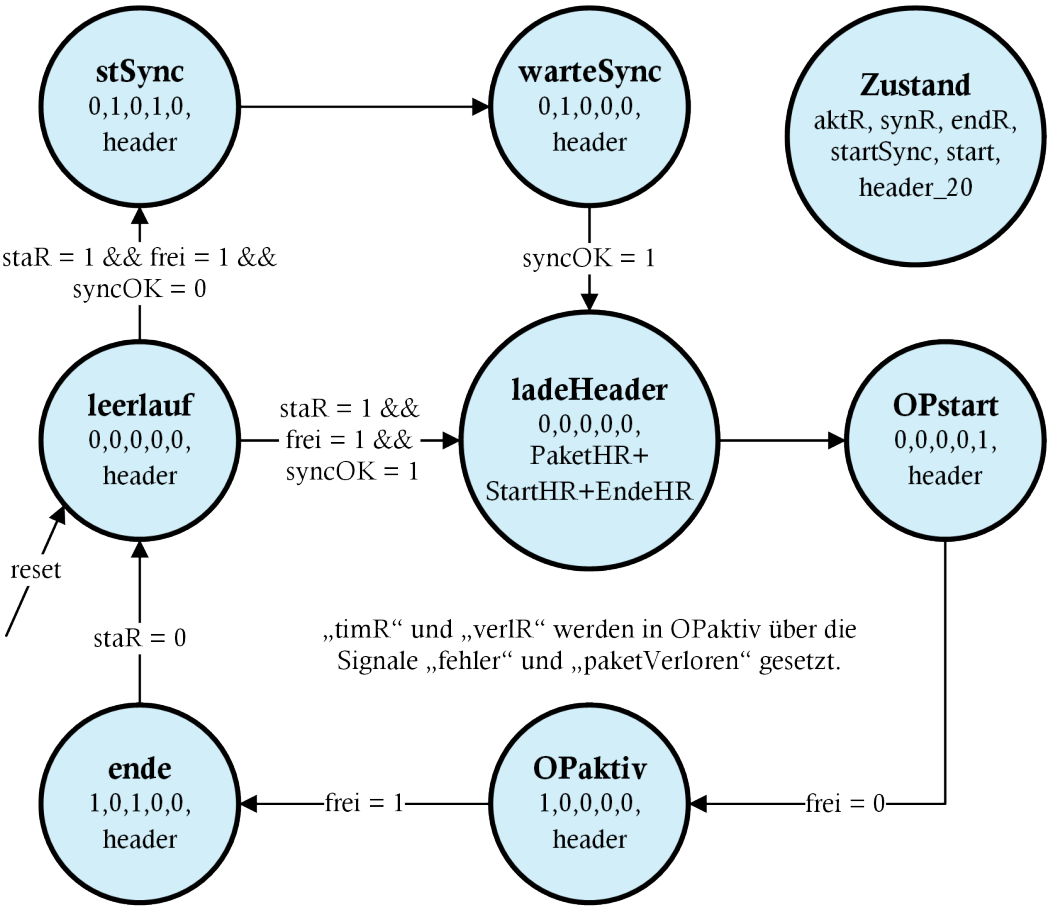

Abbildung 5.14.: Zustandsautomat Systemkontrolle

beendet. Im Zustand “ende“ wird ein Handshake zwischen Automat und PS durchgeführt. Im Handshake setzt der Automat das Signal “endR“ auf Eins und wartet darauf, dass das PS das Signal “staR“ wieder auf Null setzt. Ist dies geschehen, springt der Automat in den Zustand “leerlauf“ und es kann eine neue Operation gestartet werden. Der Automat wurde über eine Testbench erfolgreich verifiziert.

6. Erprobung und Auswertung

In diesem Kapitel soll das implementierte System verifiziert werden. Dafür wird im Abschnitt 6.1 auf die Paketerkennungsrate eingegangen. Im Abschnitt 6.2 wird eine Aufnahme durchgeführt und die daraus resultierenden Daten analysiert. Im letzten Abschnitt werden mögliche Systemoptimierungen des Auslesesystems diskutiert.

6.1. Paketerkennungsrate

Um die Paketerkennungsrate zu ermitteln, wird die Paketerkennung mit einer Testbench getestet. Wie in Abbildung 6.1 zu sehen ist, werden dafür zwei Test-Prozesse implementiert. Die Datenstromgenerierung erzeugt einen zufälligen Timepix3 Datenstrom mit Start-,

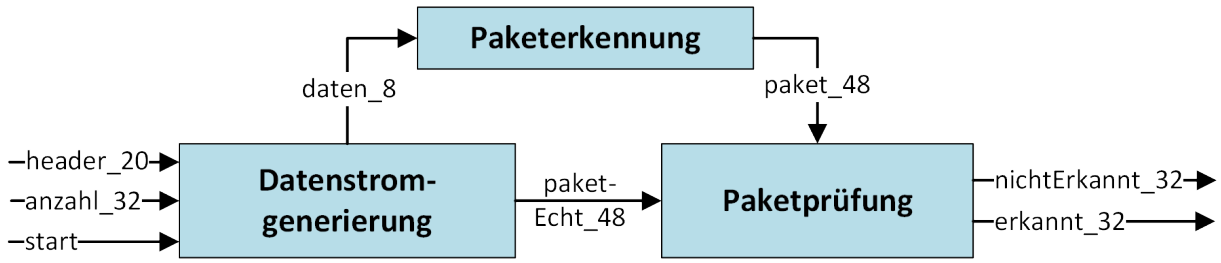

Abbildung 6.1.: Blockschaltbild Testbench Paketerkennung

Ende- und Datenpaketen. Über das Signal “anzahl_32“ kann die Anzahl an zu generierenden Paketen übergeben werden. Zwei Zufallsgeneratoren erzeugen den Datenstrom für die Paketerkennung. Dafür erzeugt der erste Zufallsgenerator zufällige Daten für die Datenpakete. Der zweite Zufallsgenerator entscheidet, ob nach einem gesendeten Paket ein weiteres Paket gesendet wird oder eine Synchronisationssequenz von zufälliger Länge folgt. Die Parameter der Zufallsgeneratoren werden so gewählt, dass der erzeugte Datenstrom “daten_8“ am ehesten der Realität entspricht. Die wirklich generierten Pakete sendet die Datenstromgenerierung mit dem Signal “paketEcht_48“ an die Paketprüfung. Die Paketprüfung wertet kontinuierlich aus, ob die Paketerkennung die Pakete korrekt ausgibt. Da die Paketerkennung abhängig von der Länge des analysierten Datenstromes mehrere Takte in Anspruch nimmt, werden die gesendeten Pakete der Datenstromgenerierung in einem internen Ringbuffer gespeichert. Gibt die Paketerkennung ein Paket aus, wird das Paket auf die Korrektheit im Ringbuffer überprüft. Abhängig davon, ob die Paketerkennung das Paket korrekt, nicht korrekt oder gar nicht erkannt hat, werden die Signale “erkannt_32“ oder “nichtErkannt_32“ inkrementiert.

In der Paketerkennung kann die Anzahl der zu analysierenden Paketen über Generics konfiguriert werden. Um eine sinnvolle Anzahl an zu analysierenden Paketen zu bestimmen, werden mehrere Simulationen durchgeführt. In jeder Simulation werden der Paketerkennung 20 Millionen Pakete gesendet. In Tabelle 6.1 sind die falsch erkannten Pakete und die daraus resultierende Wahrscheinlichkeit, dass ein Paket falsch erkannt wird, aufgeführt. Analysiert die Paketerkennung lediglich ein Paket rückwirkend, wird keine Fehlerkorrektur

Pakete	**Fehler / 20 M**	**Fehlerwahrscheinlichkeit (‰)**
1	123579	6.1790
2	7754	0.3877
3	945	0.0473
4	28	0.0014
5	7	0.0004
6	6	0.0003
7	5	0.0003

Tabelle 6.1.: Simulation Paketerkennung

durchgeführt. Wie Formel 6.1 zeigt, unterscheidet sich die simulierte von der erwarteten Fehlerwahrscheinlichkeit um den Faktor 1.6.

$$Faktor = \frac{gemessen}{berechnet} = \frac{6.179 \ / \ 1000}{1 \ / \ 256} = 1.6 \tag{6.1}$$

Das hängt damit zusammen, dass der Analysierer durch ein ID mehrere Pakete falsch erkennen kann, wenn diese zusammenhängend auftreten. Die falsch erkannten Pakete nehmen mit der Analyse von fünf rückwirkenden Paketen deutlich ab und bleiben auf einen relativ konstanten Fehlerwert. Im Anhang A.5 kann der benötigte Ressourcenaufwand für die Implementierung der entsprechenden Paketerkennung betrachtet werden. Der Zynq Ultrascale+ ZU3CG ist mit 141000 FlipFlops und 71000 LUTs ausgestattet und besitzt somit genug Ressourcen für alle in der Messung aufgeführten Implementierungen [11]. Jedoch ist ab fünf Paketen keine signifikante Verbesserung der Paketerkennung festzustellen. Aus diesem Grund wird die Paketerkennung mit fünf Paketen implementiert, was einem Ressourcenaufwand von 173 LUTs und 202 FlipFlops entspricht. Folglich wird die Wahrscheinlichkeit, dass ein Paket falsch erkannt wird, durch die implementierte Fehlerkorrektur um das 17000-fache reduziert.

Im Folgenden soll die Implementierung der Paketerkennung getestet werden. Um von Timepix3 einen Datenstrom mit 20 Millionen Paketen zu erhalten, muss Timepix3 mindestens genauso viele Treffer detektieren, was nur mit einer radioaktiven Strahlungsquelle möglich ist. Das Messen mit einer radioaktiven Strahlungsquelle ist durch Sicherheitsbestimmungen ein sehr aufwändiger Prozess. Aus diesem Grund werden die Aufnahmen mit kosmischer Strahlung durchgeführt, welche im nächsten Abschnitt näher erläutert wird. Auf der Erdoberfläche hat kosmische Strahlung den Nachteil, dass diese sehr gering ist und demzufolge nur wenig Treffer auf dem Sensor generiert. Dennoch sendet Timepix3 bei langen Aufnahmen von bis zu sechs Stunden bis zu 20000 Pakete. Bei allen durchgeführten Aufnahmen,

wurden insgesamt über 100000 Pakete empfangen. Dabei wurden über einen Logic-Analyzer 20 Zugriffe auf die Fehlerkorrektur festgestellt. Wie Formel 6.2 zeigt, ist damit die Anzahl an Zugriffen auf die Fehlerkorrektur in den durchgeführten Aufnahmen 19.5 Mal niedriger als erwartet.

$$Faktor = \frac{erwartet}{tatsächlich} = \frac{1\ /\ 256}{20\ /\ 100000} = 19.5 \tag{6.2}$$

Der Grund hierfür liegt darin, dass die Daten abhängig vom Experiment zu bestimmten Werten tendieren. Daher liegt in einer realen Messung in der Regel keine Gleichverteilung vor. Dazu kommt, dass durch unterschiedliche Informationsmodi von Timepix3 unterschiedliche Formen von Datenpaketen gesendet werden können. Beispielsweise existieren Datenpakete, die in den letzten vier Bit des Paketes einen Zählwert speichern, der in vielen Experimenten einen Überlauf anzeigt. Da der Wert ungleich "0xC" ist und somit die letzten acht Bit des Paketes nicht der Synchronisationssequenz entsprechen können, werde diese Datenpakete nicht falsch erkannt. Im Umkehrschluss kann in bestimmten Experimenten auch ein erhöhter Zugriff auf die Fehlerkorrektur erfolgen, wenn der Wert am Ende des Paketes öfter mit der Synchronisationssequenz übereinstimmt. Aus diesem Grund ist die Fehlerkorrektur zwingend notwendig. In den durchgeführten Aufnahmen konnten keine falsch oder nicht erkannten Pakete festgestellt werden.

6.2. Konfiguration und Aufnahme

Um das entwickelte System zu verifizieren, wird in diesem Abschnitt eine Aufnahme mit Timepix3 durchgeführt und analysiert. Dafür muss Timepix3 korrekt konfiguriert und angesteuert werden. Nach dem Systemstart wird Timepix3 initialisiert. Dabei werden Timepix3-Register mit Peripherieoperationen beschrieben und verschiedene Initialisierungssequenzen durchgeführt. Die genaue Konfiguration der Register kann inklusive Begründung im Anhang A.1 eingesehen werden. Für die Aufnahme wird Timepix3 im Informationsmodus "Event and iTOT" betrieben. In diesem Modus wird die Anzahl an Treffer pro Pixel anstatt über 4-Bit über ein 10-Bit-Register dargestellt. Das Register ist mit einem linear rückgekoppelten Schieberegister codiert und kann daher 0 bis 1022 Treffer pro Pixel darstellen. Um zu verifizieren, dass die von Timepix3 gesendeten Pakete korrekt im Datenerfassungssystem abgespeichert werden, wurde bereits jedes Modul der Übertragungsstrecke einzeln getestet.

Die Verifikation der Daten, welche Timepix3 sendet, muss mit einer korrekten Aufnahme durchgeführt werden. Dafür wird Timepix3 mit einer bekannten Strahlungsquelle beschossen. Anschließend müssen die Daten dekodiert und visualisiert werden. Entspricht das Bild der bekannten Strahlungsquelle, gibt Timepix3 korrekte Daten aus. Als bekannte Strahlungsquelle wird kosmische Strahlung verwendet. Der Nachteil von kosmischer Strahlung ist, dass sie zufällig auf den Sensor auftrifft und durch die niedrige Strahlungsintensität wenig Treffer auf dem Sensor erzeugt. Allerdings ist anders als bei einer radioaktiven Strahlungsquelle für die Messung mit kosmischer Strahlung kein zusätzlicher Testaufbau mit erhöhten Sicherheitsmaßnahmen nötig. Als Referenz für kosmische Strahlung wird eine Aufnahme des Hubble Weltraumteleskops genutzt, welche in Abbildung 6.2 dargestellt ist. Die Aufnahme zeigt eine Galaxie, die als heller, weißer Punkt in der Mitte des Bildes wahrzunehmen ist.

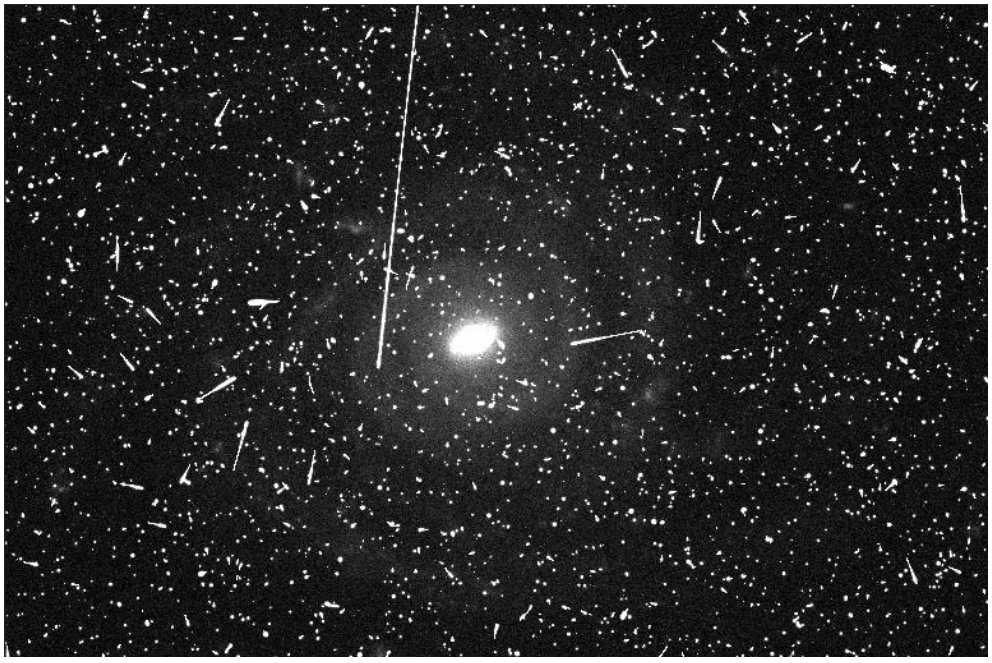

Abbildung 6.2.: Aufnahme Hubble Weltraumteleskop [4]

Außerdem können in der Aufnahme längliche, weiße Striche ausgemacht werden. Bei diesen Strichen handelt es sich um kosmische Strahlung, die auf die Sensoroberfläche des Teleskops trifft. Zu kosmischer Strahlung zählen verschiedene Teilchen, welche zu unterschiedlichen Formen bei einer Aufnahme führen. Im Rahmen dieser Arbeiten werden diese nicht separat aufgeführt. Eine weitere Referenz für kosmische Strahlung ist im Anhang A.2 Abbildung A.1 dargestellt.

Im Folgenden wird versucht, mit Timepix3 kosmische Strahlung aufzunehmen. Dabei wird der Ablauf des Kalibrationsvorganges mit seinen signifikanten Ereignissen beschrieben. Als erstes werden die empfangenen Daten von Timepix3 ohne zusätzliche Konfigurationsarbeit visualisiert. Dabei entsteht ein Bild (Anhang A.2 Abbildung A.2), das anzeigt, dass jeder Pixel einmal getroffen wurde. Das Bild zeigt keine kosmische Strahlung. Dennoch lassen sich durch das Bild erste Rückschlüsse auf die Korrektheit der Pakete ziehen. Es ist naheliegend, dass die Adressen und die Treffer pro Pixel korrekt übertragen und dekodiert wurden. Wäre das nicht der Fall, würden wahrscheinlich nicht alle Pixel gleichmäßig einen Treffer anzeigen. Allerdings müssen die Annahmen mit weiteren Kalibrierungen, die es ermöglichen kosmische Strahlung zu detektieren, verfestigt werden.

Dafür wird das analoge Pixel-Frontend kalibriert (vgl. Abschnitt 2.2.2). Im analogen Pixelfrontend wird mit zwei Strömen “I_{DAC}“ und “I_{TH}“ eine Schwellspannung eingestellt, die entscheidet, ob ein eintreffendes Teilchen als Treffer gewertet wird. Da alle Pixel einen Treffer anzeigen, wird vermutet, dass die Schwellspannung zu niedrig ist und beim Start einer Aufnahme sofort ein Treffer ausgelöst wird. Die restliche Zeit der Aufnahme werden keine weiteren Treffer angezeigt, da sich die Schwellspannung durchgehend unter der Ausgangsspannung der Krummenacher Rückkopplung befindet. Daher wird die Schwellspannung sukzessive an

die Sensitivität der erzeugten Spannung von kosmischer Strahlung herangeführt. Dabei entsteht ein Bild (Anhang A.2 Abbildung A.3), das erste Formen von kosmischer Strahlung zeigt. Demzufolge kann auf eine korrekte Übertragung und Dekodierung der Pixeladressen geschlossen werden.

Allerdings fällt auf, dass die kosmische Strahlung an manchen Stellen mehr als ein Treffer anzeigt. Grundsätzlich kann dies korrekt sein, wenn sich zum Beispiel zwei Teilchen schneiden oder hintereinander auf den Sensor treffen. Da in den markierten Bereichen im Bild jedoch keine Schnittstellen von kosmischen Teilchen zu erkennen sind, muss es sich um ein Fehler in der Kalibrierung handeln. Daraufhin wird festgestellt, dass die Treffer pro Pixel sich mit der Stärke des Stromes "I_{krum}" verändern. Der Strom steuert die Entladungszeit des durch den Treffer aufgeladenen Kondensators im analogen Pixelfrontend. Es wird vermutet, dass bei langen Entladungszeiten die Ausgangsspannung der Krummenacher Rückkopplung die Schwellspannung so langsam schneidet, dass durch Rauschen mehrere Treffer im digitalen Pixelfrontend detektiert werden. Daher wird der Strom "Ikrum" erhöht, damit der Kondensator sich schneller entlädt und das Rauschen auf den Spannungen keine weiteren Treffer erzeugt.

Als letztes werden die "Hot Pixel" maskiert. "Hot Pixel" sind Pixel, die zum Beispiel durch Fertigungsfehler durchgehend Treffer anzeigen. Um die Koordinaten der "Hot Pixel" ausfindig zumachen, werden mehrere kurze Aufnahme von 1 ms Länge getätigt (Anhang A.2 Abbildung A.6). Die Maskierung erfolgt über erneutes konfigurieren der Pixelmatrix, indem das Maskierungsbit der betroffenen Pixel auf Eins gesetzt wird. Demgemäß wird der Ausgang des analogen Frontends von den betroffenen Pixeln nicht auf das digitale Pixelfrontend gesendet. Abbildung 6.3 zeigt eine korrekte Aufnahme von kosmischer Strahlung nach der Maskierung der "Hot Pixel". Im Anhang A.2 sind weitere Aufnahmen von kosmischer

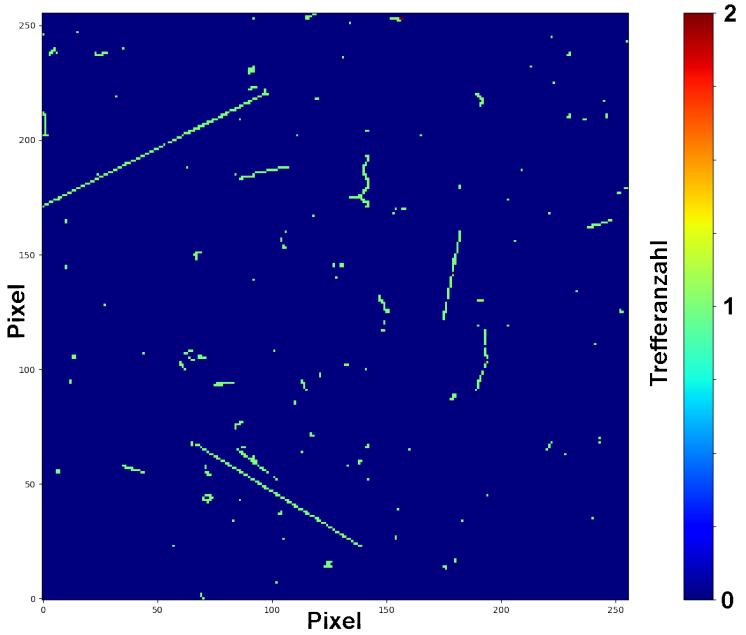

Abbildung 6.3.: Korrekte Aufnahme von kosmischer Strahlung mit Timepix3, 5 Minuten

Strahlung durch das Auslesesystem mit Timepix3 aufgeführt. Das System konnte mit der Messung von kosmischer Strahlung erfolgreich verifiziert werden.

6.3. Systemoptimierung

Bei der Entwicklung und Erprobung eines Timepix3 basierenden Auslesesystems konnten einige mögliche Systemoptimierungen festgestellt werden. Dabei wird im Rahmen dieser Arbeit lediglich auf die signifikanten Punkte eingegangen. Für zusätzlich empfohlene Systemoptimierungen wird auf den Anhang A.3 verwiesen.

Um eine sichere Paketerkennung zu gewährleisten, sollten GTH-Tranceiver implementiert werden. Somit kann der Ausgangsdatenstrom in der 8B/10B-Codierung betrieben werden, was eine sicherere Übertragung zur Folge hat. Des Weiteren wäre eine aufwändige Paketerkennung nicht nötig, da die Tranceiver die Synchronisationssequenzen von den Paketen trennen. Für die Implementierung mit vier Timepix3-Chips sind 32 GTH-Tranceiver notwendig. Ein weiteres Problem in der bestehenden Hardware ist die 1GBit Ethernet-Schnittstelle, welche mit vier Timepix3-Chips bei maximaler Datenrate überlastet ist. Daher muss eine Ethernet-Schnittstelle implementiert werden, die mindestens 20 GBit/s übertragen kann. Xilinx bietet Zynqs an, welche einen fest integrierten 100GBit-Ethernet-IP-Core besitzen. Der IP-Core erzeugt über die PL des Zynqs mit 10 GTH- oder GTY-Tranceivern eine Ethernet-Schnittstelle [36]. Ein zusätzlicher Vorteil der Verwendung des IP-Cores ist, dass dieser ohne das PS die Daten über Ethernet übertragen kann. Daher würde dieser Ansatz den Prozessor weiter entlasten und eine schnellere Übertragung zur Folge haben.

Um beide Optimierungsvorschläge in der Hardware zu implementieren, wird ein Zynq mit 42 GTH- oder GTY-Tranceiver benötigt. Der Zynq Ultrascale+ ZU11EG erfüllt diese Anforderung [37]. Er besitzt 32 GTH-Tranceiver und 16 GTY-Tranceiver und könnte somit alle Ausgangssignale von vier Timepix3-Chips erfassen. Gleichzeitig ist in dem Modell der 100GBit-Ethernet-IP-Core fest integriert. Eine günstigere Alternative ist der FPGA Kintex Ultrascale+ KU11P, der ebenso die benötigte Anzahl an Tranceivern, sowie einen integrierten 100GBit-Ethernet-IP-Core besitzt [38]. Allerdings müssten sämtliche Prozessoranwendungen, die bei einem Zynq im PS realisiert werden können, auf einen Soft-IP-Prozessor ausgelagert werden.

Sollte sich dennoch gegen die Verwendung von GTH- und GTY-Tranceivern entschieden werden, ist es ratsam, Teile des Systemes zu überarbeiten. Die implementierte Datenaufnahme hat sich in der Testphase als sichere Datenerfassung erwiesen, weshalb diese beibehalten werden sollte. Allerdings sollte geprüft werden, ob die Paketerkennung durch die Verwendung der 8B/10B-Codierung eliminiert werden kann. Dafür müsste, wie bereits im Kapitel 4.3 erwähnt wurde, eine komplexere Logik, welche die Dekodierung des 8B/10B-Codes übernimmt, implementiert werden. Auf diesem Weg könnte es möglich sein, alle Pakete ohne GTH- und GTY-Tranceiver korrekt zu detektieren. Aufgrund der aufwändigen Implementierung wird jedoch dazu geraten, den Datenstrom mit GTH- und GTY-Tranceivern zu erfassen.

Zum Abschluss werden die Schritte erläutert, welche getätigt werden müssen, um das System mit vier Timepix3-Chips zu betreiben. Die Datenwiedergabe und Datenaufnahme des Systemes muss für jeden weiteren Chip und Ausgangsdatenstrom erneut instanziiert werden.

Das “Paket zu Stream“-Modul des Speichermanagements muss überarbeitet werden. Bei 32 aktiven Ausgangsdatenströmen mit maximaler Datenrate müsste das Modul 20 GBit pro Sekunde in den SDRAM schreiben. Da der DMA-Controller maximal 300 MByte pro Sekunde übertragen kann, müsste das “Paket zu Stream“-Modul mit einem fest integrierten 100GBit-Ethernet-IP-Core kommunizieren [21, S. 9]. Das Modul müsste mehrere Pakete gleichzeitig verarbeiten und an den Ethernet-Core senden, was eine Überarbeitung des Automaten erfordert. Zusätzlich müssen die Funktionen im PS auf vier Timepix3-Chips angepasst werden. Dabei sollte die Systemkontrolle in der PL so erweitert werden, dass diese Operationen auf vier Timepix3-Chips anstatt auf einen Chip durchführt. Die Software auf dem Datenerfassungssystem dient ausschließlich für Entwicklungszecke und ist für die Verarbeitung von 20 GBit/s nicht geeignet. Um eine sichere Verarbeitung zu gewährleisten, müssten die Aufgaben auf mehrere Kerne des Prozessors aufgeteilt werden. Beispielsweise müssten mehrere Kerne die Decodierung der Datenpakete übernehmen.

7. Zusammenfassung und Ausblick

Durch die Entwicklung eines Timepix3 basierenden Auslesesystemes konnten erfolgreich erste Einblicke in die Funktionsweise von Timepix-Systemen gewonnen werden. Darüber hinaus wurden grundlegende Funktionen von Timepix3 durch die korrekte Aufnahme von kosmischer Strahlung erprobt und verifiziert. Dabei war es möglich, flexibel über das PS die Konfiguration sowie die allgemeine Kommunikation mit Timepix3 durchzuführen und zu testen. Infolgedessen konnten wichtige Informationen über die Funktionsweise der Peripherie, der Pixelmatrix und dem Pixelfrontend gewonnen werden. Darüber hinaus können Teile der Firmware, wie zum Beispiel die Datenwiedergabe, für weitere Timepix-Systeme übernommen werden. Zusätzlich konnten notwendige Systemoptimierungen in der bestehenden Hardware erkannt werden. Dabei hat es die bestehende Hardware erschwert, die Pakete im Datenstrom korrekt zu erkennen. Dennoch konnten mit der implementierten Paketerkennung sehr gute Werte erzielt werden, welche die Fehlerwahrscheinlichkeit auf unter ein Millionstel reduziert.

Für die weitere Entwicklung eines Timepix-Systemes ist eine Neuauflage der Hardware unumgänglich. Dabei werden die vorgeschlagenen Systemoptimierungen den Einstieg in die Entwicklung eines Timepix4 basierenden Auslesesystemes erleichtern. Jedoch sollte bevor mit der Entwicklung begonnen wird, die verifizierte Firmware dazu genutzt werden, die restlichen Funktionen von Timepix3 zu erproben. Dazu gehören beispielsweise die Messungen der Energie und der Ankunftszeit von eintreffenden Teilchen auf den Sensor. Um diese durchzuführen, muss das Pixelfrontend von Timepix3 weiter kalibriert und analysiert werden. Dadurch können weitere Erkenntnisse über Timepix3 gewonnen werden, die Änderung in einer Neuauflage der Hardware zur Folge haben könnten. Durch das gleichbleibende Kommunikationsprotokoll müssen die Module in der PL für andere Informationsmodi nicht abgeändert werden. Folglich kann Timepix3 über das PS des Zynqs schnell und flexibel umkonfiguriert werden, um weitere Informationsmodi zu testen.

Durch die begrenzende 1G-Ethernetverbindung wurde Timepix3 nicht mit 5 GBit/s betrieben. Dennoch ist es möglich, über kurze Zeit Timepix3 mit der maximalen Datenrate zu betreiben. Dafür müsste der Speicherbereich des DMA-Controllers erhöht werden, sodass Timepix3 für kurze Zeit den kompletten externen RAM beschreiben kann. Bei dieser Messung ist die Verwendung einer radioaktiven Quelle zwingend notwendig, da andere Strahlungsquellen nicht genügend Treffer erzeugen. Über ein solchen Test könnten ebenso neue Einblicke in Timepix gewonnen werden. Des Weiteren wurden im Rahmen dieser Arbeit lediglich sequentielle Aufnahmen durchgeführt, in welchen die Daten am Ende der Aufnahme ausgegeben werden. Ein weiterer Schritt wäre Timepix3 so zu konfigurieren, dass dieser die Daten während einer Aufnahme ausgibt. Dafür muss die Firmware im PS des Zynqs sowie die Software im Datenerfassungssystem in ihrem Kommunikationsablauf geringfügig modifiziert werden. Ein weiterer Eingriff in die Module der PL ist nicht nötig.

A. Anhang

A.1. Timepix3 Registereinstellung

Im Folgenden werden die Einstellungen für Register dargestellt, die nicht in den Standardeinstellungen betrieben werden. Für die genaue Funktion der Register wird auf das Datenblatt [2] verwiesen.

General Configuration Register

Wert[11:0]:
000000000100
Bemerkung:
Das analoge Pixelfrontend reagiert auf Löcher. Der Operationsmodus ist "Event and iTOT". Das Acknowledgement-Paket wird deaktiviert, da es sich in der Testphase als nicht aussagekräftiges Signal herausgestellt hat.

PLL Configuration Register

Wert[13:0]:
00000000000110
Bemerkung:
Die PLL mit wird dem internen Voltage-Controlled Oscillator (VCO) betrieben. Dabei wird sie im "single edge clock"-Modus betrieben. Ein Test mit "dual edge clock" konnte ebenso erfolgreich durchgeführt werden. Das Signal "PLLOut" ist deaktiviert, da es für die Entwicklung und Tests nicht benötigt wird. Es wurden Probleme mit dem "Clk_phaseShift_divider" festgestellt. Somit hat Timepix3 sporadisch bei Aufnahmen, wenn der Takt um weniger als 16 geteilt wurde, kein Ende-Paket gesendet.

Output Block Configuration Register

Wert[15:0]:
0011000100000001
Bemerkung:
Es ist ein Ausgangskanal mit 160 MHz aktiviert. Bei der Konfiguration des Ausgangstaktes muss darauf geachtet werden, in welchem Modus die PLL betrieben wird ("single edge" oder "dual edge"). 8B/10B-Codierung ist deaktiviert. "ClkOut" ist aktiviert.

SLVS Configuration Register

Wert[5:0]:
11000
Bemerkung:
Es wurde keine weiteren Tests mit den Stromstärken der SLVS-Treiber durchgeführt. Die 100 Ohm Terminierung von "ClkInRefPLL" wird aktiviert. Mehr dazu in Abschnitt A.3.

DAC Register

Ibias_PixelDAC Wert[7:0]:
00000000
Vthreshold_coarse Wert[3:0]:
0101
Vthreshold_fine Wert[8:0]:
100111011
Bemerkung:
Die analogen Spannungen beeinflussen abhängig von einander die Sensitivität des analogen Pixelfrontends. Siehe Kapitel 6.2.

DAC Register: Ibias_Ikrum

Wert[7:0]:
01110000
Bemerkung:
Siehe Kapitel 6.2.

Pixelregister

Wert[5:0]:
000000
000001 (maskiert)
Bemerkung:
Siehe Kapitel 6.2.

A.2. Aufnahmen

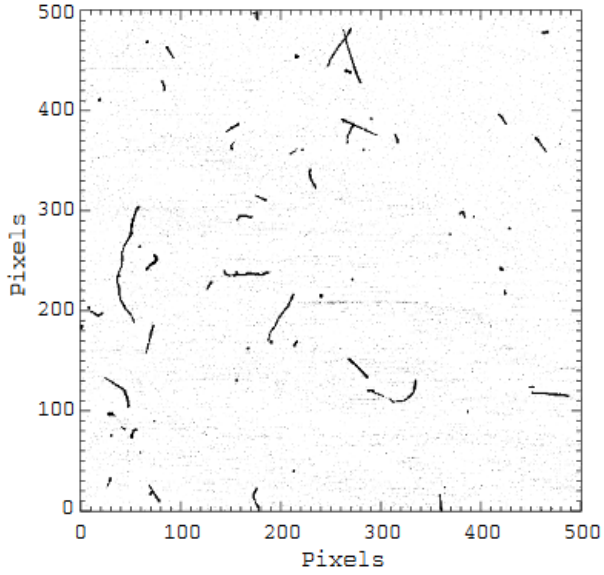

Abbildung A.1.: Referenz für kosmische Strahlung [5]

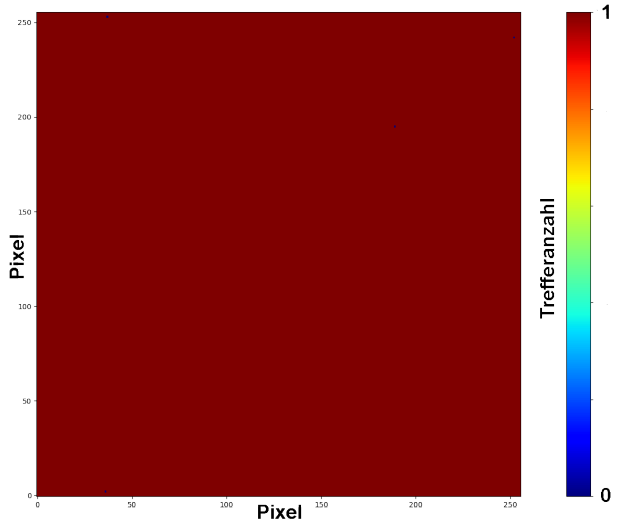

Abbildung A.2.: Aufnahme: keine weitere Konfiguration, 1 s

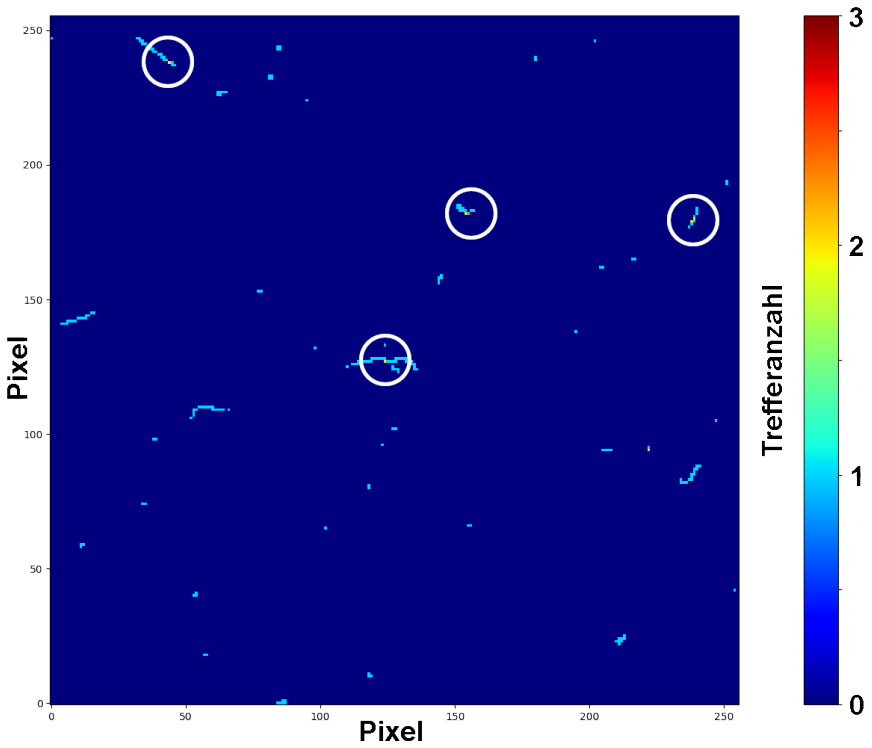

Abbildung A.3.: Aufnahme: Ikrum zu niedrig, 5 Minuten

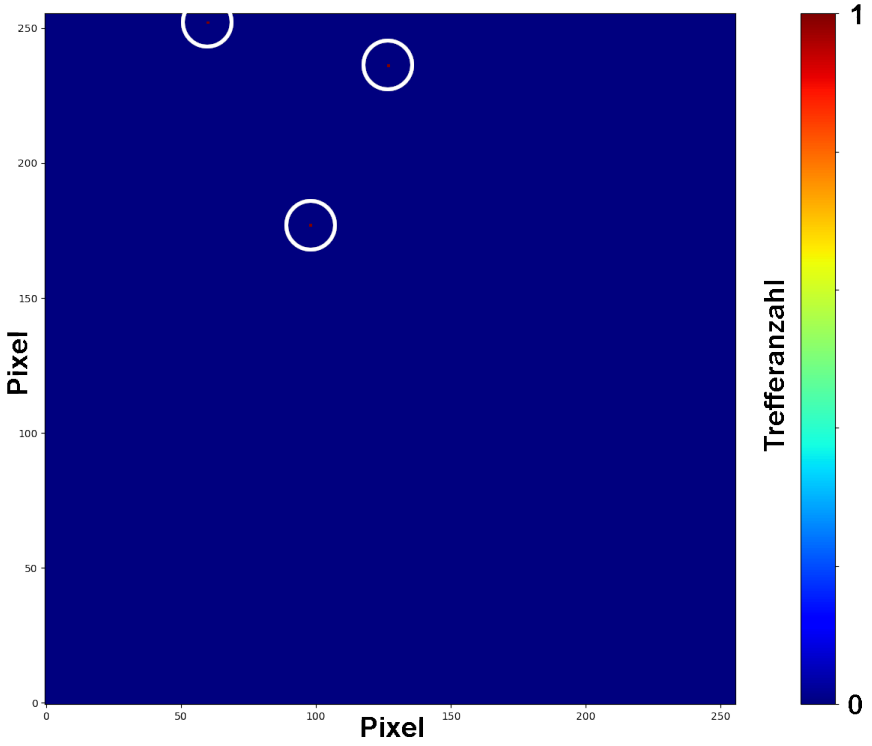

Abbildung A.4.: Aufnahme: Hot Pixel, 1 ms

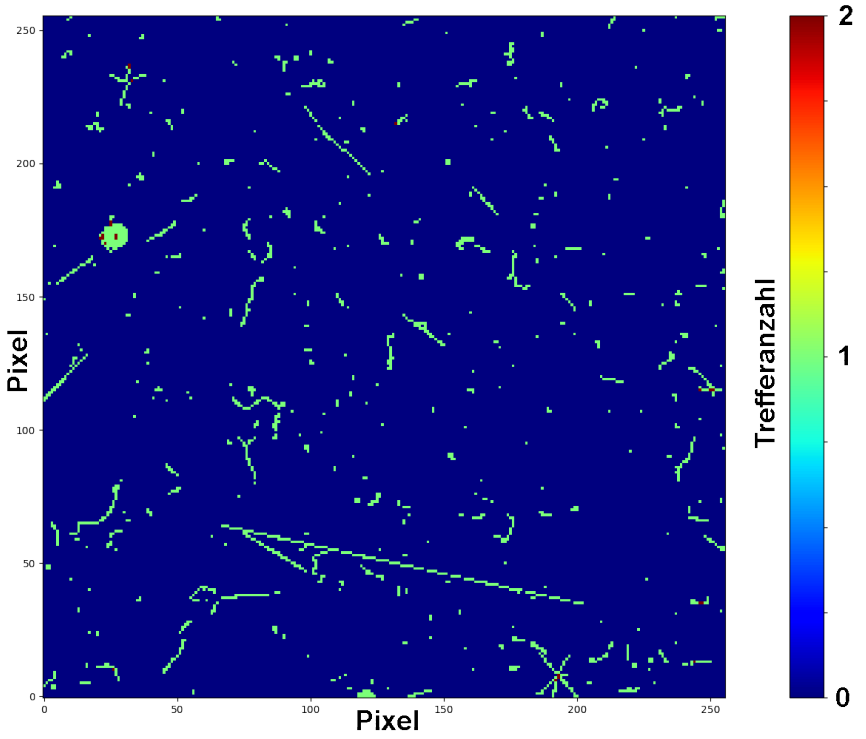

Abbildung A.5.: Aufnahme: Konfiguration abgeschlossen, 30 Minuten

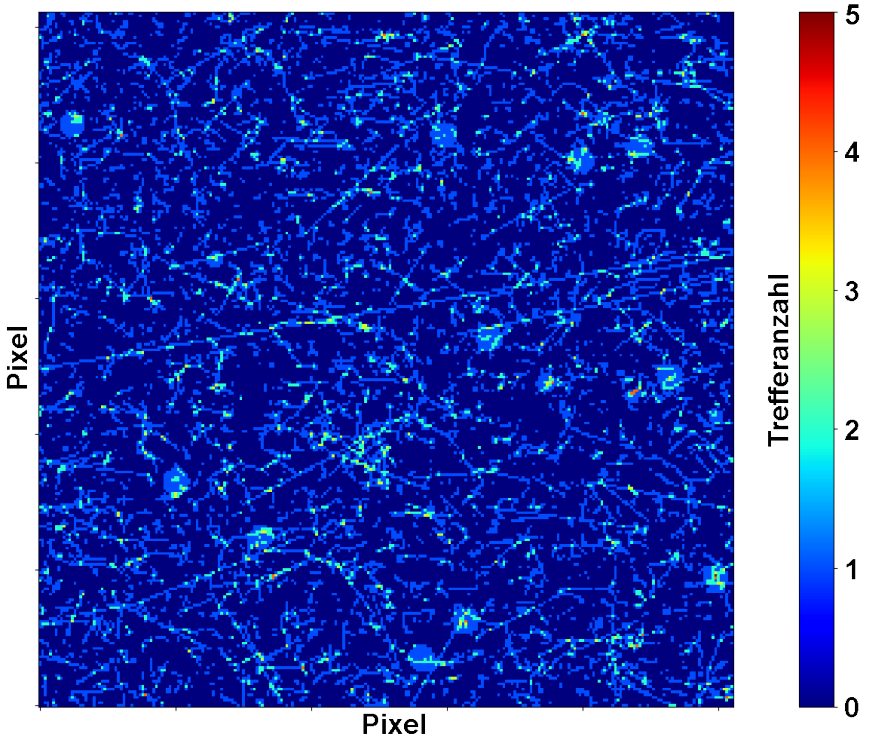

Abbildung A.6.: Aufnahme: Konfiguration abgeschlossen, 6 Stunden

A.3. Weitere Hardwareoptimierungen

- “ClkIn40“ und “ClkInRefPLL“ sollten elektronisch miteinander verbunden werden. Das spart Ressourcen und Taktbuffer. Beides sind Taktsignale und werden in der Regel mit einem Takt von 40 MHz betrieben. Die interne Terminierung, welche durch “SLVS_TERM“ aktiviert wird, betrifft nicht das Signal “ClkInRefPLL“. Das Signal “ClkInRefPLL“ kann separat über das “SLVS Configuration“-Register terminiert werden. Wenn beide Taktleitungen verbunden werden, muss die Terminierung für “ClkInRefPLL“ deaktiviert werden. Nachteil der Verbindung wäre, dass die interne PLL zwar noch überbrückt werden kann, jedoch nur mit einem Takt von 40 MHz.

- Das Signal “PLLOut“ muss nicht mit einem Taktbuffer des Zynqs verbunden werden. Mit dem Signal können interne Timepix3-Signale für Debug und Entwicklungszwecke ausgegeben werden. Dabei handelt es sich nicht um ein Taktsignal. Des Weiteren wurde das Signal in der Entwicklung nicht benötigt, weswegen es bei der Hardwareentwicklung keine hohe Priorität haben sollte.

- In der Hardware ist die Versorgungsspannung “VDDA33“ nicht angeschlossen. Für den Betrieb ist dies auch nicht nötig. Die getesteten Timepix3-Chips senden bei der Kommunikation alle mit der ID “0“. Soll die ID geändert werden, muss “VDDA33“ angeschlossen werden, um interne E-Fuses zu beschreiben. Dies ist nötig, wenn Teile der Datenleitung als Bus betrieben werden sollen.

- Bevor eine neue Hardwareentwicklung durchgeführt wird, sollte das Signal “T0_Sync“ getestet werden. Das Signal triggert bei der Messung der Ankunftszeit von eingetroffenen Teilchen den internen Zähler von Timpix3. Sollen sehr genaue Messungen mit externen Triggerereignis durchgeführt werden, wird eventuell eine externe Hardware zur Verarbeitung des Signales benötigt.

A.4. DVD Struktur

In Abbildung A.7 ist die Ordnerstruktur der beigelegten DVD dargestellt. Da die Vivado-Projektstruktur ein wenig unübersichtlich ist, sind die VHDL-Codes im Ordner "VHDL" noch einmal zusammengefasst. Das Blockdiagramm, sowie die konfigurierten IP-Cores können aus der Vivado-Projektstruktur entnommen werden.

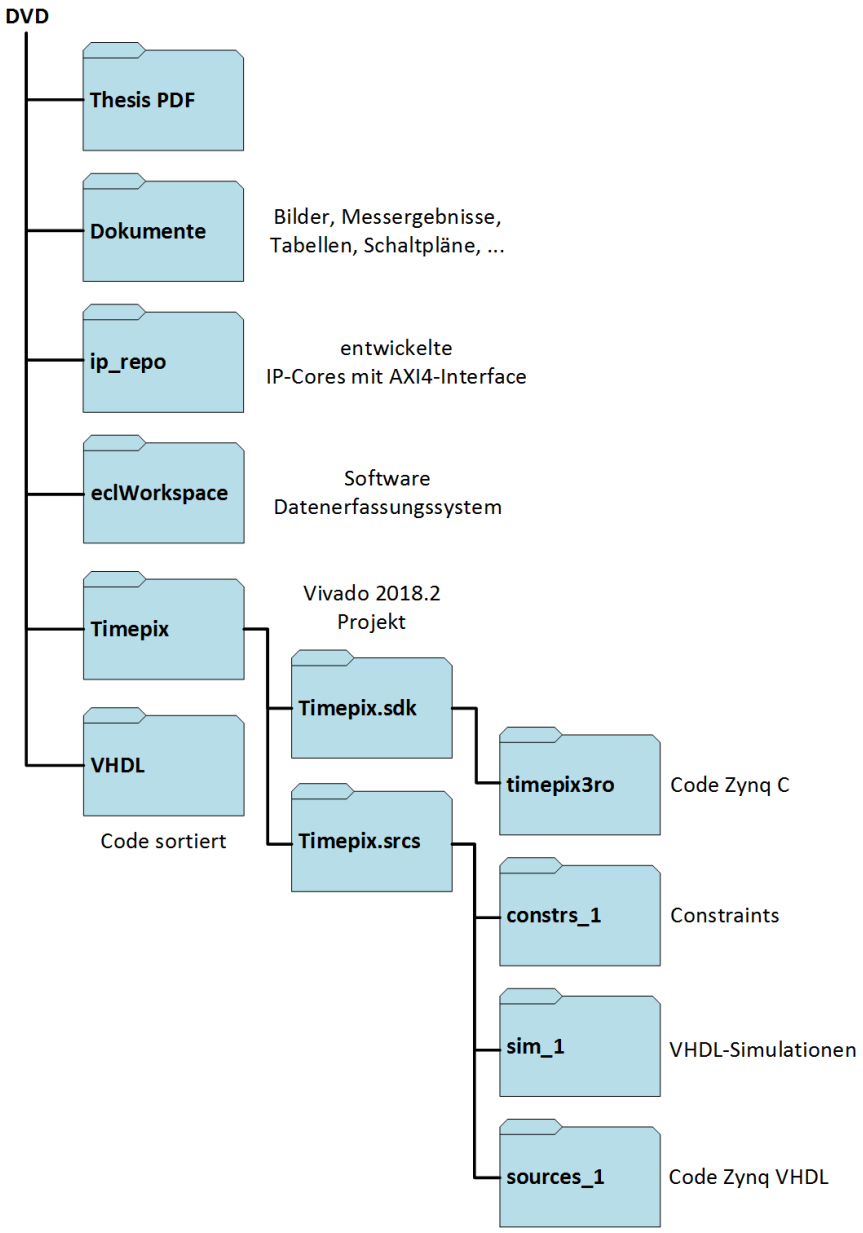

Abbildung A.7.: DVD Struktur

A.5. Sonstiges

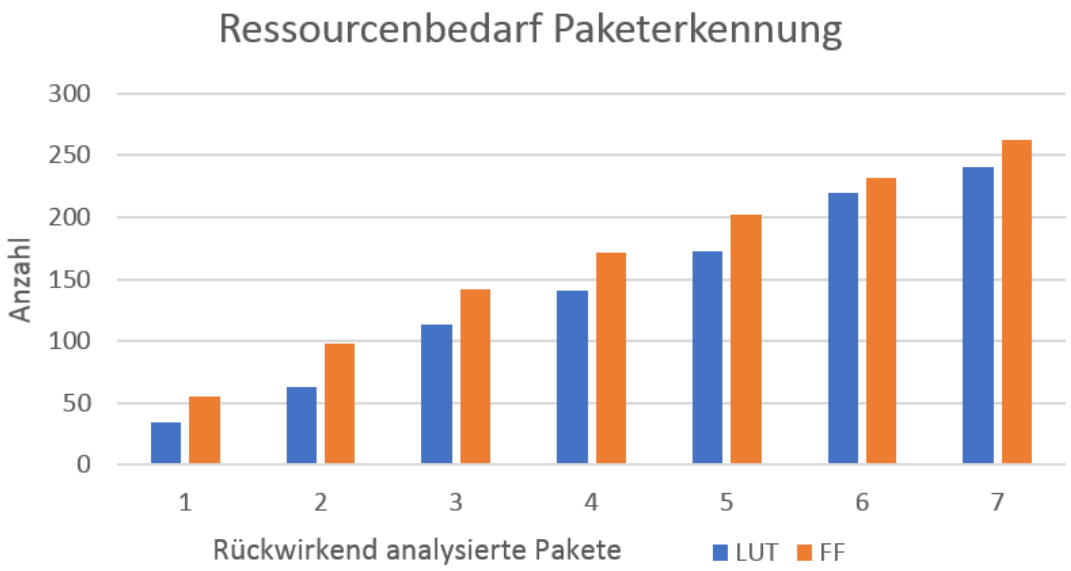

Abbildung A.8.: Ressourcenbedarf Paketerkennung

Literaturverzeichnis

[1] S. Smoljanin, “Entwicklung der firmware eines auslese- und steuerungssystems für das "large area medipix based detektor array",” Diplomarbeit, Hochschule Mittweida, Deutschland, 2013.

[2] X. Llopart und T. Poikela, “Timepix3 manual v1.9,” März 2014, [Zugriff am 15.02.2019]. [Online]. Available: https://twiki.cern.ch/twiki/pub/CLIC/LabTestingTimepix3/timepix3_manual.pdf

[3] *AXI4-Stream FIFO v4.1*, PG080, Xilinx Inc., April 2016.

[4] “How to handle hubble images,” April 2010, [Zugriff am 02.03.2019]. [Online]. Available: https://blog.galaxyzoo.org/2010/04/12/how-to-handle-hubble-images/

[5] A. Smith, R. McDonald, D. Hurley, S. Holland, D. Groom, W. Brown, D. Gilmore, R. Stover, and M. Wei, “Radiation events in astronomical ccd images,” 12 2001, [Zugriff am 02.03.2019].

[6] *Forschung für die Zukunft*, DESY, [Zugriff am 03.01.2019]. [Online]. Available: http://pr.desy.de/e143921/index_ger.html

[7] *Ziel der Grundlagenforschung*, Max-Planck-Institut für biologische Kybernetik, [Zugriff am 03.01.2019]. [Online]. Available: http://hirnforschung.kyb.mpg.de/hirnforschung/ziel-der-grundlagenforschung.html

[8] *Beschleuniger - Rennmaschinen für Höchstleistungen*, DESY, [Zugriff am 03.01.2019]. [Online]. Available: http://www.desy.de/ueber_desy/desy/forschung_fuer_die_zukunft/index_ger.html

[9] Bundesamt für Strahlenschutz, “Ionisierende Strahlung,” [Zugriff am 01.03.2019]. [Online]. Available: http://www.bfs.de/DE/themen/ion/ion_node.html

[10] C.Brezina, Y.Fu, M.De Gaspari, V.Gromov, X.Llopart, T.Poikela, F.Zappon, A.Kruth, *The Timepix3 Chip*, Universität Bonn, CERN, NIKHEF, [Zugriff am 22.01.2019]. [Online]. Available: https://indico.cern.ch/event/267425/attachments/477859/661149/Timepix3_final.pdf

[11] *Zynq UltraScale+ MPSoC Product Tables and Product Selection Guide*, Xilinx Inc., [Zugriff am 10.01.2019]. [Online]. Available: https://www.xilinx.com/support/documentation/selection-guides/zynq-ultrascale-plus-product-selection-guide.pdf#CG

[12] C. Kreidl, "Bumping and flip-chip of asics," 2017, [Zugriff am 17.01.2019].

[13] R. Ballabriga, J. Alozy, M. Campbell, E. Frojdh, E. Heijne, T. Koenig, X. Llopart, J. Marchal, D. Pennicard, T. Poikela, L. Tlustos, P. Valerio, W. Wong, and M. Zuber, "Review of hybrid pixel detector readout ASICs for spectroscopic x-ray imaging," *Journal of Instrumentation*, vol. 11, no. 01, pp. P01 007–P01 007, jan 2016, [Zugriff am 18.01.2019]. [Online]. Available: https://doi.org/10.1088%2F1748-0221%2F11%2F01%2Fp01007

[14] M. D. Gaspari, J. Alozy, R. Ballabriga, M. Campbell, E. Fröjdh, J. Idarraga, S. Kulis, X. Llopart, T. Poikela, P. Valerio, and W. Wong, "Design of the analog front-end for the timepix3 and smallpix hybrid pixel detectors in 130 nm CMOS technology," *Journal of Instrumentation*, vol. 9, no. 01, pp. C01 037–C01 037, jan 2014, [Zugriff am 02.02.2019]. [Online]. Available: https://doi.org/10.1088%2F1748-0221%2F9%2F01%2Fc01037

[15] E. Monteil, L. Pacher, A. Paternò, F. Loddo, N. Demaria, L. Gaioni, F. D. Canio, G. Traversi, V. Re, L. Ratti, A. Rivetti, M. D. R. Rolo, G. Dellacasa, G. Mazza, C. Marzocca, F. Licciulli, F. Ciciriello, S. Marconi, P. Placidi, G. Magazzù, A. Stabile, S. Mattiazzo, and C. Veri, "A prototype of a new generation readout ASIC in 65nm CMOS for pixel detectors at HL-LHC," *Journal of Instrumentation*, vol. 11, no. 12, pp. C12 044–C12 044, dec 2016, [Zugriff am 03.02.2019]. [Online]. Available: https://doi.org/10.1088%2F1748-0221%2F11%2F12%2Fc12044

[16] F. Tavernier und P. Moreira, "SLVS interface circuits," [Zugriff am 15.01.2019]. [Online]. Available: https://indico.cern.ch/event/404345/contributions/1850202/attachments/809975/1109940/SLVS_IO.pdf

[17] Y. Sheynin, F. Shutenko, E. Suvorova, and E. Yablokov, "High-rate serial interconnections for embedded and distributed systems with power and resource constraints," *Proc SPIE*, 05 2008, [Zugriff am 17.01.2019].

[18] *Zynq UltraScale+ MPSoC Data Sheet Overview*, DS891, Xilinx Inc., November 2018, v1.7.

[19] *Zynq UltraScale+ MPSoC Software Developer Guide*, UG1137, Xilinx Inc., Juni 2018, v8.0.

[20] *AXI Reference Guide*, UG761, Xilinx Inc., Januar 2012, v13.4.

[21] *AXI DMA v7.1*, PG021, Xilinx Inc., April 2018.

[22] R. Nishimura, Y. Arai, T. Miyoshi, K. Hirano, S. Kishimoto, R. Hashimoto, Y. Lu, L. Song, and Q. Ouyang, "Development of a new high-speed readout system for soi pixel detectors," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 10 2018, [Zugriff am 23.01.2019].

[23] J. Yiu, "A beginner's guide on interrupt latency - and interrupt latency of the arm cortex-m processors," *Journal of Instrumentation*, 2016, [Zugriff am 13.02.2019]. [Online]. Available: https://community.arm.com/processors/b/blog/posts/beginner-guide-on-interrupt-latency-and-interrupt-latency-of-the-arm-cortex-m-processors

[24] *Ultrascale Architecture SelectIO Resources*, UG571, Xilinx Inc., Juni 2018, v1.9.

[25] *Ultrascale Architecture GTH Tranceivers*, UG576, Xilinx Inc., August 2018, v1.5.1.

[26] *Ultrascale Architecture GTY Tranceivers*, UG578, Xilinx Inc., September 2017, v1.3.

[27] *Designing Using SelectIO Interface Component Primitives*, XAPP1324, Xilinx Inc., Jim Tatsukawa, August 2018, v1.1.

[28] *Bitslip in Logic*, XAPP1208, Xilinx Inc., Marc Defossez, Mai 2014, v1.0.

[29] *AXI Block RAM (BRAM) Controller v4.0*, PG078, Xilinx Inc., Oktober 2016.

[30] *RAM-Based Shift Register v12.0*, PG122, Xilinx Inc., November 2015.

[31] Dr. Michael Zapf, "Techniken und Dienste des Internets," [Zugriff am 04.02.2019]. [Online]. Available: http://ls4-www.cs.tu-dortmund.de/Lehre/07-42351/TDI_03_TCP_UDP_HTTP.pdf

[32] *AXI GPIO v2.0*, PG144, Xilinx Inc., Oktober 2016.

[33] *Vivado Design Suite User Guide, Using Constraints*, UG903, Xilinx Inc., Juni 2018, v2018.2.

[34] *Clocking Wizard v5.3*, PG065, Xilinx Inc., Oktober 2016.

[35] L. Winkler, "Internet(2) transportdienste und -protokolle," *Hochschule Mittweida*, September 2011, [Zugriff am 20.02.2019].

[36] *Ultrascale+ Devices Integrated 100G Ethernet Subsystem v2.4*, PG203, Xilinx Inc., April 2018.

[37] *Zynq UltraScale+ MPSoC Product Tables and Product Selection Guide*, Xilinx Inc., [Zugriff am 28.02.2019]. [Online]. Available: https://www.xilinx.com/support/documentation/selection-guides/zynq-ultrascale-plus-product-selection-guide.pdf#EG

[38] *Zynq UltraScale+ FPGAs Product Tables and Product Selection Guide*, Xilinx Inc., [Zugriff am 28.02.2019]. [Online]. Available: https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf

Abkürzungsverzeichnis

FPGA Field Programmable Gate Array 10

SLVS Scalable Low Voltage Signaling 16

LVDS Low Voltage Differential Signaling 16

DDR Double Data Rate 15

SDR Single Data Rate 30

PLL Phase-Locked Loop 14

PL Programmable Logic 21

PS Processing System 21

DMA Direct Memory Access 22

FIFO First In First Out 21

SDRAM Synchronous Dynamic Random-Access Memory 11

SoC System On Chip 11

DAC Digital-to-Analog Converter 14

IO Input Output 21

GPIO General-Purpose Input/Output 44

FS-DS Photon-Science Detector-Group 10

VCO Voltage-Controlled Oscillator 69

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Konzept und Entwicklung eines Hochgeschwindigkeits-Auslesesystems für einen Röntgenstrahlendetektor auf einem SoC

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original