Studien zur Realisierung der Feinstleiterplatine für das zukünftige ATLAS-Pixeldetektormodul

Bachelor-Arbeit im Studiengang Physik
zur Erlangung des akademischen Grades
Bachelor of Science
(B.Sc)

der Universität Siegen

Department Physik

Vorgelegt von
Nico Malinowski (1166671)
Erstgutachter: Prof. Dr. Peter Buchholz

17. November 2020

Inhaltsverzeichnis

1. Einleitung **1**

2. Der Large Hadron Collider, ATLAS und der Inner-Detector **1**
2.1. Der Large Hadron Collider 1
2.2. Der ATLAS Detektor 2
2.3. Der Inner Detector und der Pixeldetektor 3

3. Das Quad-Modul und der FE-I4B-Chip **6**

4. Das Testsystem **8**
4.1. Das Kühlsystem 9
4.2. Die USBPix3-Software 10
4.2.1. Die Testreihenvorbereitung 12
4.3. Die Testprozedur 12
4.3.1. Initialisierung 12
4.3.2. Test der digitalen Chipelektronik 12
4.3.3. Test der analogen Chipelektronik 13
4.3.4. Angleichen der Pixelmatrixantwort 14
4.3.5. Die automatisierten Maskierungs-Skripte 15
4.3.6. Überprüfen der Pixelmatrixantwort 15
4.3.7. Test des Sensors 16
4.3.8. Messen der Sperrstrom-Sperrspannungs-Kennlinie 17

5. Die Datenanalyse **18**
5.1. Der Dateneingang 18
5.2. Die Analyseschritte 19

6. Ergebnisse **21**
6.1. Wiederverbindung 4 22
6.2. Wiederverbindung 5 23
6.3. Wiederverbindung 6 24
6.4. Wiederverbindung 7 25
6.5. Wiederverbindung 8 26
6.6. Wiederverbindung 9 27
6.7. Wiederverbindung 10 28
6.8. Interpretation 29

7. Zusammenfassung **33**

8. Danksagung **34**

A. Anhang **38**
A.1. Globale Konfiguration 38

A.2. scan_digital.py 39
A.3. scan_analog.py 39
A.4. tune_fei4.py 40
A.5. scan_threshold.py 40
A.6. scan_fei4_self_trigger.py 42
A.7. scan_iv.py 43
A.8. Pythonmodule 44
A.9. Config.py 44
A.10.Erstverbindung 0 45
A.11.Wiederverbindung 1 46
A.12.Wiederverbindung 2 47
A.13.Wiederverbindung 3 48

B. Selbstständigkeitserklärung **62**

1. Einleitung

Um die fundamentalen Wechselwirkungen von Teilchen zu untersuchen, benötigt es leistungsstarke Beschleuniger und hochauflösende Detektoren. Eines dieser Experimente befindet sich am europäischen Zentrum für Teilchenphysik (CERN) an der Grenze zwischen Frankreich und der Schweiz. Der ATLAS-Detektor [3] ist einer der zwei Mehrzweckdetektoren des Large Hadron Colliders (LHC) [6] (siehe Kap. 2). Nach Ende der dritten Datennahmephase (LHC Run 3. Plan 2021-2024) des LHCs wird dieser einem Upgrade unterzogen, um höhere Kollisionsraten zu erreichen. Die Phase nach dem Upgrade wird als High-Luminosity LHC [11] bezeichnet. Aufgrund der höheren Ereignisraten, welche zu Aufhäufung von Spuren und einer stärkeren Strahlungsumgebung im Detektor führen, und Strahlenbeschädigungen aus den vorherigen Runs, muss der gesamte Inner-Detector [1] ersetzt werden. Der neue Inner-Tracker [9] wird ein reiner Silizium-Detektor sein und ist in zwei Untersysteme aufgeteilt: dem Silizium-Stripdetektor und dem Pixeldetektor. Der Pixeldetektor hat eine modulare Bauweise und jedes der Module besteht aus einem Silizium-Sensor, mindestens einem Front End Chip und einer Feinstleiterplatine (siehe Kap. 3). Um die Produktion der Module möglichst einfach zu gestalten, wurde beschlossen, den Steckverbinder, der die Module mit den aus dem Detektor führenden Systemen verbindet, über dem Sensor anzubringen. Dies wirft die Frage auf, ob das Öffnen und Schließen dieses Steckverbinders zu Beschädigungen am Modul führt. Um diese Frage zu beantworten, wurde nach jedem Verbindungsvorgang des Steckverbinders eine Reihe an Funktionstests an einem Prototyp-Modul durch geführt. (siehe Kap. 4) Die Ergebnisse dieser Testreihen wurden mittels eines Skripts ausgewertet. (siehe Kap. 5)

2. Der Large Hadron Collider, ATLAS und der Inner-Detector

Der Large Hadron Collider [6] (LHC) ist der leistungsstärkste Teilchenbeschleuniger der Welt. Betrieben wird er vom Europäischen Zentrum für Teilchenphysik (CERN). Gegründet wurde das CERN 1954 und liegt auf beiden Seiten der Grenze zwischen der Schweiz und Frankreich nahe Genf. Das CERN hat 23 Mitgliedsstaaten. Deutschland ist eines der 12 Gründerländer. Insgesamt arbeiten ca. 2 500 Menschen als Mitarbeiter am CERN. Ihre Hauptaufgaben sind die Entwicklung, Vorbereitung und Wartung der Experimente sowie die Unterstützung der ca. 12 200 Wissenschaftler, die als Nutzer CERNs an Bau, Betrieb und Datenanalyse der Experimente beteiligt sind.

2.1. Der Large Hadron Collider

Der LHC ist ein Hadronen-Ringbeschleuniger [6]. Er bringt Protonen an den Kreuzpunkten der gegenläufigen Strahlen zur Kollision. Es können auch schwere Ionen im LHC beschleunigt werden. Die tatsächlich zur Verfügung stehende Kollisionsenergie variiert, da der Elementarprozess zwischen den Quarks und Gluonen der Protonen stattfindet. Sie tragen nur einen durch die Parton Distribution Function beschriebenen Anteil

am Gesamtimpuls des Protons. Diese Variation der Energie führt dazu, dass alle teilchenphysikalischen Prozesse, die möglich sind, stattfinden, weshalb der LHC auch als Entdeckungsmaschine bezeichnet wird. Der Beschleunigerring hat einen Umfang von 27 km und besteht aus 1 232, 15 m langen Dipol-Magneten, die die Protonenstrahlen auf der Kreisbahn halten, und 392, 5-7 m Quadrupol-Magneten, die der Fokussierung der Strahlen dienen. Die Magnete müssen auf -271,3 °C gekühlt werden, um die benötigten Magnetfelder kosteneffizient zu erzeugen, wobei der elektrische Widerstand aufgrund der Supraleitung verschwindet. Es wird suprafluides Helium benutzt, um diese Temperaturen zu erreichen. An vier Positionen im Beschleunigerring werden die Strahlen zu Kreuzung und zur Kollision gebracht. An diesen Positionen sitzen die vier großen Detektoren ALICE, ATLAS, CMS und LHCb. Die Kollisionenergie der Teilchen in den Detektoren soll im LHC Run 3 (2022-2024) maximal 14 TeV erreichen.

2.2. Der ATLAS Detektor

Der ATLAS(A Toroidal LHC Apparatus)-Detektor [3] ist einer von zwei Mehrzweck-Detektoren am LHC. Der zweite Mehrzweck-Detektor ist der Compact Myon Solenoid (CMS)-Detektor [8]. Die beiden Detektoren unterscheiden sich im wesentlichen durch das Magnetsystem und das Myondetektionssystem. Die Daten, die beide Detektoren sammeln, werden benutzt, um offene Fragen der Teilchenphysik zu klären. So wurde 2012 das Higgs-Boson [2] gefunden. Weiter werden die Daten benutzt, um nach möglicherweise existierenden/ aus bekannten Theorien vorhergesagten Teilchen z. B. der dunklen Materie oder supersymmetrischen Teilchen zu suchen. Der ATLAS Detektor befindet sich 100 m unter der Erdoberfläche und ist 44 m lang und 25 m hoch und breit. Der Detektor besteht aus 6 verschiedenen Detektorsystemen, die in Schichten um den Interaktionspunkt angeordnet sind und zusammen Ort, Energie und Impuls der resultierenden Teilchen ermitteln. Ein Auslösesystem basierend auf unverarbeiteten Detektordaten macht es möglich, die Datenmengen, die bei der Kollision der Teilchen entstehen, auch auszulesen. Es werden nur Daten gespeichert, wenn das Auslösesystem ein Ereignis im Detektor als wichtig einstuft, z. B. aus der Anzahl an erzeugten Myonen. Die Schichten des ATLAS-Detektors sind vom Kollisionspunkt ausgehend: Der Inner Detector, mit seinen 3 Subsystemen, das Kalorimetersystem, das Magnetsystem und das Myon-Spektrometer.

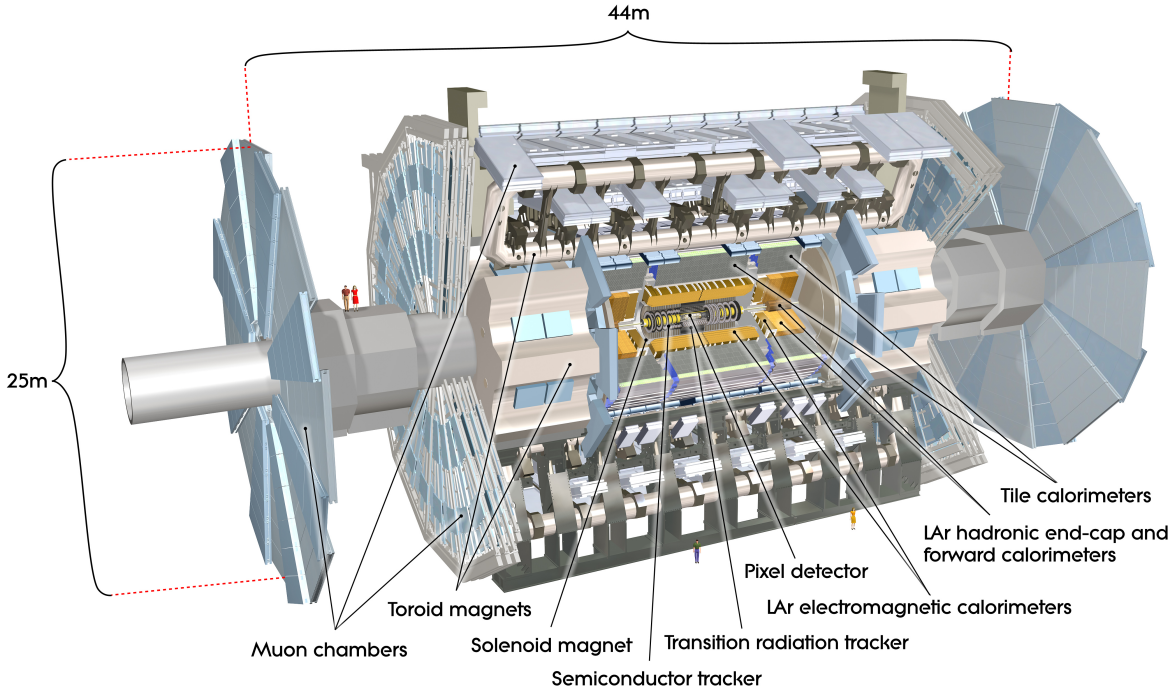

Abbildung 2.2.1: Schematische Übersicht der Komponenten des ATLAS-Detektors [3]

2.3. Der Inner Detector und der Pixeldetektor

Der Inner Detector [1] hat die Aufgabe, den Impuls geladener Teilchen präzise zu vermessen. Dazu werden die geladenen Teilchen, welche bei den Proton-Proton-Kollisionen entstehen, in einem solenoiden Magnetfeld abgelenkt. Durch Vermessen ihrer Flugbahn kann die Krümmung bestimmt und daraus der Impuls der Teilchen berechnet werden. Zur genauen Bestimmung der Flugbahn benutzt der Inner Detector verschiedene Detektortypen (Abb. 2.3.1) . Vier Lagen Pixeldetektor werden gefolgt von vier Lagen Semiconductor Tracker (SCT). Abgeschlossen wird der Inner Detector vom Transition Radiation Tracker (TRT). Der TRT detektiert Teilchen mit Hilfe der Gasionisation. Die einzelnen Kanäle sind gasgefüllte Rohre mit einem goldüberzogenen Wolframdraht in der Mitte. Diese Röhren geben dem TRT eine Auflösung von 0.17 mm transversal zu den Röhren. Der Semiconductor Tracker benutzt Siliziumsensoren (genauere Beschreibung der Funktionsweise in Abs. 3), die nur in einer Dimension unterteilt sind. Dies verringert die Anzahl der benötigten Auslesekanäle im Vergleich zum Pixeldetektor. Die größeren Streifengrößen sorgen dafür, dass eine größere Fläche immer noch effektiv mit Siliziumsensoren abgedeckt werden kann. Der SCT hat eine Auflösung von 80 μm transversal zu der Richtung der Streifen. Der SCT besitzt insgesamt eine Fläche von 60 m^2 Silizum aufgeteilt in 4 Lagen und 18 Endringe.

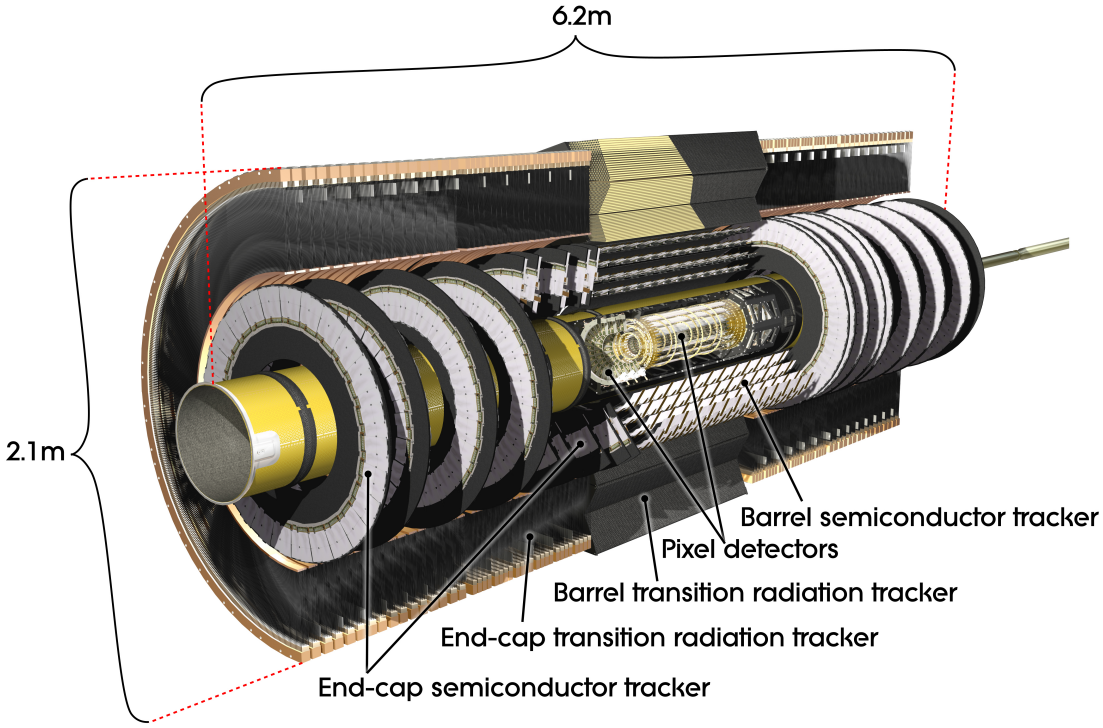

Abbildung 2.3.1: Schematische Darstellung des Inner-Detectors des ATLAS-Detektors [3]

Der Silizium-Pixeldetektor hat die höchste räumliche Auflösung. Die Pixelgröße beträgt 50μm x 400μm . Er hat eine aktive Fläche von 1.7 m^2 aufgeteilt in 3 Lagen und 6 Endkappen. Die nach außen folgenden Detektortypen (SC,TRT) besitzen weniger hohe Auflösungen, da sie einen größeren Abstand zum Kollisionspunkt haben und so größere Flächen abdecken müssen. Die aktive Fläche des Pixeldetektors besteht aus 1 744 Modulen (Abb.2.3.2) mit je 10 cm^2 Fläche. Ein Modul besteht aus einem Siliziumsensor, der von Auslesechips ausgelesen wird und einer flexiblen Leiterplatine, auf der sich der Module Control Chip (MCC) befindet. Die Leiterplatine zusammen mit dem MCC leiten die Daten des Moduls zu den aus dem Detektor führenden Systemen weiter. Da der Inner Detector bei Betrieb des LHC hoher Teilchenstrahlung ausgesetzt [4] ist, hat er nur eine begrenzte Lebensdauer. Deshalb wurde 2014 eine vierte Lage mit kleinerem Radius in den Pixeldetektor eingefügt [7]. Der Insertable-B-Layer (IBL) sollte dabei sicherstellen, dass die Detektionseffizenz von Teilchenspuren am Ende der Lebensdauer des Pixeldetektors noch ausreichend ist. Der IBL benutzt ein anders Modul-Design (Abb. 2.3.3) als der restliche Pixeldetektor. Der Sensor ist kleiner und es werden nur zwei Auslesechips benutzt, weshalb dieser Typ auch als Dual-Modul bezeichnet wird. Die Auslesechips sind vom Typ FE-I4B und werden im Kapitel 3 genauer beschrieben.
Der gesamte Inner Detector muss, um den Spezifikationen des High Luminosity LHC

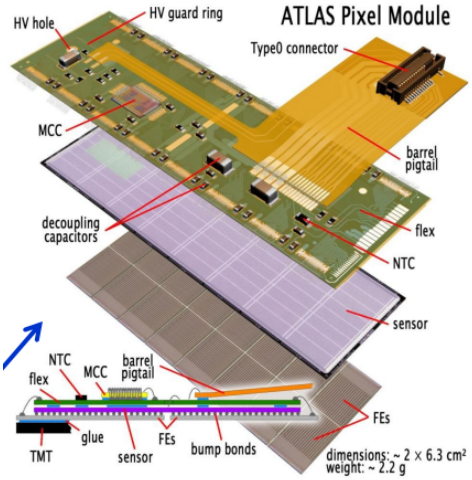

Abbildung 2.3.2: Modulkonzept des ATLAS Pixeldetektors [13]

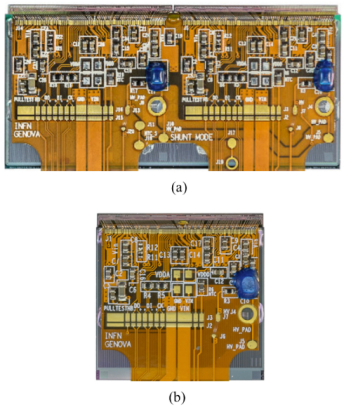

Abbildung 2.3.3: IBL Modul in Dual- (a) und Single-Chip (b) Variante [13]

Upgrades zu entsprechen, parallel zum Upgrade des Beschleunigers ausgetauscht werden. Der neue Inner Tracker soll vollständig aus Siliziumsensoren aufgebaut werden, wobei sowohl der neue Pixeldetektor als auch der neue Streifendetektor zusammen den momentanen Inner Detector ersetzen werden. Das Konzept des neuen Pixeldetektors erfordert 3 Typen von Modulen: Single- und Dual-Modul sowie Quad-Modul, welche im nächsten Kapitel genauer erläutert werden.

3. Das Quad-Modul und der FE-I4B-Chip

Ein Modul (siehe Abschnitt 2.3) des zukünftigen ATLAS-Pixeldetektors [10] wird als Quad-Modul bezeichnet, wenn es einen Sensor mit vier Auslesechips besitzt. Weitere Modultypen sind Dual-Module, mit zwei Chips, und Single-Module, mit nur einem Chip. Das Quad-Modul besteht aus drei verschiedenen Komponenten.

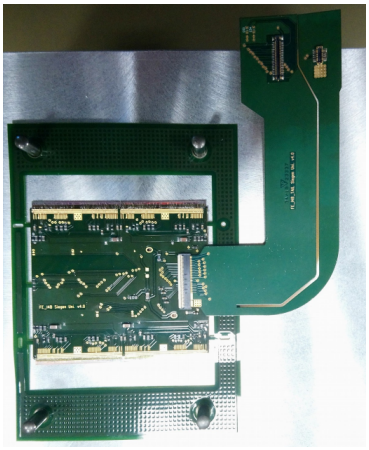

Abbildung 3.0.1: Ansicht eines Quad-Moduls von oben [12]

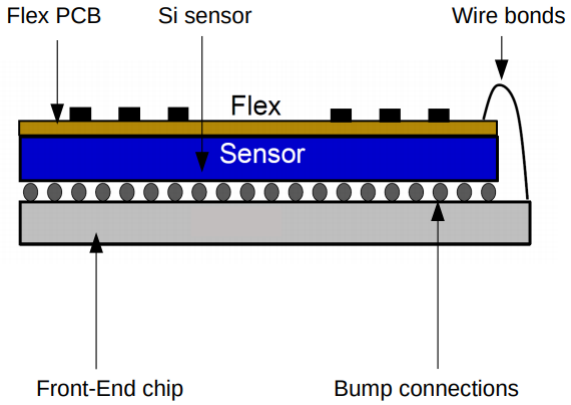

Abbildung 3.0.2: Modulkonzept dargestellt für einen Chip [12]

Von oben nach unten (Abb.3.0.2) aufgeführt sind die Komponenten : Die Feinstleiterplatine, der Siliziumsensor und die vier Auslesechips. Der Sensor basiert auf dem Prinzip einer PIN-Diode, die in Sperrrichtung betrieben wird (Abb. 3.0.3). Die durch Festkörperionisation entstandenen Elektronen driften im elektrischen Feld zu den p^+ dotierten Flächen des Sensors, welche mit den Kontaktflächen des Chips verbunden sind. Auf die Funktion der Auslesechips wird im nächsten Abschnitt genauer eingegangen. Die Leiterplatine besteht aus flexiblem Polyimid, auf welches beidseitig Kupferleiterbahnen aufgebracht sind. Die Leiterbahnen sind mittels Lötstopplack versiegelt und elektrisch isoliert. Auf der Oberseite der Platine befinden sich verschiedene passive Komponenten, die zur Stromversorgung und dem Signaltransfer der FE-I4B Chips und des Sensors dienen. Des weiteren befindet sich der Steckverbinder für den abnehmbaren flexiblen Adapter in der Mitte des rechten Randes der flexiblen Leiterplatine (Flex). Der Flex ist mittels Epoxyd-Harz-Kleber und Klebe-Transfer-Band auf dem Sensor befestigt. Die elektrischen Verbindungen zwischen Flex und Sensor, sowie den FE-I4B Chips erfolgen über Aluminiumdrähte mit 25 μm Durchmesser. Der Sensor ist mittels Bump-Bonds mit den Auslesechips verbunden. Die Widerstandsfähigkeit der Bump-Bonds bezüglich mechanischen Drucks wird in dieser Arbeit untersucht. Eine mechanische Belastung der Bump-Bonds kann durch Schließen des erwähnten Steckverbinders entstehen.

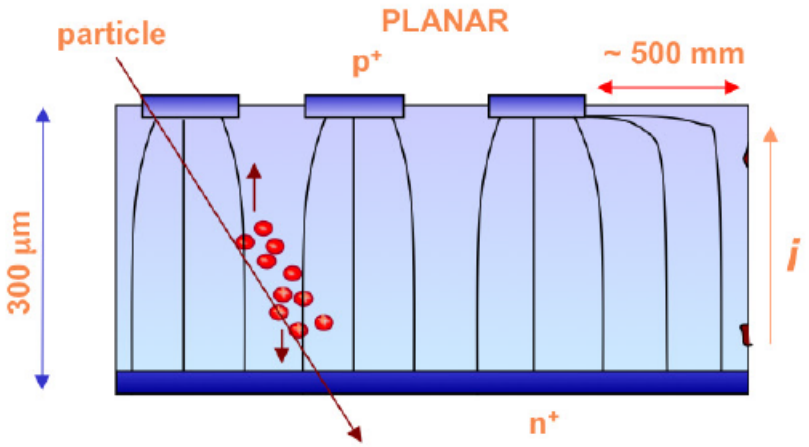

Abbildung 3.0.3: Schematische Darstellung der Siliziumsensortypen [10]

Der FE-I4B-Chip

Der FE-I4B-Pixel-Auslesechip [5] [7] ist eine Revision des FE-I4A-Chips. Die B-Variante ist für die Produktion des Insertable B-Layer (IBL) des ATLAS-Pixeldetektors entwickelt und benutzt worden . Der Chip besitzt 26 880 Pixel, aufgeteilt in 80 Spalten mit 250 μm Breite und 336 Zeilen mit 50 μm Breite. Der Chip wurde mittels 130 nm CMOS Technologie erstellt. In Abbildung 3.0.4 ist der schematische Ablauf des Datennahme Prozesses des FE-I4 Chips zu sehen. Der Chip erhält von außen Befehle, die im Kommandodekodierer (CMD) verarbeitet und von der End Of Chip Logic (EOCHL) ausgeführt werden. Gleichzeitig wird der Chip mit einer externen 40 MHz-Taktung versorgt. Der Chip synchronisiert die interne Uhr (PLL) mit dieser Taktung und die PLL gibt ein regeneriertes 40 MHz-Signal an die Chiplogik, sowie ein 160 MHz-Signal an den Datenausgang (DOB) weiter, welches den Ausgangsdatenstrom antreibt. Die Pixelmatrix des Chips ist in 40 digitale Doppelspalten (DDC) aufgeteilt. Jede DDC ist mit 2 Spalten analoger Pixelregion (A) verbunden. Die Doppelspalten (DDC und 2 A) können in Logikeinheiten von 2x2 Pixeln aufgeteilt werden, wobei der DDC-Anteil der 4 Pixel 20 Speicherregister (5 pro Pixel) für die aktive Zeit (Time Over Threshold, TOT) der analogen Pixel enthält. Diese Speicherregister speichern die TOT für maximal 255 Taktschritte und werden, wenn sie nicht ausgelesen werden und alle 5 Speicherzellen eines Pixel benutzt wurden, dem Alter nach überschrieben. Das Pixel in der analogen Pixelregion enthält die Kontaktflächen zum Sensor, sowie eine einstellbare doppelte Verstärkerstufe und einen einstellbaren Diskriminator, der sein Ergebnis an die DDC weitergibt. Das Auslesen der in den DDC gespeicherten Daten erfolgt über ein Signal, das von der EOCHL erzeugt und durch externen Befehl via CMD ausgelöst wird. Das Signal wird von der End of Doublecolumn

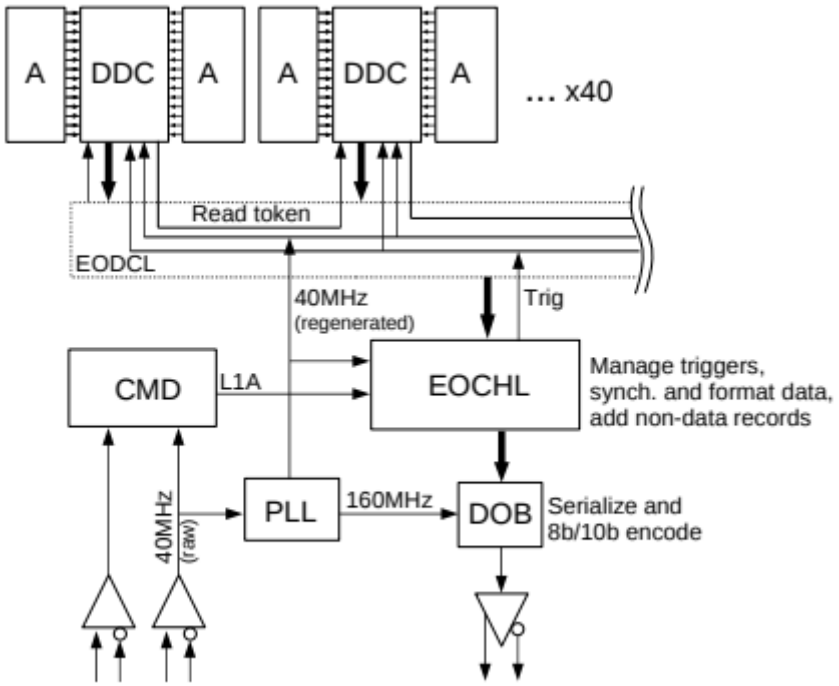

Abbildung 3.0.4: Vereinfachte schematische Darstellung der Chip-Architektur bei der Datennahme [7]

Logic (EODCL) zu allen DDC verbreitet und die Antwort der DDC wird von der EODCL an die EOCHL weitergeleitet, dort verarbeitet und via DOB aus dem Chip versendet.

4. Das Testsystem

Um ein Quad-Modul basierend auf FE-I4B-Chips auslesen zu können, sind externe Hardware- und Software-Komponenten notwendig. Zum Betrieb benötigte Hardware-Komponenten sind: Ein Netzteil zur Erzeugung der Niederspannung, benötigt für den Betrieb der 4 FE-I4B-Chips. Ein weiteres Netzteil zur Erzeugung der Depletionsspannung, benötigt zum Betrieb des Siliziumsensors. Dieses Netzteil muss computeransteuerbar sein, um automatisch das Verhalten des Sensors im Depletionsbereich aufzeichnen zu können. Das Verbinden des Quad-Moduls mit den Netzteilen sowie der Ausleseelektronik erfolgt über eine Feinstleiteradapterkarte (Quad-Adapter)(Abb.4.0.1 (a)). Der Quad-Adapter wurde vom SiLab der Universität Bonn entwickelt und hergestellt und benutzt einen Feinstleitersteckverbinder auf der zum Modul führenden Seite, um das Modul mit Strom und Datenverbindungen zu versorgen. Auf der vom Modul wegführenden Seite werden folgende Komponenten benutzt: Ein BNC-Steckverbinder, um den Sensor mit Spannung zu versorgen, ein 4 Pin-Steckverbinder, um die FE-I4B-Chips mit Strom zu versorgen und

vier RJ45-Ethernetbuchsen für die Datenübertragung zum Modul. Der Steckverbinder des Quad-Adapters wird über verschiedene flexible Leiterplatinen (Abb.4.0.1 (b+c)) mit dem Steckverbinder des Quad-Moduls verbunden. Die Datenauslese und Erzeugung von Befehlen für das Quad-Modul erfolgt durch die Multi-Module-Control-Karte (MMC3). Die MMC3-Karte wurde ebenfalls von der Universität Bonn entwickelt und wird mit der USBPix3 PC-Software ausgelesen/bedient. Um das Quad-Modul auf möglichst konstanter Temperatur zu halten, wird ein Kühlsystem benutzt, das im folgenden Abschnitt beschrieben wird.

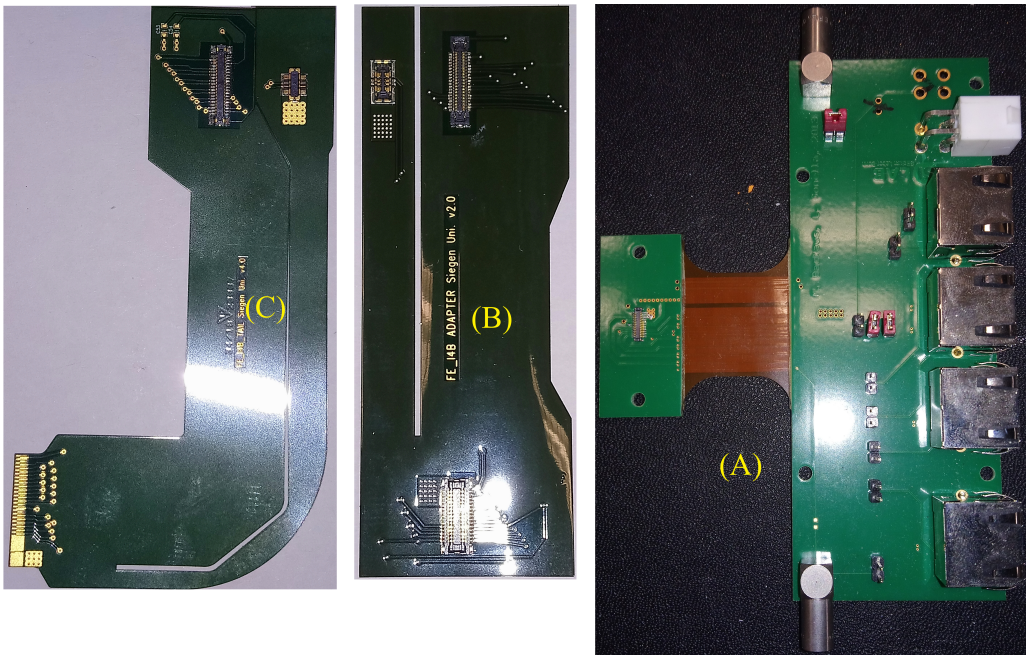

Abbildung 4.0.1: Der Quad-Adapter(A) und weitere flexible Adapter(B+C) zum Modul-Verbinder

4.1. Das Kühlsystem

Das Kühlsystem (Abb. 4.1.1) besteht aus einer in einem 3D-Drucker gedruckten Kammer, die mittels Peltierelementen gekühlt wird. Die Peltierelemente sind mit Isolierschaum umgeben, um den Wärmetransport zwischen den warmen und den kalten Seiten zu minimieren. Die warmen Seiten der Peltierelemente sind thermisch leitend mit einem Luftkühlsystem aus Aluminium-Finnen und einem Lüfter verbunden. Die kalten Seiten sind über thermisch leitendes Material mit einer Aluminium-Platte verbunden. Die Aluminium-Platte besitzt eine Erhebung mit thermisch leitender elektrisch isolierender Folie sowie Löcher, um ein Quad-Modul zu halten. Das Kühlsystem wird mit einem selbstregulierendem Steuermodul angesteuert und mit Strom versorgt. Das Steuermodul kann Temperaturen zwischen 20 °C und -20 °C bereitstellen. Für die Tests in dieser Arbeit ist das

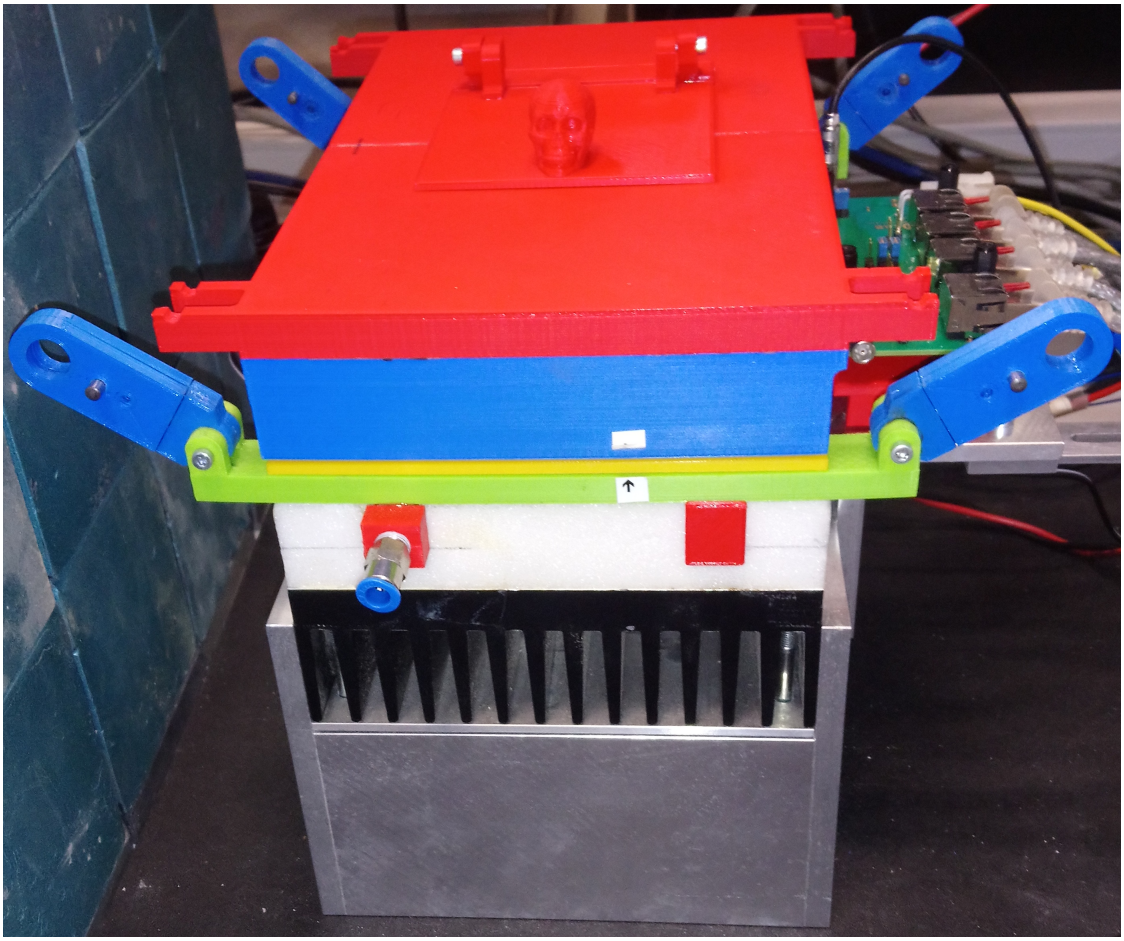

Abbildung 4.1.1: Das projektspezifische Kühlsystem

Steuermodul auf einen Sollwert von 10 °C eingestellt, was einer Modulbetriebstemperatur von ca. 20 °C entspricht. Die Kammer, in der sich das Quad-Modul befindet, wird kontinuierlich mit Trockenluft geflutet. Dies verhindert Kondensation von Wassertropfen auf den Oberflächen. Diese könnten zur Korrosion der Komponenten des Moduls führen.

4.2. Die USBPix3-Software

Die Auslesesoftware für die FE-I4 Chips (USBPIX3), wurde von der Arbeitsgruppe der Universität Bonn entwickelt. Sie benutzt eine externe Mikrokontroller-Karte, in diesem Fall eine MMC3 (Abb. 4.2.1), um einzelne FE-I4-Chips oder auf FE-I4-Chips basierende Hybrid-Module anzusteuern und auszulesen. USBPix3 ist in Python 2.7 geschrieben und alle Tests für FE-I4-Chips sind Python-Skripte, die entweder direkt in einem Terminal oder über eine Entwicklungsumgebung für Python, z.B. Eclipse, ausgeführt werden können. Mir der USBPix3-Software wird auch das Netzteil, das den Sensor mit Spannung

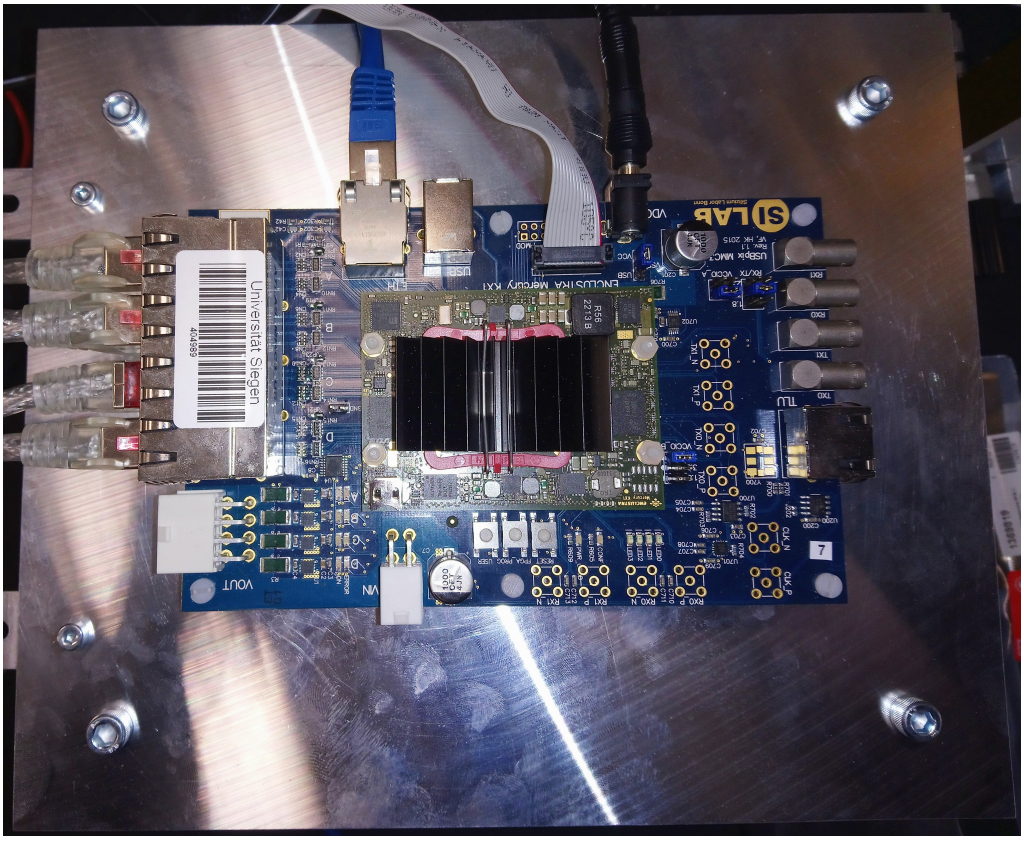

Abbildung 4.2.1: Foto der MMC3-Auslese-Karte

versorgt, gesteuert. Dies erfolgt über ein USB-zu-Seriellem Interface und ist notwendig, um die Sperrrichtungskennlinie des Sensors automatisch aufzunehmen. In den folgenden Abschnitten werden die für diese Arbeit notwendigen Tests beschrieben und die Einstellungen der Test erklärt.

4.2.1. Die Testreihenvorbereitung

Um USBPix3 Zugriff auf einen Chip zu geben, muss man die Konfigurationsdatei mit den Adressdaten des Chips und der Testreihe füllen. Wichtige anzugebende Parameter sind dabei der Pfad des Ordners, unter welchem die Testdaten gespeichert werden, sowie Angaben zu Anzahl, Typ, Chip-Adresse, physikalischem Kanal an der MMC3 Karte und Aktivierungsstatus des Chips. Eine genauere Beschreibung des notwendigen Programmcodes ist im Anhang unter A.1 zu finden. Um die Funktionsfähigkeit der erstellten Konfiguration zu testen, wird nach Anschalten der Testsystem-Hardware ein Initiationsskript ausgeführt. Das Skript führt unter anderem (Details im nächsten Abschnitt) einen Kommunikationstest mit den in der Konfigurationsdatei angegeben Chips durch. Wenn dieser erfolgreich ist, kann eine echte Testreihe gestartet werden.

4.3. Die Testprozedur

Um die mechanische Stabilität und Funktionsfähigkeit des Moduls nach jedem Neuverbinden des sich auf dem Modul befindenden Steckverbinders zu überprüfen, müssen eine Reihe von Tests mit dem Quad-Modul durchgeführt werden. Eine Testreihe beginnt mit dem Verbinden oder Wiederverbinden des Modulsteckverbinders und endet mit der Messung der Sperrstrom-Sperrspannungs-Kurve des Sensors des Moduls. Die folgenden Abschnitte beschreiben den Ablauf einer Testreihe und die zu erwartenden Ergebnisse anhand eines unbeschädigten Moduls. Technische Details zu den folgenden Tests sind im Anhang unter A.2 ,A.3, A.4, A.5, A.6 und A.7 zu finden.

4.3.1. Initialisierung

Das Initialisierungsskript wird zum Start einer Testreihe ausgeführt. Dadurch werden alle notwendigen Ordner, Standardkonfigurationen und Masken der Chips erstellt. Zudem werden Standardkonfigurationen und Masken in die Chips geladen. Des weiteren überprüft das Skript die Verbindung zu den in der Konfiguration angegebenen Chips. Für die Testreihen in dieser Arbeit wird nach dem Ausführen des Initialisierungsskriptes die Standard-Enable-Maske der Chips durch die letzte Enable-Maske der vorherigen Messreihe ersetzt, um zu gewährleisten, dass die Anzahl an maskierten Pixeln nur zunehmen kann.

4.3.2. Test der digitalen Chipelektronik

Um die Funktionsweise des Digitalteils des Chips zu überprüfen wird ein Digitalscan-Skript (scan_digital.py) ausgeführt. Dieses Skript versucht in jedem Pixel des Chips ein Signal auf digitaler Ebene zu erzeugen, wonach der gesamte Chip ausgelesen wird.

Dies wird 100 mal wiederholt und die erhaltenen Daten werden, sowohl in Computer verarbeitbarer Form (HDF5-Datei) als auch in lesbarer Form (PDF-Datei) gespeichert. Für einen perfekten Chip sähe das Ergebnis wie folgt aus (Abb.4.3.1): Das Ergebnis ist

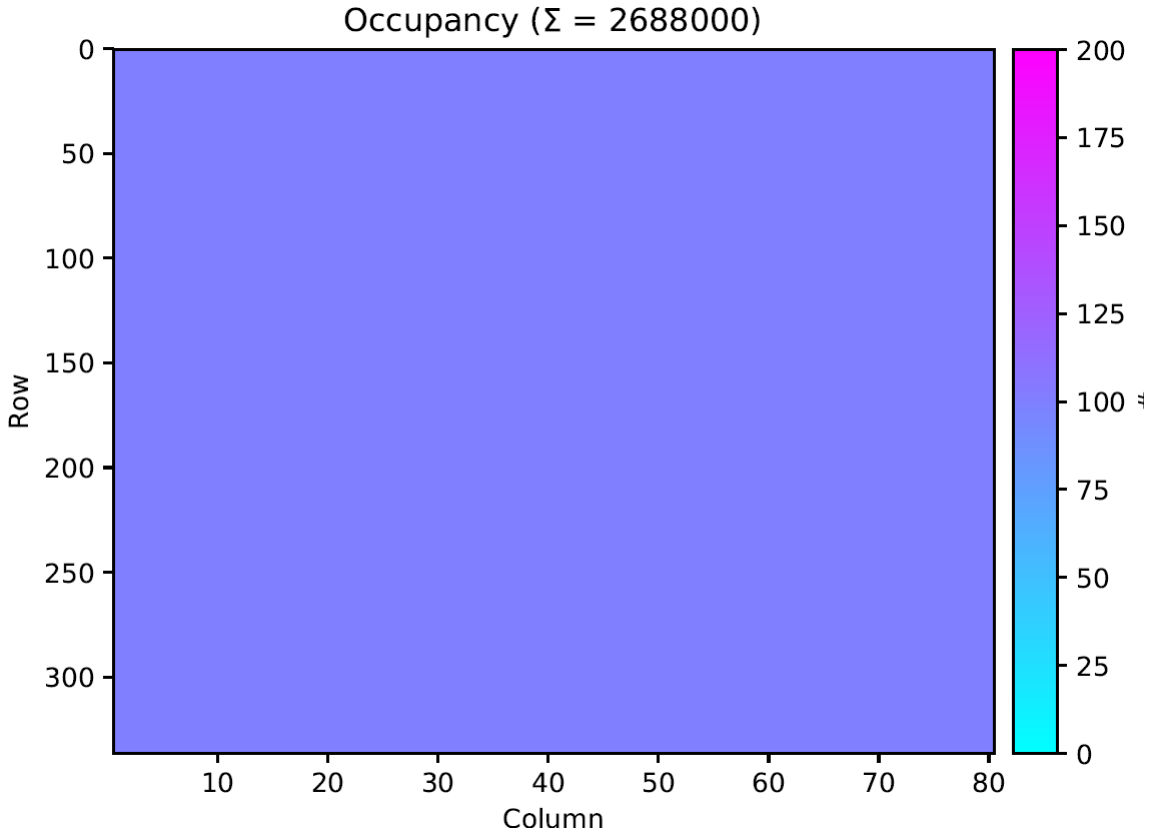

Abbildung 4.3.1: Die farbkodierte Treffermatrix eines perfekten digitalen Scans

eine 80 x 336 Matrix, in der jedes Pixel 100 Mal getroffen ist. Dies bedeutet, dass alle Pixel des Chip, zumindest auf digitaler Ebene funktionieren. Pixel, die weniger als 100 Treffer zeigen, werden in der späteren Auswertung als digital tot bezeichnet.

4.3.3. Test der analogen Chipelektronik

Zum Test des analogen Teils der Chipelektronik wird das Analogscan-Skript (scan_analog.py) ausgeführt. Dieses Skript benutzt chip-interne Elektronik, um Treffer eines Sensors zu simulieren. Dazu werden eine bestimmte Menge an Ladung, Größenordnung Kiloelektronen, in die Eingangszellen der Pixel eingekoppelt und danach ein Auslesesignal gesendet. Dieses Signal bringt den Chip dazu, alle registrierten Treffer an die Auslesesoftware weiterzugeben. Die benutzte Anzahl an Elektronen ist dabei sehr viel höher als die durch die im Chip vorhandenen Verstärker gegebene durchschnittliche Auslöseschwelle. Dies ergibt in der Datenauswertung nur die Möglichkeiten, entweder das Pixel löst aus

oder es löst nicht aus. Wie schon beim Test der digitalen Elektronik wird jedes Pixel 100 mal getestet und das Ergebnis in einer farbkodierten Matrix präsentiert (Abb. 4.3.2).

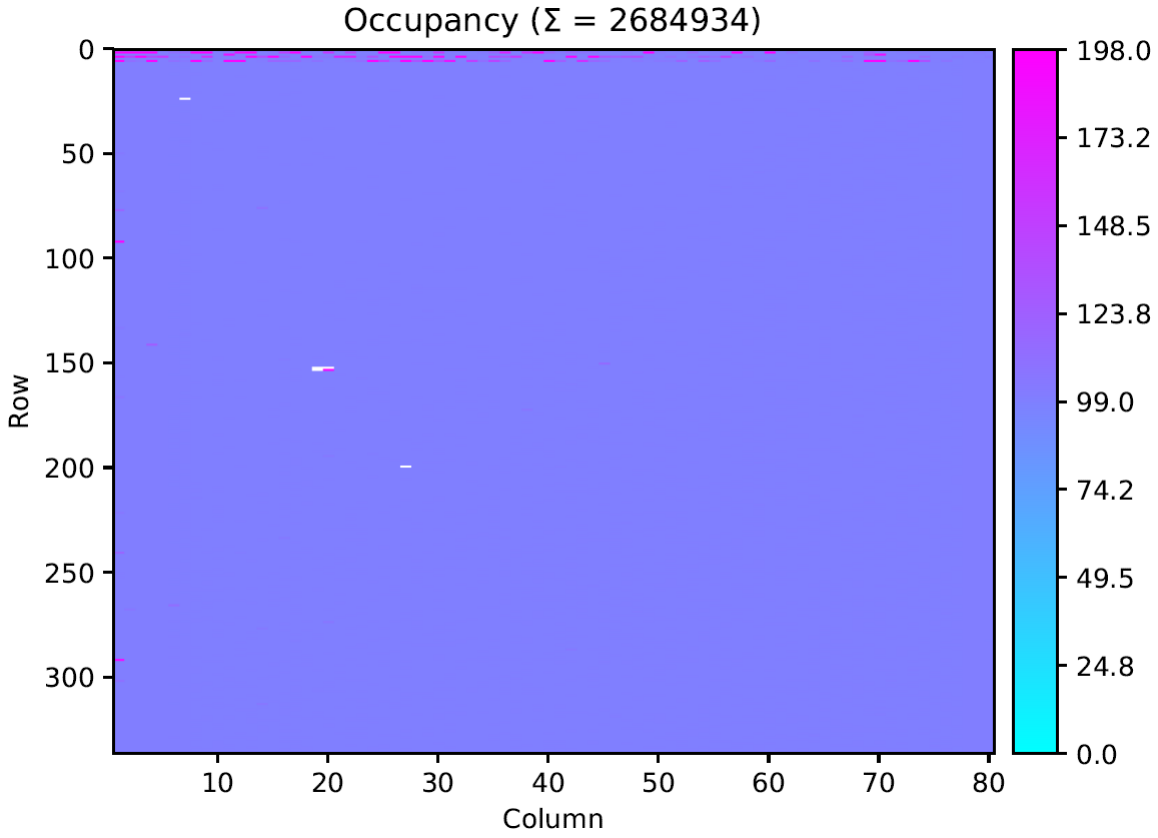

Abbildung 4.3.2: Die farbcodierte Treffermatrix eines fast perfekten analogen Scans

Für einen perfekt funktionierenden Chip ohne Sensor wäre ein Ergebnis wie beim digitalen Scan zu erwarten, d.h. eine volle Matrix mit 100 Treffern. Da die Kombination von Sensor und Auslesechip an den Kanten des Chips andere Pixelgrößen zum Überbrücken des Zwischenraums besitzt, erzeugt dies andere Kapazitäten in den Randpixeln, was zu Streifen im Ergebnis führt. Des weiteren ist der analoge Bereich anfälliger für Produktionsfehler, was zu defekten/ schlechter ansprechenden Einzelpixeln führt. Pixel, die in diesem Scan weniger als 95 Treffer haben, werden in der Auswertung als analog tot bezeichnet.

4.3.4. Angleichen der Pixelmatrixantwort

Durch produktionsbedingte Ungleichheiten und Dotierungsdichteschwankungen im Silizium der Chips folgt die Schaltschwelle der Pixel einer breiten Gaußverteilung. Idealerweise sollte die Gaußverteilung so schmal wie möglich und um die ausgewählte Schaltschwelle verteilt sein. Dafür wird das Pixel-Schwellentuning-Skript (tune_fei4.py) ausgeführt.

Das Skript passt die Schaltschwelle der Pixel an einen gegebenen Zielwert in Elektronen an und macht die Antwort der Pixelmatrix am Schwellenwert möglichst gleichmäßig.

4.3.5. Die automatisierten Maskierungs-Skripte

Um fehlerhafte oder rauschende Pixel in der Pixelmatrix zu finden und zu maskieren werden drei Skripte ausgeführt. Zuerst wird das Feststeckende-Pixel-Skript (tune_stuck_ pixel.py), dann das Rauschunterdrückungs-Skript (tune_noise_occupency.py) und zuletzt das Heiße-Pixel-Skript (tune_hot_pixel.py) ausgeführt. Das Feststeckende-Pixel-Skript findet Pixel, die in ihrem Trefferzustand feststecken, d.h. die Pixel registrieren entweder immer einen Treffer oder nie einen Treffer unabhängig von dem tatsächlichen Zustand der analogen Pixelregion. Das Rauschunterdrückungs-Skript setzt den Chip für die Dauer des Scans in einen selbstauslösenden Auslese-Modus (Abschnitt 4.3.7). Da sich bei diesem Scan keine radioaktive Quelle über dem Modul befindet, sind alle aufgezeichneten Treffer entweder kosmischer Strahlung oder elektronischem Rauschen zuzuordnen. Um elektronisch rauschende Pixel zu maskieren wird eine obere Schranke von standardmäßig 100 Treffern angesetzt, da rauschende Pixel meist weit häufiger (über 1 000 Treffer) in der Zeit des Scans aktiviert werden. Das letzte Skript findet heiße Pixel in der Pixelmatrix. Ein Pixel wird als heiß bezeichnet, wenn es nach einem registrierten Treffer nicht zurückgesetzt wird, d.h das Pixel bleibt nach einem Treffer für eine Weile im getroffenen Zustand hängen und wird weiter als getroffen ausgelesen. Alle Pixel, die von diesen drei Skripten gefunden werden, werden in der Enable-Maske deaktiviert, wodurch sie von bestimmten zukünftigen Scans ignoriert werden.

4.3.6. Überprüfen der Pixelmatrixantwort

Um sicher zu gehen, dass das Anpassen der Pixelmatrixantwort erfolgreich war, und um dessen Qualität zu überprüfen, wird ein Schwellenscan-Skript (scan_threshold.py) ausgeführt. Es führt analoge Scans mit bestimmten Elektronenanzahlen durch, um das Verhalten der Pixelmatrix in Vergleich mit der eingesetzten Elektronenanzahl aufzuzeichnen. Aus den so gewonnenen Daten werden für jedes Pixel die sogenannten S-Kurven (Kumulative Gaußfunktionen) angepasst.
Die Wendepunkte der S-Kurven entsprechen den Schaltwerten der Pixel. Die Verteilung der Schaltwerte sollte nach erfolgreichem Anpassen einer schmalen Gaußverteilung um den ausgewählten Elektronenwert entsprechen (Abb. 4.3.4).

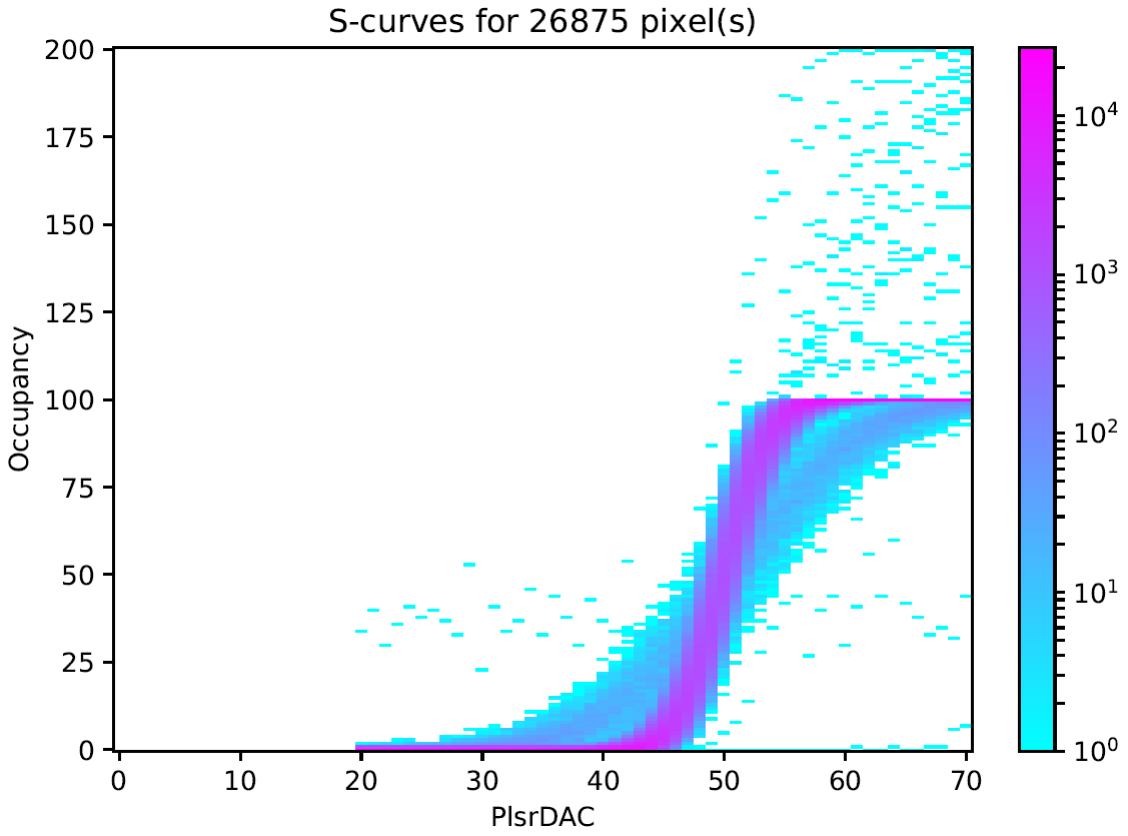

Abbildung 4.3.3: Das S-Kurven-Diagramm eines angepassten Chips

4.3.7. Test des Sensors

Zur Untersuchung der Funktionsfähigkeit des Sensors und dessen Verbindungen mit den Chips wird ein selbstauslösender Scan benutzt. Dieser Scan setzt die Chips in einen speziellen Betriebsmodus. Bei jedem registrierten Treffer eines Pixels wird chip-intern ein Auslesesignal gesendet, sodass die Daten gespeichert werden. Für die Tests in dieser Arbeit wird eine Strontium-90-Quelle benutzt, welche energiereiche Elektronen aus radioaktiven Betazerfällen liefert, um die Treffer im Sensor zu erzeugen. Die Daten der Treffer werden über 5 Minuten gesammelt, wobei sich die Quelle dabei über der Mitte einer der vier Chips befindet. Um das komplette Modul zu testen, werden vier Tests kombiniert und ein fünfter Test wird durchgeführt, um speziell den Steckverbinder zu untersuchen, wobei sich die Quelle direkt über dem Verbinder befindet und alle Chips gleichzeitig Daten sammeln.
Abb.4.3.5 zeigt das kombinierte Ergebnis eines Scans mit der Quelle in der Mitte über den Chips. Nach 5 Minuten Datennahme registrierten die Pixel direkt unter der Quelle ca. 200 Treffer und am Rand der Chips oder unter Komponenten der Leiterplatine ca. 10 Treffer. Pixel, die keine Treffer besitzen, sind weiß dargestellt und sind entweder nicht

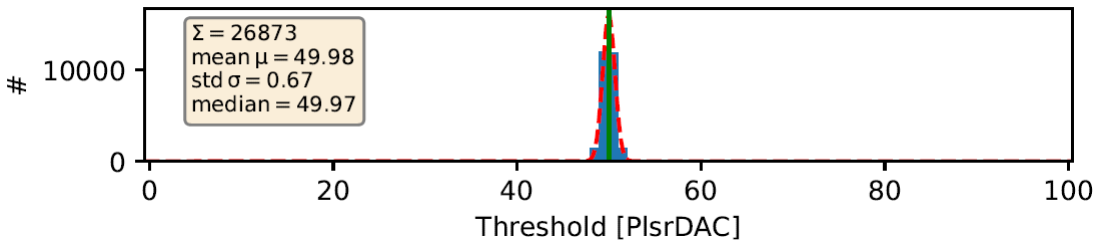

Abbildung 4.3.4: Das Histogramm der S-Kurven-Wendepunkte

funktionsfähig oder maskiert. Aus diesem Test werden in der Auswertung die Pixel ermittelt, welche die vorherigen Test bestanden haben aber dennoch keine Reaktion unter Bestrahlung zeigen. Diese zusätzlichen toten Pixel sind interessant, da sie auf eine mögliche Beschädigung des Sensors oder der Verbindung zum Sensor hinweisen. Der Test mit der Platzierung der Quelle über dem Verbinder wird nur benötigt, um zu zeigen, dass durch das Wiederverbinden des Steckverbinders keine großflächigen Beschädigungen in der Nähe des Steckverbinders entstehen.

4.3.8. Messen der Sperrstrom-Sperrspannungs-Kennlinie

Zuletzt wird die Sperrstrom-Sperrspannungs-Kennlinie (IV-Kurve) des Sensors bestimmt. Dazu fährt das Testsystem die externe Spannungsquelle des Sensors von 2 Volt in 2 Volt Schritten bis 200 Volt Sperrspannung durch und zeichnet den gemessenen Sperrstrom auf. Idealerweise sollte die Kurve der einer idealen Diode folgen (vor Zusammenbau des Moduls), aber experimentell sieht man, dass durch das Aufkleben der Leiterplatine auf den Sensor ein Kontaktwiderstand entsteht, was einem linearen Verlauf der Kurve bei hohen Spannungen entspricht (anstelle eines Plateaus). Nach dem Beenden der Messung wird die Spannung automatisch auf den eingestellten Betriebswert von 10 Volt in Stufen zurückgefahren, um Schäden zu vermeiden.

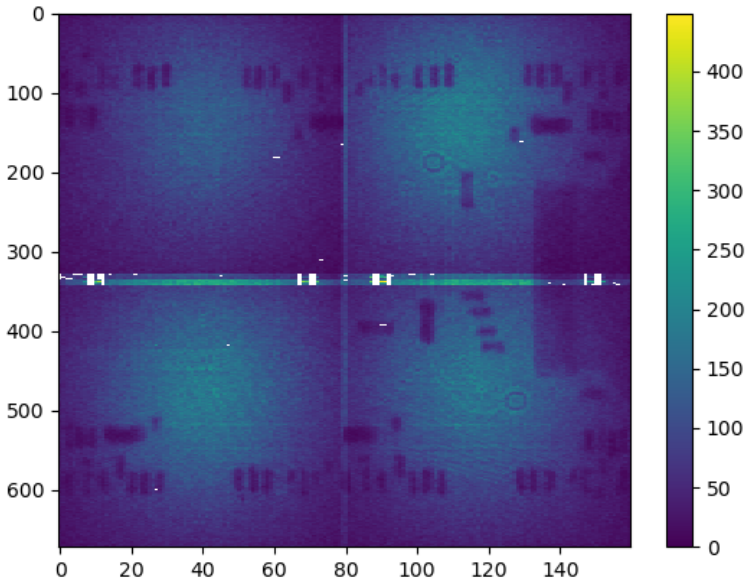

Abbildung 4.3.5: Die farbkodierte Treffermatrix des kombinierten selbstauslösenden Scans mit radioaktiver Quelle

5. Die Datenanalyse

Um die Frage zu beantworten, ob wiederholtes Öffnen und Schließen des Modulsteckverbinders zu Schäden am Quad-Modul (hauptsächlich Brechen der Bump-Bonds zwischen Chips und Sensor) führt, werden die Daten der im vorherigen Kapitel beschriebenen Messreihen mittels eines automatisierten Skripts ausgewertet. Dabei werden hauptsächlich defekte Pixel auf dem Quad-Modul gesucht, welche die elektronischen Tests auf Chipebene bestanden haben, aber bei der Datennahme mittels radioaktiver Quelle keine Daten liefern. Fänden sich eine große Anzahl dieser Pixel, insbesondere in der Umgebung um den Steckverbinder, dann wäre die mechanische Belastung ein wahrscheinlicher Auslöser von Brüchen der Bump-Bonds.

5.1. Der Dateneingang

Die Daten, die von den in Kapitel 4 beschriebenen Scans produziert werden, sind in Dateien im HDF5-Format gespeichert. Das HDF5-Format ist ein wissenschaftliches Dateiformat, das zur Speicherung von komplexen und/oder großen Datensätzen entwickelt wurde. Die zu analysierenden Daten in dieser Arbeit sind sowohl komplex als auch groß, da in einer der von den Scans produzierten Dateien alle Konfigurationen und der vollständige Datenstrom des Chips in dezimal komprimierter Form gespeichert sind. Au-

ßerdem enthält die Datei mit zusätzlichem Namensanhang (_interpreted) die vom Scan ausgewerteten Daten des USBPIX3-Scans. Diese ausgewerteten Daten sind in der HDF5-Datei direkt als Pythonobjekte vom Typ Matrix (Numpy nd-Array) hinterlegt, sodass die HDF5-Datei direkt mit Python geöffnet werden kann. Daten, die vom Analyseskript erstellt werden, werden in Textdateien mit Endung (.dat) gespeichert. Der Inhalt dieser Textdateien folgt einer vorgegebenen Form, um späteres Einlesen zu erleichtern, sowie die Auswertung der Daten durch ROOT zu ermöglichen. Da diese Textdateien üblicherweise Matrizen repräsentieren, sind sie wie folgt aufgebaut:

```
X/L:Y/L:Z/L  // ROOT-Headers
-1 -1 336    // Länge der Matrix
-1 -1 80     // Breite der Matrix
24 6 -1      // Erster Daten-Eintrag
.....
```

Die erste Zeile ist eine Kopfzeile, die notwendig ist, um die Textdatei in eine ROOT-Tree-Datei umzuwandeln. Die zweite Zeile gibt in der Z-Spalte die Größe der Matrix in X an und die dritte Zeile gibt ihre Größe in Y an. Einträge mit negativem Wert geben eine leere Zelle in der Tabelle an. Die Zeilen 4 und folgende sind Positionsdaten in X und Y von Pixeln/Zellen der Matrix, die für den Auswertungsschritt relevant sind. Hier wird die Spalte Z als negativ markiert, wenn die Trefferzahl des repräsentierten Pixels irrelevant für den Auswertungsschritt ist. Bei den Dateien, die defekte Pixel speichern, sind alle Einträge in Z negativ. Als Gegenbeispiel sind die Dateien, die die Daten der Quellenscans repräsentieren, anzugeben. Hier steht in der Z-Spalte die Anzahl der Treffer des Pixels.

5.2. Die Analyseschritte

Das Python-Skript, das zur Analyse der Daten der USBPIX3-Scans benutzt wird, ist in Python 3 geschrieben. Es kann aber auch in Python 2.7 ausgeführt werden, wenn entsprechende Backports der notwendigen Pythonmodule installiert sind. Zu Beginn der Analyse werden die notwendigen Pythonmodule geladen. Genaueres zu den benötigten Pythonmodulen ist im Anhang unter A.8 zu finden. Hiernach werden die Hauptspeichervariablen initialisiert, welche den Anzahlen der gesuchten defekten Pixeln in jeder USBPIX3-Testreihe entsprechen. Die Konfigurationsdatei wird geladen. Sie teilt dem Skript mit, in welchem Ordner sich die zu analysierenden Daten befinden, welche Daten zu analysieren sind und in welchen Ordner die Zwischenergebnisse und Ergebnisse gespeichert werden sollen. Eine genauere Funktionsbeschreibung der Konfigurationsdatei kann im Anhang unter A.9 nachgelesen werden. Nach dem Laden der Konfigurationsdatei beginnt die eigentliche Datenanalyseschleife. Zu Beginn der Schleife wird ein Unterordner im Ausgangsordner mit Namen der zu analysierenden USBPIX3-Testreihe erstellt. Danach werden die Daten des Tests der digitalen Chipelektronik, gefolgt von den Daten des Tests der analogen Chipelektronik und den Daten aus dem Test des Sensors analysiert. Bei der Analyse der Daten wird immer parallel zu den auszuwertenden Daten die Enable-Maske des zu analysierenden Scans geladen, um auszuschließen, dass deaktivierte Pixel

gefunden werden. Bei der Analyse der digitalen Chipelektronik wird die Treffermatrix auf Pixel untersucht, welche von der zu erwarteten Anzahl an Treffern (100) nach unten abweichen. Alle so gefundenen Pixel werden in einer Zwischenspeicherdatei mit dem Namen module_X_DigitalLow abgespeichert, wobei X dem analysierten Chip entspricht. Im nächsten Schritt, der Analyse der analogen Chipelektronik, wird die Treffermatrix des Scan nach Pixeln mit weniger als 96 Treffern durchsucht, da dies einer Detektionseffizienz von 95% oder weniger entspricht. Diese Effizienz ist eine typische Grenze, die ein Teilchendetektor erreichen sollte, um effizient zu arbeiten und ist aus diesem Grund gewählt worden. Die Kombination aus Auslesechip und Sensor führt dazu, das der Chip nicht mehr alle simulierten Treffer registriert, was zu einer Streuung der Trefferanzahlen nahe der simulierten Anzahl (100) führt. Daher musste eine untere Grenze für die Trefferanzahl eines funktionierenden Pixels festgelegt werden. Die Streuung der Trefferzahlen ist schmal, d.h. ca. 30 % der Pixel haben 100 Treffer und die restlichen 70% (ausgeschlossen jener, die unter die 95% Grenze fallen) haben 99 Treffer. Die Pixel, die unter der 95% Detektionseffizienzgrenze fallen, haben nie mehr als 10 Treffer im Test der analogen Chipelektronik.
Es folgt die Analyse der Daten aus dem Test des Sensors. Hierbei werden alle Pixel aus der Treffermatrix gesucht, die exakt 0 Treffer haben. Es wird nach exakt 0 Treffern gesucht, um eine fälschliche Erkennung von funktionierenden Pixeln in den Randregionen und unter den Platinenkomponenten zu vermeiden. Alle so identifizierten Pixel müssen also auf irgendeine Art beschädigt sein. Des Weiteren wird eine Zwischenspeicherdatei mit Namen module_X_Mask erstellt, die alle maskierten Pixel, (Pixel mit Eintrag in Enable-Maske gleich Null) abspeichert. Als nächstes werden die Daten der vier Chips kombiniert und visuell dargestellt. Hierbei werden zwei Bilddateien im PNG-Format erstellt. Das erste Bild enthält die Daten aus den Bestrahlungen des Moduls mit der radioaktiven Quelle in der Position über der Mitte der Chips. Das zweite Bild enthält die Daten der Bestrahlung des Moduls mit der radioaktiven Quelle positioniert über dem Modulsteckverbinder. Diese Bilder dienen der visuellen Inspektion der Funktionsfähigkeit des Moduls, da zu erwarten ist, wenn die Bump-Bonds durch den Druck des Schließens des Steckverbinders brechen, dies großflächig in der Nähe des Steckverbinders geschieht. Der nächste Analyseschritt dient zur Identifikation der Fehlerstelle bei den gefundenen Pixeln. Es werden von den gefundenen Pixeln aus dem Test des Sensors die gefundenen Pixel aus den Tests der analogen und digitalen Chipelektronik abgezogen. So erhält man die Pixel, welche entweder im Sensor defekt sind oder einen gebrochenen Bump-Bond besitzen. Es werden auch die Pixel aus dem Test der digitalen Chipelektronik von den gefundenen Pixeln aus dem Test der analogen Chipelektronik abgezogen, da man so die defekten Pixel auf analoger Chipebene enthält. Auf digitaler Chipebene muss nichts getan werden, da die Maske des Moduls schon bei der Suche der Pixel angewendet wurde. Hiermit endet die Analyseschleife. Die Analyseschleife wird solange wiederholt, wie in der Konfigurationsdatei angegebene USBPIX3-Testreihen zur Verfügung stehen. Zum Schluss des Analyseskripts werden alle Sperrstromsperrspannungskurven der Testreihen in einem kombinierten Diagramm dargestellt und Balkendiagramme zu den Fehlerstellen des Moduls im Verlauf der Testreihen erstellt.

6. Ergebnisse

Da die vier Chips des Quad-Moduls einzeln ausgelesen werden, existieren vier separate Ergebnisse. Um diese visuell korrekt interpretieren zu können, werden die Daten der Chips entsprechend folgenden Schemas (Abb. 6.0.1) aneinander gefügt:

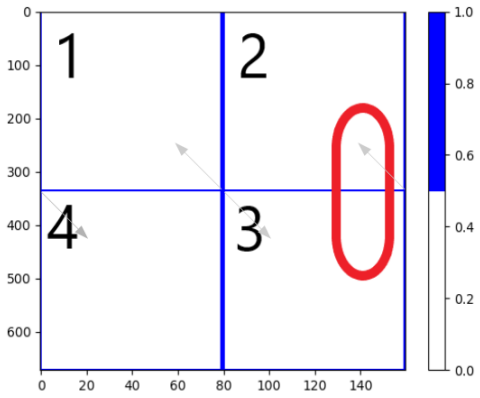

Abbildung 6.0.1: Schematische Erklärung der Modultreffermatrix

Zu sehen ist die zusammengesetzte Pixelmatrix des Moduls (160 x 672 Pixel). Die Randpixel jedes Chips sind mit blau markiert und die chip-eigene Pixelnummerierungsrichtung ist durch die grauen Pfeile angezeigt. Die Pixel, die auf den Achsen (bei den Werten 0) liegen, sind durch die Umrandung verdeckt. Die rote Region markiert die Position des Modulsteckverbinders, welche für die Auswertung von besonderem Interesse ist. Die Zahlen in schwarz sind die Chip-Identifikationsnummern.
Die USBPIX3-Testreihen haben das Benennungsschema G_Rx, wobei das G für den Ursprung des Bare-Moduls (Sensor und Bump-Bonded Chips ohne Flex) entspricht. Dieses Bare-Modul wurde der Universität Siegen von der Universität Glasgow zur Verfügung gestellt. Das R in G_Rx steht für die Wiederverbindung (eng. Reconnection) und das x gibt die Anzahl an Wiederverbindungen startend von 0 an. Die Daten zur Erstverbindung 0 des Moduls sowie der Wiederverbindungen 1-3 sind im Anhang unter A.10, A.11, A.12 und A.13 zu finden. Diese 4 Testreihen sind aus der Auswertung herausgenommen, da keine bekannte konstante Modulbetriebstemperatur gewährleistet war, da das Kühlsystemsteuermodul beschädigt und nicht vollständig funktionsfähig war.
In den folgenden Abschnitten sind jeweils zwei Bilder zur visuellen Inspektion des Moduls, sowie Listen der defekten Pixel pro Modulebene und Chip angegeben. Die Position des Pixels wird dabei als Tuple (X,Y) angegeben, wobei X im Bereich von 0-335 und Y im Bereich von 0-79 liegen kann. Auf digitaler Chipebene gilt ein Pixel als defekt, wenn weniger als 100 Treffer im Test der digitalen Chipelektronik registriert wurden. Auf analoger Chipebene gilt ein Pixel als defekt, wenn weniger als 96 Treffer im Test der analogen Chipelektronik registriert wurden und es auf digitaler Chipebene funktionsfähig ist.
Auf Ebene des Sensors gilt ein Pixel als defekt, wenn exakt 0 Treffer beim Test des

Sensors registriert wurden und das Pixel anderweitig funktionsfähig ist.

6.1. Wiederverbindung 4

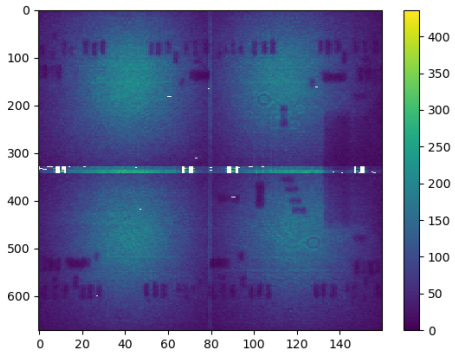

Abbildung 6.1.1: Kombiniertes Bild der einzelnen Chipquellentests für Wiederverbindung 4

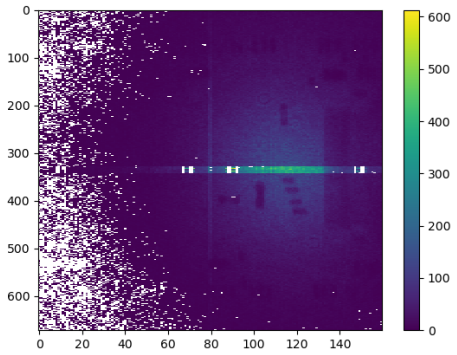

Abbildung 6.1.2: Kombiniertes Bild des Tests mit radioaktiver Quelle nahe des Steckverbinders für Wiederverbindung 4

Defekte Pixel auf digitaler Chipebene

Es wurden keine defekten Pixel gefunden.

Defekte Pixel auf analoger Chipebene

Es wurden 16 defekte Pixel gefunden.

Chip 1	Chip 2	Chip 3	Chip 4
(24,6)	(1,42)	(56,11)	(82,47)
(152,18)	(173,30)	(57,10)	(83,27)
(152,19)		(57,11)	(116,66)
(153,18)		(250,11)	(264,27)
(199,26)			(281,69)

Defekte Pixel auf Sensorebene

Es wurden 5 defekte Pixel in Chip 1 gefunden. 141 Pixel auf dem gesamten Modul maskiert.

Chip 1	(0,77)	(0,79)	(2,79)	(6,79)	(169,0)

6.2. Wiederverbindung 5

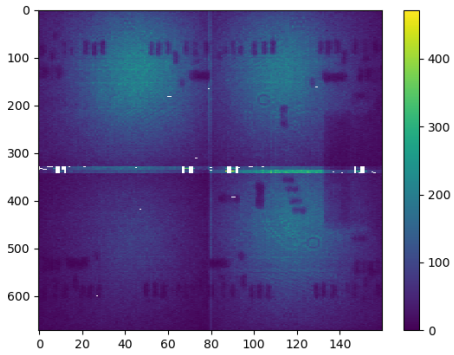

Abbildung 6.2.1: Kombiniertes Bild der einzelnen Chipquellentests für Wiederverbindung 5

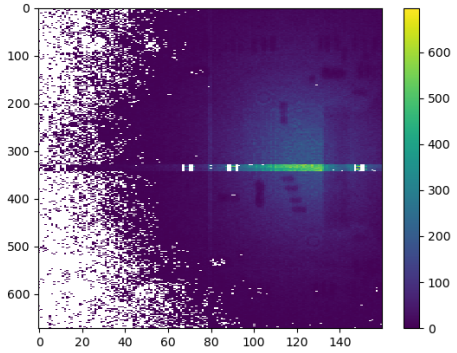

Abbildung 6.2.2: Kombiniertes Bild des Tests mit radioaktiver Quelle nahe des Steckverbinders für Wiederverbindung 5

Defekte Pixel auf digitaler Chipebene

Es wurden keine defekten Pixel gefunden.

Defekte Pixel auf analoger Chipebene

Es wurden 14 defekte Pixel gefunden.

Chip 1	Chip 2	Chip 3	Chip 4
(24,6)	(173,30)	(56,11)	(83,27)
(152,18)		(57,10)	(116,66)
(152,19)		(57,11)	(264,27)
(153,18)		(250,11)	(281,69)
(199,26)			

Defekte Pixel auf Sensorebene

Es wurden 6 defekte Pixel in Chip 1 gefunden. 141 Pixel auf dem gesamten Modul maskiert.

Chip 1	(0,77)	(0,79)	(2,79)	(3,79)	(6,79)	(169,0)

6.3. Wiederverbindung 6

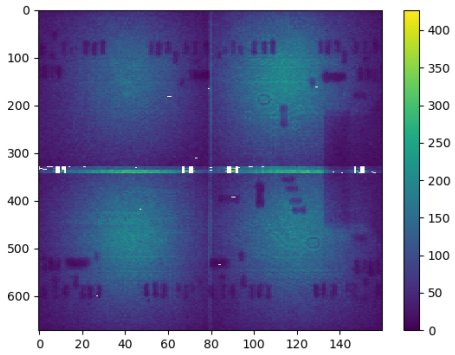

Abbildung 6.3.1: Kombiniertes Bild der einzelnen Chipquellentests für Wiederverbindung 6

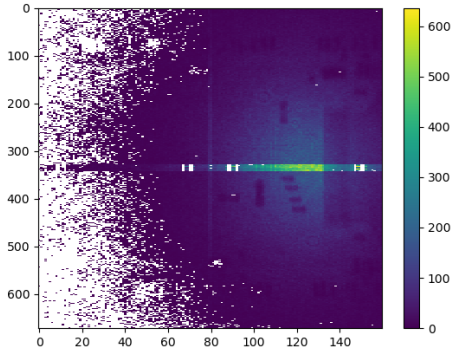

Abbildung 6.3.2: Kombiniertes Bild des Tests mit radioaktiver Quelle nahe des Steckverbinders für Wiederverbindung 6

Defekte Pixel auf digitaler Chipebene

Es wurden keine defekten Pixel gefunden.

Defekte Pixel auf analoger Chipebene

Es wurden 14 defekte Pixel gefunden.

Chip 1	Chip 2	Chip 3	Chip 4
(24,6)	(173,30)	(56,11)	(83,27)
(152,18)		(57,10)	(116,66)
(152,19)		(57,11)	(264,27)
(153,18)		(250,11)	(281,69)
(199,26)			

Defekte Pixel auf Sensorebene

Es wurden 5 defekte Pixel in Chip 1 gefunden. 141 Pixel auf dem gesamten Modul maskiert.

Chip 1	(0,77)	(0,79)	(2,79)	(6,79)	(169,0)

6.4. Wiederverbindung 7

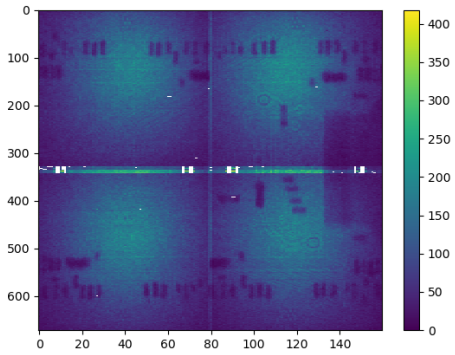

Abbildung 6.4.1: Kombiniertes Bild der einzelnen Chipquellentests für Wiederverbindung 7

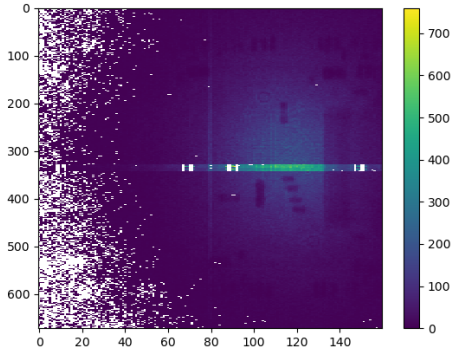

Abbildung 6.4.2: Kombiniertes Bild des Tests mit radioaktiver Quelle nahe des Steckverbinders für Wiederverbindung 7

Defekte Pixel auf digitaler Chipebene

Es wurden keine defekten Pixel gefunden.

Defekte Pixel auf analoger Chipebene

Es wurden 14 defekte Pixel gefunden.

Chip 1	Chip 2	Chip 3	Chip 4
(24,6)	(173,30)	(56,11)	(83,27)
(152,18)		(57,10)	(116,66)
(152,19)		(57,11)	(264,27)
(153,18)		(250,11)	(281,69)
(199,26)			

Defekte Pixel auf Sensorebene

Es wurden 5 defekte Pixel in Chip 1 gefunden. 141 Pixel auf dem gesamten Modul maskiert.

Chip 1	(0,77)	(0,79)	(2,79)	(6,79)	(169,0)

6.5. Wiederverbindung 8

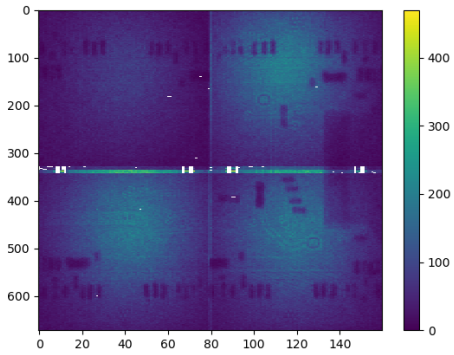

Abbildung 6.5.1: Kombiniertes Bild der einzelnen Chipquellentests für Wiederverbindung 8

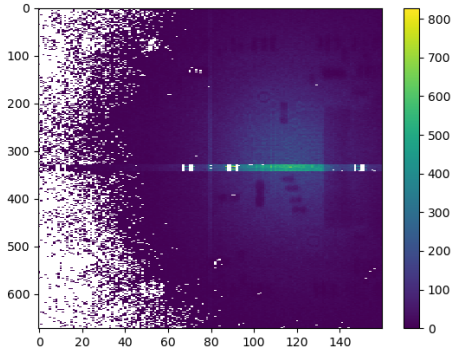

Abbildung 6.5.2: Kombiniertes Bild des Tests mit radioaktiver Quelle nahe des Steckverbinders für Wiederverbindung 8

Defekte Pixel auf digitaler Chipebene

Es wurden keine defekten Pixel gefunden.

Defekte Pixel auf analoger Chipebene

Es wurden 14 defekte Pixel gefunden.

Chip 1	Chip 2	Chip 3	Chip 4
(24,6)	(173,30)	(56,11)	(83,27)
(152,18)		(57,10)	(116,66)
(152,19)		(57,11)	(264,27)
(153,18)		(250,11)	(281,69)
(199,26)			

Defekte Pixel auf Sensorebene

Es wurden 6 defekte Pixel in Chip 1 gefunden. 141 Pixel auf dem gesamten Modul maskiert.

Chip 1	(0,77)	(0,79)	(2,79)	(3,79)	(6,79)	(169,0)

6.6. Wiederverbindung 9

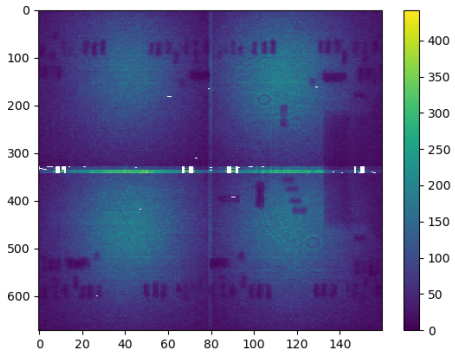

Abbildung 6.6.1: Kombiniertes Bild der einzelnen Chipquellentests für Wiederverbindung 9

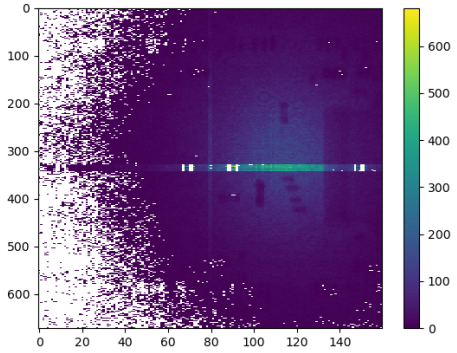

Abbildung 6.6.2: Kombiniertes Bild des Tests mit radioaktiver Quelle nahe des Steckverbinders für Wiederverbindung 9

Defekte Pixel auf digitaler Chipebene

Es wurden keine defekten Pixel gefunden.

Defekte Pixel auf analoger Chipebene

Es wurden 14 defekte Pixel gefunden.

Chip 1	Chip 2	Chip 3	Chip 4
(24,6)	(173,30)	(56,11)	(83,27)
(152,18)		(57,10)	(116,66)
(152,19)		(57,11)	(264,27)
(153,18)		(250,11)	(281,69)
(199,26)			

Defekte Pixel auf Sensorebene

Es wurden 5 defekte Pixel in Chip 1 gefunden. 141 Pixel auf dem gesamten Modul maskiert.

Chip 1	(0,77)	(0,79)	(2,79)	(6,79)	(169,0)

6.7. Wiederverbindung 10

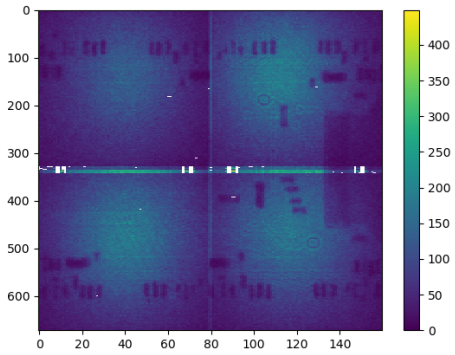

Abbildung 6.7.1: Kombiniertes Bild der einzelnen Chipquellentests für Wiederverbindung 10

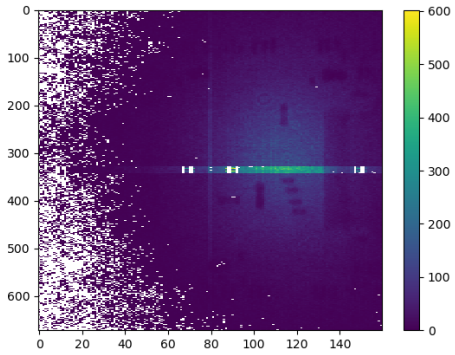

Abbildung 6.7.2: Kombiniertes Bild des Tests mit radioaktiver Quelle nahe des Steckverbinders für Wiederverbindung 10

Defekte Pixel auf digitaler Chipebene

Es wurden keine defekten Pixel gefunden.

Defekte Pixel auf analoger Chipebene

Es wurden 14 defekte Pixel gefunden.

Chip 1	Chip 2	Chip 3	Chip 4
(24,6)	(173,30)	(56,11)	(83,27)
(152,18)		(57,10)	(116,66)
(152,19)		(57,11)	(264,27)
(153,18)		(250,11)	(281,69)
(199,26)			

Defekte Pixel auf Sensorebene

Es wurden 6 defekte Pixel in Chip 1 gefunden. 141 Pixel auf dem gesamten Modul maskiert.

Chip 1	(0,77)	(0,79)	(2,79)	(3,79)	(6,79)	(169,0)

6.8. Interpretation

Digitale Chipebene

Aus der Analyse der Daten der Tests der digitalen Chipelektronik ergibt sich, dass alle Pixel des Quad-Moduls auf digitaler Ebene funktionieren und auch nach 10 Wiederverbindungen des Steckverbinders keine defekten Pixel auftreten.

Analoge Chipebene

Aus der Analyse der Daten der Tests der analogen Chipelektronik ergibt sich folgendes (Abb. 6.8.1):

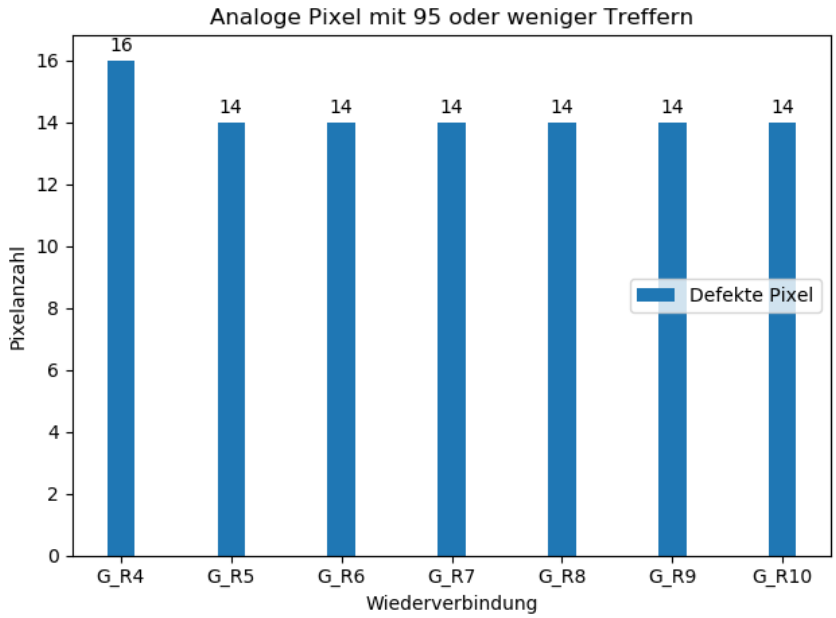

Abbildung 6.8.1: Balkendiagramm für die defekten Pixel des Quad-Moduls auf analoger Chipebene

Zu erkennen ist, das 14 Pixel auf der analogen Chip-Ebene des Modul keine Reaktion zeigen. Zwischen Schritt G_R4 und GR_5 wurden 2 Pixel, die in G_R4 als analog tot gekennzeichnet waren, beim automatisierten Maskieren als feststeckend (immer aus) erkannt und in der Enable-Maske deaktiviert. Aus diesem Grund sind sie in den folgenden Versuchsreihen nicht mehr auf analoger Chip-Ebene zu finden. Aus der Position der defekten Pixel auf analoger Chip-Ebene (Abb. 6.8.2) lassen sich keine Rückschlüsse ziehen. Die Defekte sind produktionsbedingt, da die benutzten Chips aus der Produktion des IBL stammen, aber zu niedrige Qualität hatten, um im IBL verbaut zu werden.

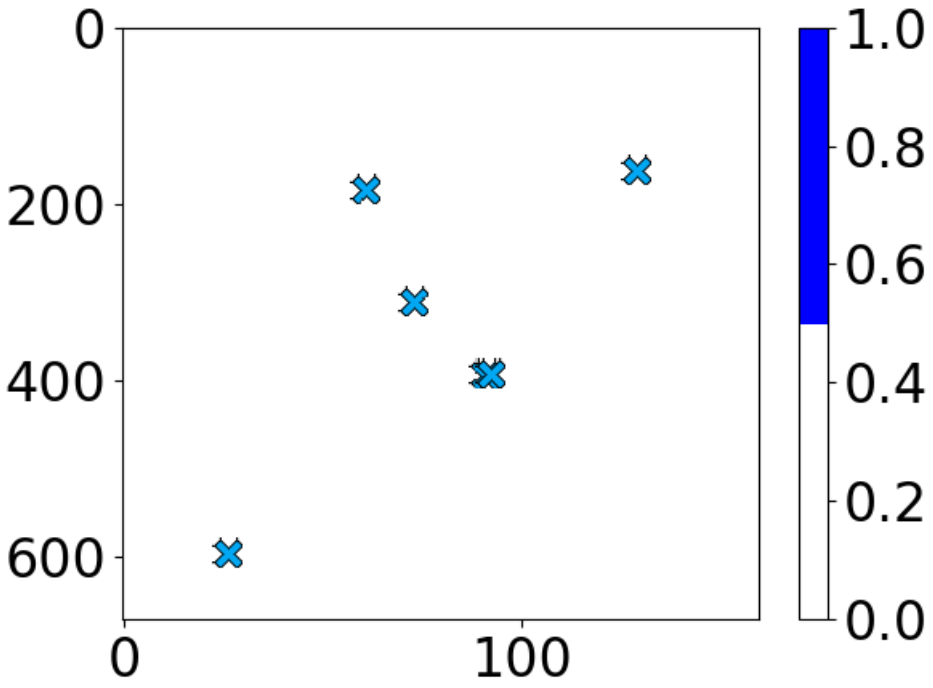

Abbildung 6.8.2: Modulpixelmatrix mit defekten Pixeln auf analogem Level aus Wiederverbindung 10

Sensorebene und Bump-Bonds

Bei der Analyse der Daten aus dem Test des Sensors (Abb. 6.8.3) wurden 5 beziehungsweise 6 Pixel gefunden, die nur auf Sensor- bzw. Bump-Bond-Ebene nicht funktionsfähig sind. Alle diese Pixel befinden sich in der Region des Sensors, die von Chip 1 bedient wird und sind Randpixel. Das Pixel (3,79), das zwischen den Versuchsreihen mal funktioniert und mal nicht, hat wenn es funktioniert, maximal 1 Treffer, wobei die Pixel, die sich am weitesten von der Strahlungsquelle entfernt befinden, im Durchschnitt 10 Treffer zeigen. Fünf dieser Pixel häufen sich in der unteren linken Ecke (aufs Modul gesehen) des Chip 1, wie in Abb. 6.8.4 zu sehen. Da die defekten Pixel isoliert, über die Testreihen konstant und weit vom Steckverbinder entfernt sind, gibt es keinen Zusammenhang zum Wiederverbinden des Steckverbinders .

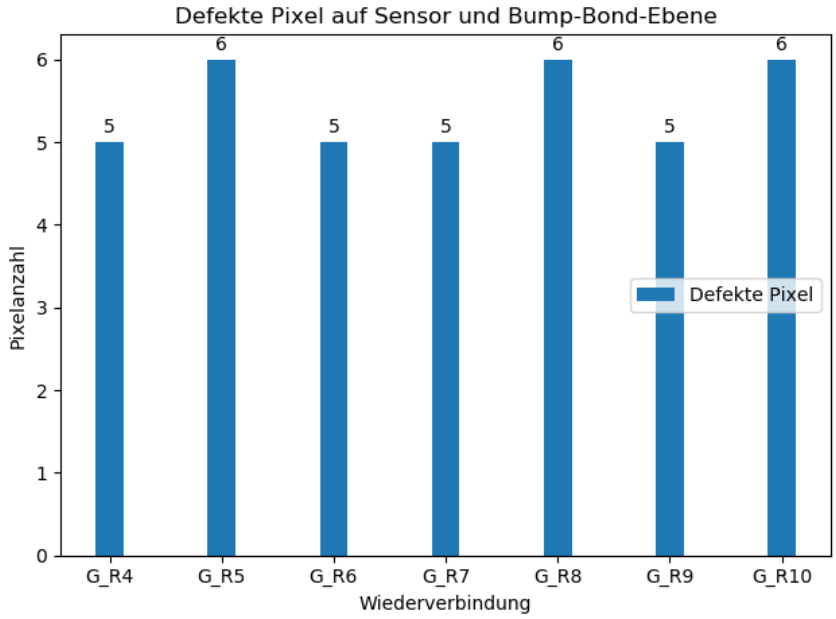

Abbildung 6.8.3: Balkendiagramm der Anzahl der auf Sensor-/Bump-Bond-Ebene defekten Pixel des Moduls

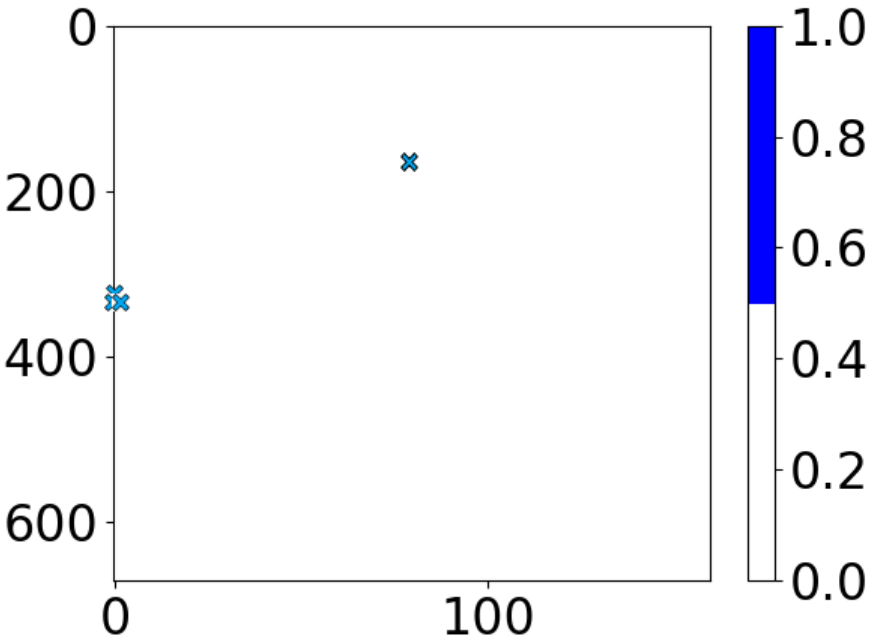

Abbildung 6.8.4: Position der defekten Pixel auf Sensor- bzw. Bump-Bond-Ebene im Modul für Wiederverbindung 10

IV-Kurven

In Abbildung 6.8.5 sind die Sperrstromsperrspannungskennlinien des Silizium-Sensors des Moduls zu sehen. Zu erkennen ist eine Verteilung der Kurven in einem Intervall von ca. 50 bis 80 Nanoampere-Breite und keiner erkennbaren Anordnung in der Versuchsreihenfolge. Die unterschiedlichen y-Achsenabschnitte der Kurven können durch leichte Unterschiede in der Luftfeuchtigkeit und durch Streulichteinfall auf das Modul erklärt werden. Die Form der Kurve ergibt sich aus der Addition der perfekten Sperrstromsperrspannungskennlinie des Sensors und einem Kontaktwiderstand, der beim Zusammenbau von Bare-Modul und Feinstleiterplatine entsteht. Es lässt sich ausschließen, dass das Wiederverbinden des Steckverbinders die Eigenschaften des Sensors beeinflusst.

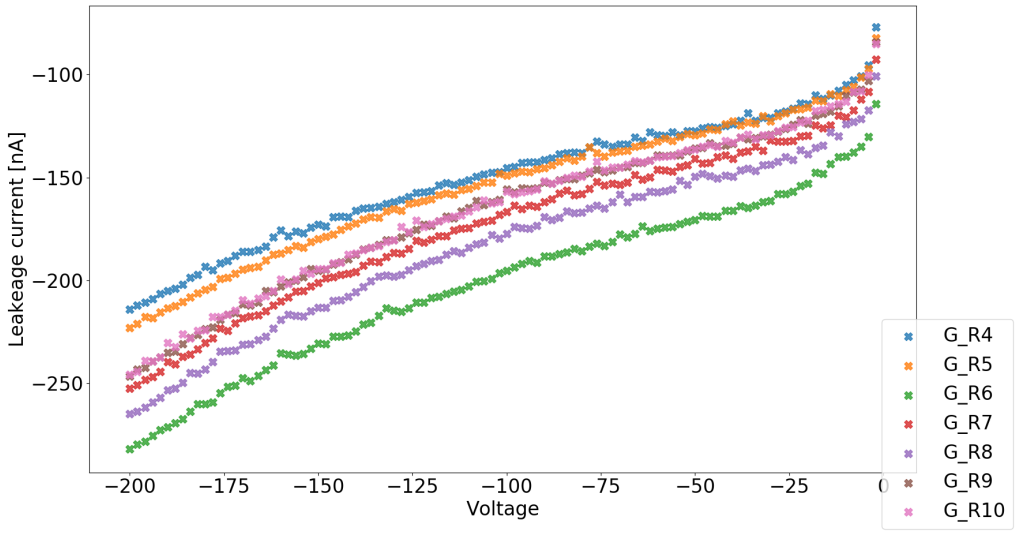

Abbildung 6.8.5: Überlagerung der IV-Kurven aller Versuchszyklen

7. Zusammenfassung

Um die Frage zu beantworten, ob das Öffnen und Schließen eines Steckverbinders auf der Oberseite des Quad-Moduls zu Beschädigungen an den Bump-Bonds zwischen Auslesechips und dem Sensor führt, wurden in dieser Arbeit 10 Wiederverbindungen des Modulsteckverbinders durchgeführt, 11 wenn man die Erstverbindung mitzählt. Nach jeder dieser Verbindungen wurde eine Reihe von Untersuchungen mit dem Modul durchgeführt, um alle Funktionsebenen des Moduls zu testen. Die ersten 3 oder 4 dieser Versuchsreihen wurden nicht ausgewertet, da das Kühlsystem nicht richtig funktionierte und so die thermische Stabilität des Moduls nicht gewährleistet war. Aus den verbliebenen Versuchsreihen kann man ableiten, dass das Öffnen und Schließen des Modulsteckverbinders keine Beschädigung am Modul verursacht. Zu sehen ist dies an der stabil bleibenden Anzahl an fehlerhaften Pixeln in der Maske, dem Analogteil der Chips sowie den aus der Analyse bestimmten toten Pixeln. Außerdem befinden sich keine der gefundenen toten Pixel im Bereich unter dem Steckverbinder. Mechanische Zerstörung der Bump-Bonds erzeugt normalerweise großflächige Bereiche, in dem der Sensor keine Daten liefert. Die wiederholte Belastung des Moduls durch das Wiederverbinden des Steckverbinders hat auch keine erkennbare Auswirkungen auf das Verhalten des Sensors unter Spannung. Die Daten zeigen alle sehr ähnliche Sperrstromsperrspannungskurven für alle Versuchszyklen, welche sich nur um weniger als 100 Nanoampere voneinander unterscheiden.

8. Danksagung

Zuletzt möchte ich mich bei einigen Personen bedanken, die mir beim Anfertigen dieser Bachelor-Arbeit geholfen haben.
Besonderer Dank gilt:

- Prof. Dr. Peter Buchholz, für die Möglichkeit, diese Bachelor-Arbeit in der Arbeitsgruppe der experimentellen Teilchenphysik durchzuführen, sowie den anspruchsvollen und interessanten Diskussionen, die mir beim Verständnis des Themas und den Ansprüchen wissenschaftlicher Arbeit halfen.
- Dr. Wolfgang Walkowiak, für Korrekturlesen, sowie schneller Hilfe bei technischen Fragen zu Python, Linux und ROOT.
- Dr. Alexey Petrukhin für die nette Arbeitsumgebung in Labor und Büro, sowie der Beschaffung und Hilfe mit der Handhabung von Labormaterialien und Laborinstrumenten.
- Meiner Familie für Korrekturlesen, moralische Unterstützung sowie der Möglichkeit, meinem Studium ohne Stress nachgehen zu können.

Danke!

Literatur

[1] *ATLAS inner detector: Technical Design Report, 1.* Technical Design Report ATLAS. CERN, Geneva, 1997.

[2] G. Aad, B. Abbott, J. Abdallah, O. Abdinov, B. Abeloos, R. Aben, O. S. AbouZeid, N. L. Abraham, H. Abramowicz, and et al. Measurements of the higgs boson production and decay rates and constraints on its couplings from a combined atlas and cms analysis of the lhc pp collision data at s = 7

$$\sqrt{s} = 7$$

and 8 tev. *Journal of High Energy Physics*, 2016(8), Aug 2016.

[3] G. Aad et al. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, 3:S08003, 2008.

[4] B. Abbott, J. Albert, F. Alberti, M. Alex, G. Alimonti, S. Alkire, P. Allport, S. Altenheiner, L.S. Ancu, E. Anderssen, A. Andreani, A. Andreazza, B. Axen, J. Arguin, M. Backhaus, G. Balbi, J. Ballansat, M. Barbero, G. Barbier, A. Bassalat, R. Bates, P. Baudin, M. Battaglia, T. Beau, R. Beccherle, A. Bell, M. Benoit, A. Bermgan, C. Bertsche, D. Bertsche, J. Bilbao de Mendizabal, F. Bindi, M. Bomben, M. Borri, C. Bortolin, N. Bousson, R.G. Boyd, P. Breugnon, G. Bruni, J. Brossamer, M. Bruschi, P. Buchholz, E. Budun, C. Buttar, F. Cadoux, G. Calderini, L. Caminada, M. Capeans, R. Carney, G. Casse, A. Catinaccio, M. Cavalli-Sforza, M. Červ, A. Cervelli, C.C. Chau, J. Chauveau, S.P. Chen, M. Chu, M. Ciapetti, V. Cindro, M. Citterio, A. Clark, M. Cobal, S. Coelli, J. Collot, O. Crespo-Lopez, G.F. Dalla Betta, C. Daly, G. D'Amen, N. Dann, V. Dao, G. Darbo, C. DaVia, P. David, S. Debieux, P. Delebecque, F. De Lorenzi, R. de Oliveira, K. Dette, W. Dietsche, B. Di Girolamo, N. Dinu, F. Dittus, D. Diyakov, F. Djama, D. Dobos, P. Dondero, K. Doonan, J. Dopke, O. Dorholt, S. Dube, D. Dzahini, K. Egorov, O. Ehrmann, K. Einsweiler, S. Elles, M. Elsing, L. Eraud, A. Ereditato, A. Eyring, D. Falchieri, A. Falou, C. Fausten, A. Favareto, Y. Favre, S. Feigl, S. Fernandez Perez, D. Ferrere, J. Fleury, T. Flick, D. Forshaw, D. Fougeron, L. Franconi, A. Gabrielli, R. Gaglione, C. Gallrapp, K.K. Gan, M. Garcia-Sciveres, G. Gariano, T. Gastaldi, I. Gavrilenko, A. Gaudiello, N. Geffroy, C. Gemme, F. Gensolen, M. George, P. Ghislain, N. Giangiacomi, S. Gibson, M.P. Giordani, D. Giugni, H. Gjersdal, K.W. Glitza, D. Gnani, J. Godlewski, L. Gonella, S. Gonzalez-Sevilla, I. Gorelov, A. Gorišek, C. Gössling, S. Grancagnolo, H. Gray, I. Gregor, P. Grenier, S. Grinstein, A. Gris, V. Gromov, D. Grondin, J. Grosse-Knetter, F. Guescini, E. Guido, P. Gutierrez, G. Hallewell, N. Hartman, S. Hauck, J. Hasi, A. Hasib, F. Hegner, S. Heidbrink, T. Heim, B. Heinemann, T. Hemperek, N.P. Hessey, M. Hetmánek, R.R. Hinman, M. Hoeferkamp, T. Holmes, J. Hostachy, S.C. Hsu, F. Hügging, C. Husi, G. Iacobucci, I. Ibragimov, J. Idarraga, Y. Ikegami, T. Ince, R. Ishmukhametov, J. M. Izen, Z. Janoška, J. Janssen, L. Jansen, L. Jeanty, F. Jensen, J. Jentzsch, S. Jezequel,

J. Joseph, H. Kagan, M. Kagan, M. Karagounis, R. Kass, A. Kastanas, C. Kenney, S. Kersten, P. Kind, M. Klein, R. Klingenberg, R. Kluit, M. Kocian, E. Koffeman, O. Korchak, I. Korolkov, I. Kostyukhina-Visoven, S. Kovalenko, M. Kretz, N. Krieger, H. Krüger, A. Kruth, A. Kugel, W. Kuykendall, A. La Rosa, C. Lai, K. Lantzsch, C. Lapoire, D. Laporte, T. Lari, S. Latorre, M. Leyton, B. Lindquist, K. Looper, I. Lopez, A. Lounis, Y. Lu, H.J. Lubatti, S. Maeland, A. Maier, U. Mallik, F. Manca, B. Mandelli, I. Mandić, D. Marchand, G. Marchiori, M. Marx, N. Massol, P. Mättig, J. Mayer, G. Mc Goldrick, A. Mekkaoui, M. Menouni, J. Menu, C. Meroni, J. Mesa, S. Michal, S. Miglioranzi, M. Mikuž, A. Miucci, K. Mochizuki, M. Monti, J. Moore, P. Morettini, A. Morley, J. Moss, D. Muenstermann, P. Murray, K. Nakamura, C. Nellist, D. Nelson, M. Nessi, R. Nisius, M. Nordberg, F. Nuiry, T. Obermann, W. Ockenfels, H. Oide, M. Oriunno, F. Ould-Saada, C. Padilla, P. Pangaud, S. Parker, G. Pelleriti, H. Pernegger, G. Piacquadio, A. Picazio, D. Pohl, A. Polini, X. Pons, J. Populе, X. Portell Bueso, K. Potamianos, M. Povoli, D. Puldon, Y. Pylypchenko, A. Quadt, B. Quayle, F. Rarbi, F. Ragusa, T. Rambure, E. Richards, C. Riegel, B. Ristic, F. Rivière, F. Rizatdinova, O. RØhne, C. Rossi, L.P. Rossi, A. Rovani, A. Rozanov, I. Rubinskiy, M.S. Rudolph, A. Rummler, E. Ruscino, F. Sabatini, D. Salek, A. Salzburger, H. Sandaker, M. Sannino, B. Sanny, T. Scanlon, J. Schipper, U. Schmidt, B. Schneider, A. Schorlemmer, N. Schroer, P. Schwemling, A. Sciuccati, S. Seidel, A. Seiden, P. Šícho, P. Skubic, M. Sloboda, D.S. Smith, M. Smith, A. Sood, E. Spencer, M. Stramaglia, M. Strauss, S. Stucci, B. Stugu, J. Stupak, N. Styles, D. Su, Y. Takubo, J. Tassan, P. Teng, A. Teixeira, S. Terzo, X. Therry, T. Todorov, M. Tomášek, K. Toms, R. Travaglini, W. Trischuk, C. Troncon, G. Troska, S. Tsiskaridze, I. Tsurin, D. Tsybychev, Y. Unno, L. Vacavant, B. Verlaat, E. Vigeolas, M. Vogt, V. Vrba, R. Vuillermet, W. Wagner, W. Walkowiak, R. Wang, S. Watts, M.S. Weber, M. Weber, J. Weingarten, S. Welch, S. Wenig, M. Wensing, N. Wermes, T. Wittig, M. Wittgen, T. Yildizkaya, Y. Yang, W. Yao, Y. Yi, A. Zaman, R. Zaidan, C. Zeitnitz, M. Ziolkowski, V. Zivkovic, A. Zoccoli, and L. Zwalinski. Production and integration of the ATLAS insertable b-layer. *Journal of Instrumentation*, 13(05):T05008–T05008, may 2018.

[5] M Backhaus. Characterization of the fe-i4b pixel readout chip production run for the atlas insertable b-layer upgrade. *Journal of Instrumentation*, 8(03):C03013–C03013, Mar 2013.

[6] Oliver Sim Brüning, Paul Collier, P Lebrun, Stephen Myers, Ranko Ostojic, John Poole, and Paul Proudlock. *LHC Design Report*. CERN Yellow Reports: Monographs. CERN, Geneva, 2004.

[7] M Capeans, G Darbo, K Einsweiller, M Elsing, T Flick, M Garcia-Sciveres, C Gemme, H Pernegger, O Rohne, and R Vuillermet. ATLAS Insertable B-Layer Technical Design Report. Technical Report CERN-LHCC-2010-013. ATLAS-TDR-19, Sep 2010.

[8] S. Chatrchyan et al. The CMS Experiment at the CERN LHC. *JINST*, 3:S08004, 2008.

[9] ATLAS Collaboration. Technical Design Report for the ATLAS Inner Tracker Pixel Detector. Technical Report CERN-LHCC-2017-021. ATLAS-TDR-030, CERN, Geneva, Sep 2017.

[10] ATLAS Collaboration. Technical Design Report for the ATLAS Inner Tracker Pixel Detector. Technical Report CERN-LHCC-2017-021. ATLAS-TDR-030, CERN, Geneva, Sep 2017.

[11] Apollinari G., Béjar Alonso I., Brüning O., Fessia P., Lamont M., Rossi L., and Tavian L. *High-Luminosity Large Hadron Collider (HL-LHC): Technical Design Report V. 0.1.* CERN Yellow Reports: Monographs. CERN, Geneva, 2017.

[12] A. Petrukhin on behalf of the ATLAS Siegen group. A detachable tail module for the phase ii atlas pixel upgrade. In *Nuclear Science Symposium*. IEEE, Oct/Nov 2019. Manchester.

[13] Alessandro La Rosa. Atlas pixel detector: Operational experience and run-1 to run-2 transition, 2014.

A. Anhang

A.1. Globale Konfiguration

Hier eine genauere Beschreibung des nötigen Codes, um einen Chip der USBPix3-Software zugänglich zu machen:

```
# *** MMC3 board with max. 8 FEI4s ***
dut : dut_mmc3_8chip_eth.yaml # DUT hardware configuration (.yaml file).....
dut_configuration : dut_configuration_mmc3_8chip_eth.yaml #....
working_dir : G_R10 # The name of the output data folder.

modules :
    module_0 :
        activate : True
        configuration :  # FE configuration file, text (.cfg) or ....
        flavor : fei4b # FEI4 flavor/type for initial configuration....
        chip_address : 1 # Chip Address for initial configuration....
        FIFO : SITCP_FIFO # As defined in DUT configuration.
        RX : DATA_CH0 # As defined in DUT configuration.
        rx_channel : 0 # As implemented in the firmware.
        TX : CMD_CH0_TO_CH7 # As defined in DUT configuration.
        tx_channel : 0 # As implemented in the firmware.
        TDC: TDC_CH0 # As defined in DUT configuration.
        tdc_channel : 1 # As implemented in the firmware.
        TLU : TRIGGER_CH0_TO_CH7 # As defined in DUT configuration.
```

Die Konfigurations-Datei enthält mehrere Code-Blöcke ähnlich dem obigen für verschiedene externe Hardware-Konfigurationen. Für Module mit nur einer command line und der MMC3-Karte benutzt man den Block mit Namen:

```
'''MMC3 board with max. 8 FEI4'''
```

Die Punkte dut und dut_configuration bestimmen Hardware-Konfigurationen des Testsystems und werden nicht verändert.
Das Stichword working_dir: gibt den Ausgabepfad für die Daten der Scans an. Wenn kein vollständiger Pfad angegeben wird, erstellt sich der Ordner im pybar-Ordner des lokalen Git-Ordners.
Unter dem Stichword modules muss man die auszulesende Chip Hardware angeben .
Das folgende Stichword module_x startet den Code Block für einen Chip und wird durchnummeriert mit 0 bis max.7 .
activated: gibt im Boolean Format an, ob ein Chip gelesen/angesprochen werden soll (Großschreibung ist hier notwendig).
Mit configuration: kann man eine feste Chip-Konfigurations-Datei angeben oder wenn leer gelassen, wird die neuste, existierende Chip-Konfigurations-Datei benutzt.

flavor: gibt die Version des benutzten FEI4-Chips an. Mögliche Versionen sind 'fei4a' oder 'fei4b'. Das benutzte Quad-Modul hat nur Chips vom Typ 'fei4b'.
chip_address: ist die physikalische Adresse des Chips, welche durch wirebonds bestimmt ist. 0-8 sind mögliche Angaben, wobei 8 bedeutet, dass der Chip auf alle Befehle reagiert, wodurch diese Option nur für einen einzigen angeschlossenen/aktivierten Chip funktioniert.
Für RX , rx_channel und TDC muss die angegebene Zahl mit dem Kanal der MMC3-Karte übereinstimmen, mit dem der Chip über Ethernet-Kabel verbunden ist.
tx_channel muss dem Kanal der MMC3-Karte entsprechen, der die command line des Chips/Moduls besitzt.
TX und TLU müssen bei dieser Hardware-Einstellung nicht angepasst werden, da nur ein Kanal die command line besitzt.

A.2. scan_digital.py

```
class DigitalScan(Fei4RunBase):
    '''Digital scan
    '''
    _default_run_conf = {
        "broadcast_commands": True,
        "threaded_scan": True,
        "mask_steps": 3,  # mask steps
        "n_injections": 100,  # number of injections
        "use_enable_mask": False  # if True, use Enable mask during scan, ...
        }
```

Das scan_digital.py Skript testet den digitalen Teil der Auslesekette des Chips. Dazu wird per Kommando das Hit-Bit/Hit-Register auf 1/True gesetzt und das zugehörige TOT-Register mit 13 überschrieben. Dies wird für jeden Pixel entsprechend der n_injections Variable (100) mal ausgeführt und ausgelesen. Da das Auslesen aller Pixel gleichzeitig durch Bandweitenbegrenzungen nicht möglich ist, wird die Pixelmatrix in mask_steps (min. 3) Teilstücke unterteilt. Die Stichworte broadcast_commands und threaded_scans erlauben/verbieten die parallele Auslese von mehren Chips mit geteilter command_lane und das parallele Auslesen/Ansprechen von mehreren command_lanes. Das Stichwort use_enable_mask erzwingt die Benutzung der Enable-Maske wenn True oder der gesamten Pixelmatrix wenn False.

A.3. scan_analog.py

```
class AnalogScan(Fei4RunBase):
    '''Analog scan
    '''
    _default_run_conf = {
```

```
        "broadcast_commands": True,  # keep this False,...
        "threaded_scan": True,
        "mask_steps": 3,  # mask steps,...
        "n_injections": 100,  # number of injections[100]
        "scan_parameters": [('PlsrDAC', 280)],  # the PlsrDAC setting
        "use_enable_mask": True,  # if True, use Enable mask during scan,...
```

Das Script scan_analog.py testet den analogen Teil der Auslesekette des Chips. Dazu wird mittels einer chip-internen Schaltung eine bekannte Menge an Elektronen in die Auslesezelle eines Pixels injiziert und die Antwort des Chips ausgelesen. Die gesamte Pixelmatrix wird wie beim digitalen Scan abgefragt. Die Anzahl der injizierten Elektronen wird in scan_parameters mit PlsrDAC angegeben wobei 50 PlsrDAC ungefähr 2800 Elektronen entspricht. Die Standardanzahl ist 280 PlsrDAC, um sicher zu gehen, dass ein funktionierender Pixel auch jedes Mal ausgelöst wird.

A.4. tune_fei4.py

```
class Fei4Tuning(GdacTuning, TdacTuning, FeedbackTuning, FdacTuning):
    '''Fully automatic FEI4 Tuning'''
    _default_run_conf = {
        "broadcast_commands": False,  # do not change, keep False...
        "threaded_scan": True,  # if True, parallelize tuning
        # tuning parameters
        "target_threshold": 50,  # target threshold
        "target_charge": 280,  # target charge
        "target_tot": 5,  # target ToT
        ....
    }
```

Das tune_fei4.py Skript passt die Antwort der Pixelmatrix auf ein bestimmtes target_threshold (50 DAC / 2800 e^-) an injizierter Ladung an. Dies bedeutet, dass nach dem Tuning beim Einfügen von einer Ladung entsprechend target_threshold jeder Pixel idealerweise mit 50% Wahrscheinlichkeit aktiv wird. Dies wird erreicht, indem zuerst die Verstärker im analogen Teil der Pixel angepasst werden, so dass der Mittelwert der Verteilung bei target_threshold liegt. Danach wird die Verteilung fokussiert, indem die Lokalen/ pro Pixel Variablen (TDAC) angepasst werden. Die vorhergehenden Schritte werden mehrmals wiederholt, da die verschiedenen Methoden sich beeinflussen.

A.5. scan_threshold.py

```
class ThresholdScan(Fei4RunBase):
    _default_run_conf = {
        "broadcast_commands": True,
        "threaded_scan": True,
```

```
    "mask_steps": 3,  # mask steps ....
    "n_injections": 100,  # number of injections per PlsrDAC step
    "scan_parameters": [('PlsrDAC', [20, 70])],  # the PlsrDAC range
    "step_size": 1,  # step size of the PlsrDAC during scan
    .....
}
```

Das Skript scan_threshold.py ist dafür da, die Funktionsfähigkeit und Qualität des FEI4 Tunings zu überprüfen. Dazu werden analoge Scans mit verschieden PlsrDAC Parametern durchgeführt. Im oben angegebenen Beispiel von 20 bis 70 PlsrDAC mit einer Stufenbreite von step_size (1). Das heißt, dass 50 analoge Scans pro Threshold scan durchgeführt werden.

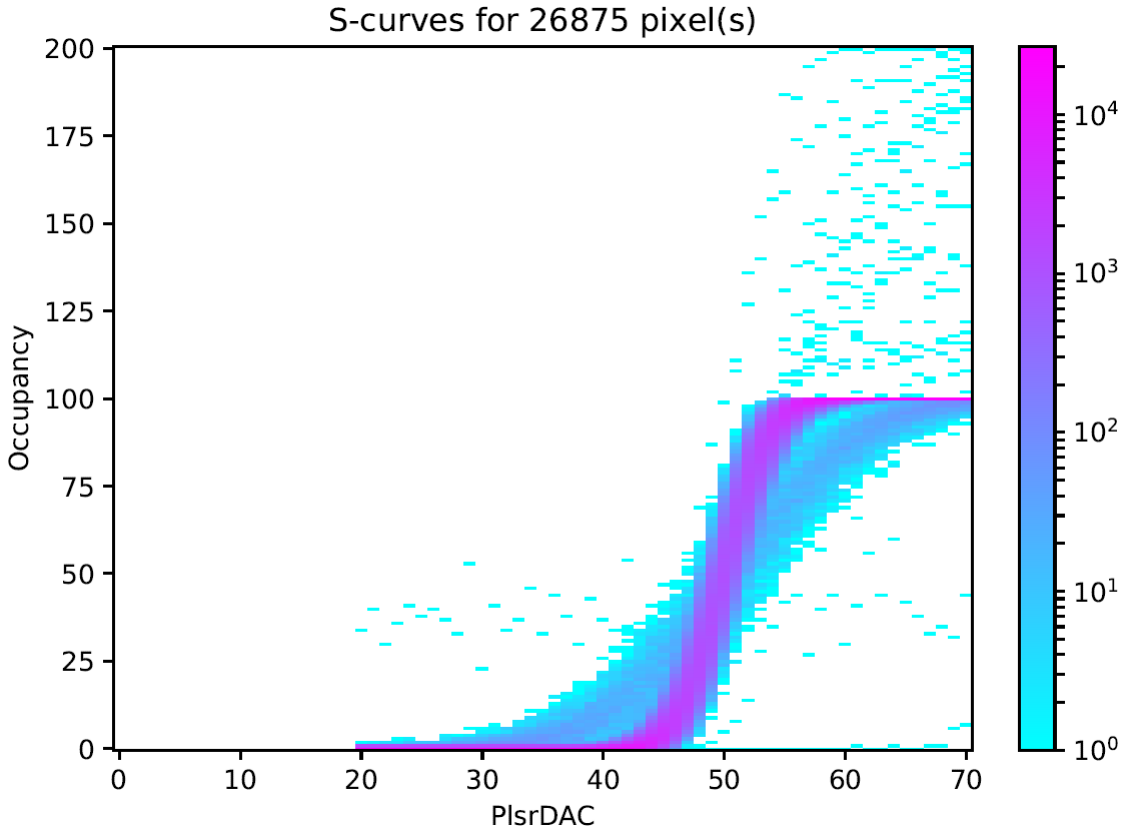

Abbildung A.5.1: Das S-Kurven Diagramm eines Chips (50 PlsrDAC tuning)

Im idealen Fall für FE-I4B bedeutet dies 26880 Pixel, die für jeden PlsrDAC Schritt eine Occupancy/Trefferzahl von 0-100 haben. An die Daten jedes Pixels werden dann S-Kurven angepasst. Die S-Kurven entsprechen der Kommulativverteilung einer Gaussverteilung, d.h. für einen perfekt getunten Chip erwartet man sehr steile Kurven, die ihre halbe Höhe am vorher festgelegten Tuning Parameter (50 PlsrDAC) erreichen. Da

es aber keine perfekten Chips gibt, werden die Punkte halber Höhe aller Kurven gegen ihre PlsrDAC Position aufgetragen und die Qualität über die Breite des Gaussfits auf diese Daten bestimmt.

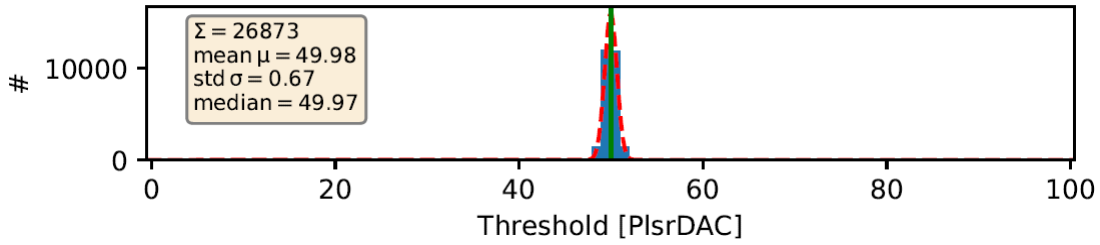

Abbildung A.5.2: Das Histogramm der S-Kurven Wendepunkte

A.6. scan_fei4_self_trigger.py

```
class Fei4SelfTriggerScan(Fei4RunBase):

    _default_run_conf = {
        "broadcast_commands": True,
        "threaded_scan": True,
        ....
        "no_data_timeout": 700 ,  # no data timeout after which the  ...
        "scan_timeout": 300,  # timeout for scan after which the scan ...
    }
```

Das Skript scan_fei4_self_trigger.py versetzt die Chips in einen Selbstauslöser-Modus. Das heißt, wenn ein Pixel des Chips einen Hit registriert wird chip-intern ein Auslesesignal erzeugt, das dann die Daten aus dem Chip in die Ausleseelektronik weiterleitet. Dieser Scan benötigt echte Hit-Daten, d.h. der Sensor muss genügend Elektronen durch einen Teilchentreffer erzeugen, um die Chipelektronik auszulösen. Um dies zu erreichen wird eine radioaktive Quelle benötigt, die in der Lage ist, die flexible Leiterplatine zu durchdringen. Bei den Scans in dieser Arbeit wurde eine Sr^{90} Quelle benutzt, sowie eine Bestrahlungsdauer von 5 Minuten (300 Sekunden) entsprechend scan_timeout. Der Parameter no_data_timeout muss länger sein als der Parameter scan_timeout, da die Zeitmessung nur erfolgt, wenn Daten vom Chip empfangen/erzeugt werden. Somit ist no_data_timeout die maximale Dauer des Scans , wenn keine kontinuierliche Datennahme erfolgt.
Das Ergebnis ist eine 80x336 Matrix, die die Anzahl der Treffer über die gesamte Zeit des Scans enthält. Pixel, die hier keine Treffer in 5 Minuten verzeichnen, werden in der Auswertung als no_hit Pixel bezeichnet.

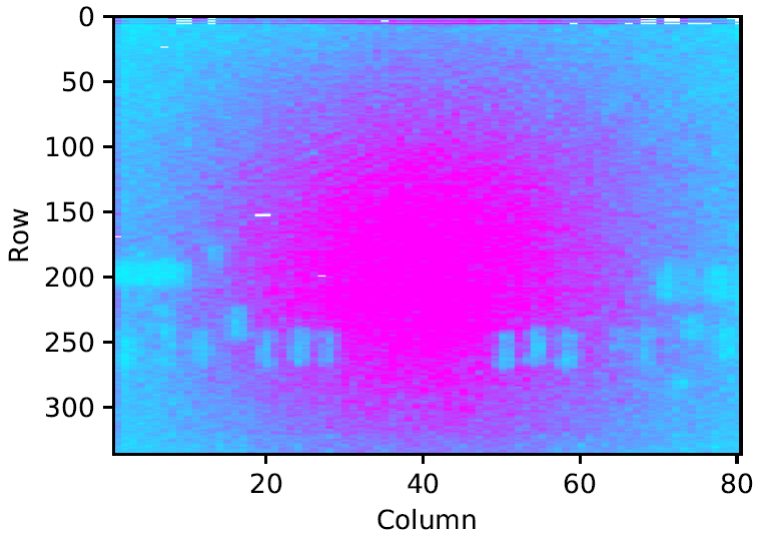

Abbildung A.6.1: Die farbkodierte Treffermatrix eines Selbstauslöser Scans mit radioaktiver Quelle

A.7. scan_iv.py

```
class IVScan(Fei4RunBase):
    _default_run_conf = {
        "broadcast_commands": False,
        "threaded_scan": False,
        "voltages": np.arange(-2, -201, -2).tolist(),  # voltage steps ...
        "max_leakage": 1e-6,  # is set for 10uA, scan aborts if ...
        "max_voltage": -200,  # for safety, scan aborts if voltage is ...
        "minimum_delay": 0.5,  # minimum delay between current ...
        "bias_voltage": -10  # if defined ramp bias to bias voltage ...
        }
```

Der IV-Scan ist der einzige Scan, der nicht die FEI4-Chips benutzt. Er spricht mit dem Hochspannungsnetzteil, das die Sperrspannnung des Sensors erzeugt. Der Scan dient der Bestimmung der Dioden-Kennlinie des Sensors in Sperrrichtung, wobei die negative Spannung entsprechend des Parameters Voltages, welches eine Python-Liste von ganzen Zahlen ist, abgefahren wird. Diese Liste startet bei -2 und geht in -2 Schritten zu -200. max_leakage gibt den maximalen Sperrstrom in Ampere an, der erlaubt ist, bevor der Scan abgebrochen wird, um den Sensor nicht zu beschädigen. Ähnlich ist der Parameter max_voltage, angegeben in Volt, auch nur ein Sicherheitskriterium, da die maximale Spannung auch schon in Voltages definiert ist. minimum_delay gibt die minimale Wartezeit in Sekunden an, die zwischen zwei Sperrstrom-Messungen vergehen muss, um die Spannung zu erhöhen. Diese Zeit kann länger sein, da der Sperrstrom eine Weile benötigt, um sich zu stabilisieren, d.h. die Messung schreitet nur voran, wenn der Sperrstrom

in einem Messschritt konstant bleibt. Zuletzt gibt der Parameter bias_voltage an, auf welchen Betrag die Sperrspannung am Ende des Scans wieder reduziert wird.

A.8. Pythonmodule

Folgende Python 3 Module oder Python 2.7 Backports müssen installiert werden, um das automatisierte Analyseskript auszuführen:

- hdf5,
- os,
- numpy,
- pathlib2,
- tables,
- mathplotlib.

A.9. Config.py

Das Skript Config.py enthält nur eine Funktion, welche ein fest codiertes Python-Wörterbuch ausgibt. Ein Python-Wörterbuch ist eine Liste von Objekten, bei der jedem enthaltenen Objekt ein Identifikationswort/String zugeordnet ist. Durch dieses Wort lassen sich Objekte einfach und schnell identifizieren.

```
config = {
        "OutputDir": "E:/Output",
        "InputDir" : "E:/pybar",
        "Data": {"Run0": {
                    "Name"          : "G_R4" ,
                    "RN_Digital"    : 2 ,
                    "RN_Analog"     : 3 ,
                    "RN_Source"     : [24,16,18,21] ,
                    "RN_CONSource"  : 26 ,
                    "RN_IV"         : 27
                },
```

Ein Python-Wörterbuch wird durch geschweifte Klammern deklariert und Objekte innerhalb des Wörterbuchs sind mit ',' voneinander getrennt. Ein Wörterbuchobjekt beginnt mit einem String markiert durch "" und ordnet diesem ein beliebiges Objekt zu, welches von dem String mittels ':' getrennt wird. OutputDir gibt den Ausgabe-Ordner für das Analyseskript an. InputDir gibt den Ort an, an dem die Versuchsreihen-Daten gespeichert sind. Zuletzt wird unter Data ein weiteres Wörterbuch hinterlegt, welches Wörterbücher zu den Versuchsreihen enthält. Die Versuchsreihen haben einen Namen (Name), der mit dem Ordnernamen der Versuchsreihe übereinstimmen muss und eine Reihe von Run Numbers, welche die Nummern der Scans angeben, die ausgewertet werden sollen.

A.10. Erstverbindung 0

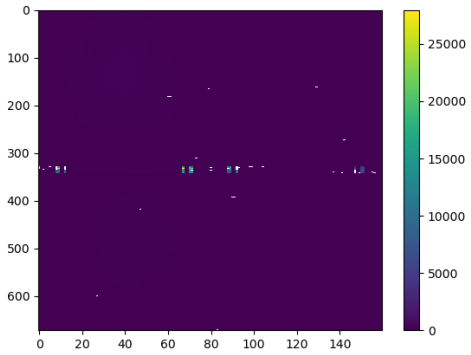

Abbildung A.10.1: Kombiniertes Bild der einzelnen Chipquellentests für Erstverbindung 0

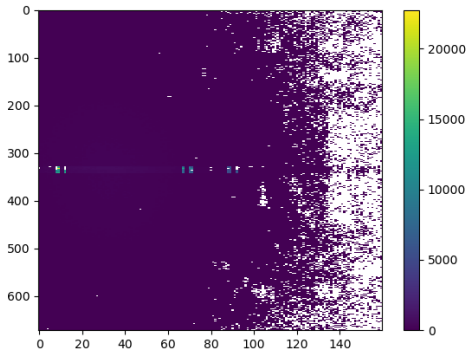

Abbildung A.10.2: Kombiniertes Bild des Tests mit radioaktiver Quelle nahe des Steckverbinders für Erstverbindung 0

Defekte Pixel auf digitaler Chipebene

Es wurden keine defekten Pixel gefunden.

Defekte Pixel auf analoger Chipebene

Es wurden 16 defekte Pixel gefunden.

Chip 1	Chip 2	Chip 3	Chip 4
(24,6)	(1,42)	(56,11)	(82,47)
(152,18)	(173,30)	(57,10)	(83,27)
(152,19)	(232,37)	(57,11)	(116,66)
(153,18)			(264,27)
(199,26)			(281,69)

Defekte Pixel auf Sensorebene

Es wurden 4 defekte Pixel in Chip 1 gefunden. 33 Pixel auf gesamten Modul maskiert.

Chip 1	(0,77)	(2,79)	(6,79)	(169,0)

A.11. Wiederverbindung 1

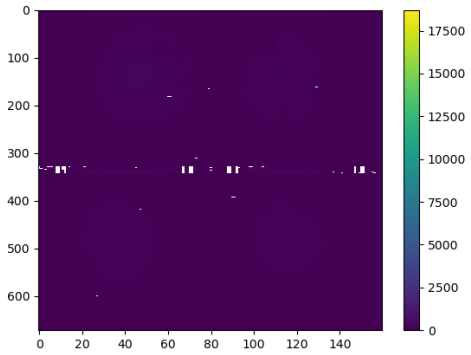

Abbildung A.11.1: Kombiniertes Bild der einzelnen Chipquellentests für Wiederverbindung 1

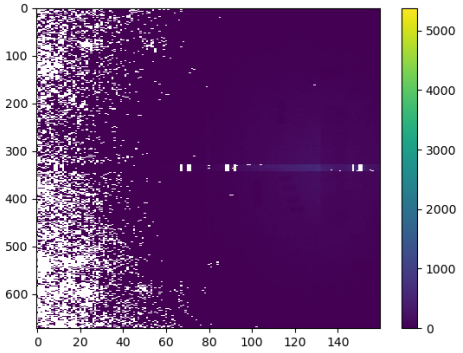

Abbildung A.11.2: Kombiniertes Bild des Tests mit radioaktiver Quelle nahe des Steckverbinders für Wiederverbindung 1

Defekte Pixel auf digitaler Chipebene

Es wurden keine defekten Pixel gefunden.

Defekte Pixel auf analoger Chipebene

Es wurden 14 defekte Pixel gefunden.

Chip 1	Chip 2	Chip 3	Chip 4
(24,6)	(1,42)	(56,11)	(83,47)
(152,18)	(173,30)	(57,10)	(116,66)
(152,19)		(57,11)	(264,27)
(153,18)			(281,69)
(199,26)			

Defekte Pixel auf Sensorebene

Es wurden 3 defekte Pixel in Chip 1 gefunden. 128 Pixel auf gesamten Modul maskiert.

Chip 1	(2,79)	(6,79)	(169,0)

A.12. Wiederverbindung 2

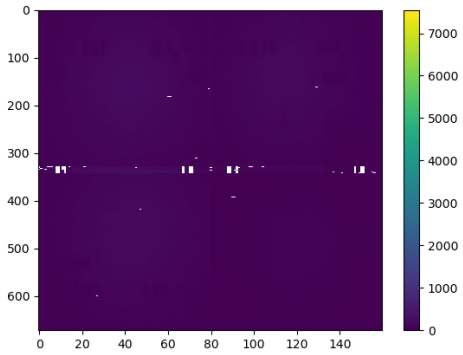

Abbildung A.12.1: Kombiniertes Bild der einzelnen Chipquellentests für Wiederverbindung 2

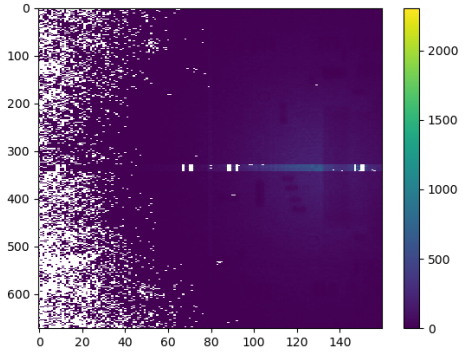

Abbildung A.12.2: Kombiniertes Bild des Tests mit radioaktiver Quelle nahe des Steckverbinders für Wiederverbindung 2

Defekte Pixel auf digitaler Chipebene

Es wurden keine defekten Pixel gefunden.

Defekte Pixel auf analoger Chipebene

Es wurden 13 defekte Pixel gefunden.

Chip 1	Chip 2	Chip 3	Chip 4
(24,6)	(173,30)	(56,11)	(83,27)
(152,18)		(57,10)	(116,66)
(152,19)		(57,11)	(264,27)
(153,18)			(281,69)
(199,26)			

Defekte Pixel auf Sensorebene

Es wurden 2 defekte Pixel in Chip 1 gefunden. 138 Pixel auf gesamten Modul maskiert.

Chip 1	(6,79)	(169,0)

A.13. Wiederverbindung 3

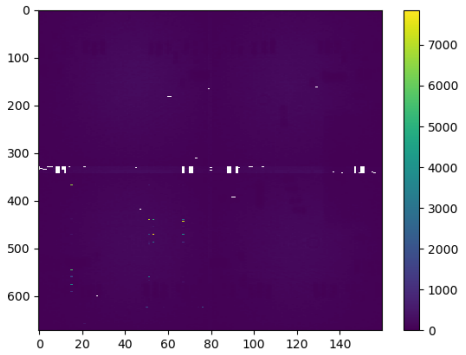

Abbildung A.13.1: Kombiniertes Bild der einzelnen Chipquellentests für Wiederverbindung 3

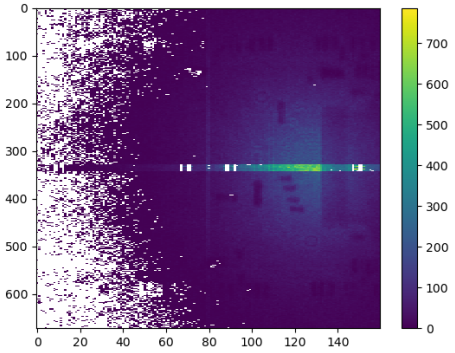

Abbildung A.13.2: Kombiniertes Bild des Tests mit radioaktiver Quelle nahe des Steckverbinders für Wiederverbindung 3

Defekte Pixel auf digitaler Chipebene

Es wurden keine defekten Pixel gefunden.

Defekte Pixel auf analoger Chipebene

Es wurden 13 defekte Pixel gefunden.

Chip 1	Chip 2	Chip 3	Chip 4
(24,6)	(173,30)	(56,11)	(83,27)
(152,18)		(57,10)	(116,66)
(152,19)		(57,11)	(264,27)
(153,18)			(281,69)
(199,26)			

Defekte Pixel auf Sensorebene

Es wurden 5 defekte Pixel in Chip 1 gefunden. 139 Pixel auf gesamten Modul maskiert.

Chip 1	(0,77)	(0,79)	(2,79)	(6,79)	(169,0)

Quellencode

Main.py

```
#-------------------------------------------------------------------------------
# Imports functions for mainscript
import os
import Utils.IO as IO
import Utils.Plotter as Plot
from Utils.Config import Config

#-------------------------------------------------------------------------------
#loads Config dictionary from Config.py in Utils
CFG=Config()
#pixel count variable
SourceDead=[]
AnalogDead=[]
DigitalDead=[]
Labels=[]
IVList=[]
# main loop over Runs
for Run in CFG["Data"].items():
    print(Run[1]["Name"])
    Labels.append(Run[1]["Name"])
    IVList.append(Run[1]["RN_IV"])
    # path string generation for In and output Folders
    InPath=CFG["InputDir"]+"/"+Run[1]["Name"]
    Outpath=CFG["OutputDir"]+"/"+Run[1]["Name"]
    # Generating output Folder if it does not exist already
    if not os.path.exists(Outpath):
        os.makedirs(Outpath)
    # generation of Dictionary for all .h5 files folowing folder
    #structure
    ModuleDict=IO.makeFiledictH5(InPath)
    # Analysis of Digital scans
    # Transfer of Config RN Number
    RN_Digital= Run[1]["RN_Digital"]
    for module in ModuleDict:
        print(module+"_Digital_scan")
        # search for Relevant scan via Run Number string
        #(i added for interpreted data)
        Scanpath=ModuleDict[module][str(RN_Digital)+"i"]
        Scanpath=InPath+"/"+module+"/"+Scanpath
        #generating File with Pixel that have exactly 0 hits in
```

```
    #digital scan
    IO.searchAryData_E(Scanpath,Outpath+"/"+module+"_DigitalZero.dat",0)
    IO.GetMaskData(Scanpath,Outpath+"/"+module+"_Mask.dat",0)
# Analysis of Analog scans
RN_Analog= Run[1]["RN_Analog"]
for module in ModuleDict:
    print(module+" Analog_scan")
    Scanpath=ModuleDict[module][str(RN_Analog)+"i"]
    Scanpath=InPath+"/"+module+"/"+Scanpath
    ##generating File with Pixel that have less than 95 hits in
    #Analog scan
    IO.searchAryData_L(Scanpath,Outpath+"/"+module+"_AnalogLow.dat",95)
#analysis of Source Scans
RN_Source=Run[1]["RN_Source"]
Quadlist=[]
for module in ModuleDict:
    print(module+" Source_scan")
    MN=int(module.split("_")[1])
    Scanpath=ModuleDict[module][str(RN_Source[MN])+"i"]
    Scanpath=InPath+"/"+module+"/"+Scanpath
    IO.searchAryData_E(Scanpath,Outpath+"/"+module+"_SourceLow.dat",0)
    IO.GetData(Scanpath,Outpath+"/"+module+"_SourceData.dat")
    Quadlist.append((module,Scanpath))
#Plotting of Full module with Source data and CONData
##########
Plot.plotQuadfromDataList(Quadlist,Outpath+"_Quadplot.png")
RN_CONSource=Run[1]["RN_CONSource"]
Quadlist=[]
for module in ModuleDict:
    print(module+" CON_scan")
    Scanpath=ModuleDict[module][str(RN_CONSource)+"i"]
    Scanpath=InPath+"/"+module+"/"+Scanpath
    Quadlist.append((module,Scanpath))
Plot.plotQuadfromDataList(Quadlist,Outpath+"_CONPlot.png")
#data Analysis for "Dead" pixel in each layer of the Chips
IO.ModuleCompare(Outpath,["_SourceLow.dat","_AnalogLow.dat"],
"_SourceAnalogRemoved.dat")
IO.ModuleCompare(Outpath,["_SourceAnalogRemoved.dat","_DigitalZero.dat"],
"_SourceAnDRemoved.dat")
IO.ModuleCompare(Outpath,["_AnalogLow.dat","_DigitalZero.dat"],
"_AnalogDigitalRemoved.dat")
#Writing count into python list for Plotting
SourceDead.append(IO.ModuleCount(Outpath,"_SourceAnDRemoved.dat"))
```

```
        AnalogDead.append(IO.ModuleCount(Outpath,"_AnalogDigitalRemoved.dat"))
        DigitalDead.append(IO.ModuleCount(Outpath,"_DigitalZero.dat"))
print(SourceDead)
print(AnalogDead)
print(DigitalDead)
#plotting Bar Diagramms
Title=["Defekte Pixel","","","Pixelanzahl",
"Analoge Pixel mit 95 oder weniger Treffern"]
Plot.plotbarM3(CFG["OutputDir"
]+"\ChipPixelCount.PNG",Labels, AnalogDead, Names=Title)
Title=["Defekte Pixel","","","Pixelanzahl",
"Defekte Pixel auf Sensor und Bump-Bond-Ebene"]
Plot.plotbarM3(CFG["OutputDir"]+
"\ChipPixelCount2.PNG",Labels, SourceDead, Names=Title)
#plotting IV Diagramm
Plot.plotIV(IVList, Labels,CFG["InputDir"],CFG["OutputDir"]+"/IV.png")
Data=["E:/Output/G_R10/module_0_AnalogDigitalRemoved.dat",
"E:/Output/G_R10/module_1_AnalogDigitalRemoved.dat",
"E:/Output/G_R10/module_2_AnalogDigitalRemoved.dat",
"E:/Output/G_R10/module_3_AnalogDigitalRemoved.dat"]
Temp=Plot.genQuadAryfromData2(Data)
Plot.plotAry2(Temp, CFG["OutputDir"]+"/Analog.png")
Data=["E:/Output/G_R10/module_0_SourceAnalogRemoved.dat",
"E:/Output/G_R10/module_1_SourceAnalogRemoved.dat",
"E:/Output/G_R10/module_2_SourceAnalogRemoved.dat",
"E:/Output/G_R10/module_3_SourceAnalogRemoved.dat"]
Temp=Plot.genQuadAryfromData2(Data)
Plot.plotAry2(Temp, CFG["OutputDir"]+"/Sourcedead.png")
Data=["E:/Output/G_R10/module_0_Mask.dat",
"E:/Output/G_R10/module_1_Mask.dat",
"E:/Output/G_R10/module_2_Mask.dat",
"E:/Output/G_R10/module_3_Mask.dat"]
Temp=Plot.genQuadAryfromData2(Data)
Plot.plotAry2(Temp, CFG["OutputDir"]+"/Mask.png")
```

IO.py

```
import os
import tables
import numpy as np
import pathlib2 as pl2

def makeFiledictH5(WorkingDir):
```

```
    #generate Pathlib2 Path object
    cwd=pl2.Path(WorkingDir)
    #initialize output dictionary
    Output={}
    #Flag for name change of subfolder
    module=""
    #Loop over all file which fit Searchstring
    for file in cwd.glob("**/*.h5"):
        #Convert Path object to Folder List
        Path=str(file.absolute())
        Path=Path.split(os.sep)
        #Making of new subdictionary for each Module
        if not Path[3]==module:
            Output[Path[3]]={}
            module=Path[3]
        # generation entry for RunNumber(RN) with Path string to Object
        RN=Path[4].split("_")[0]
        # add "i" to RN if the .h5 file is interpreted
        if Path[4].split("_")[len(Path[4].split("_"))-1]=="interpreted.h5":
            RN=RN+"i"
        Output[Path[3]][RN]=Path[4]
    #Returning Full Dictionary of Found Files
    return(Output)

def h5_node_reader(url,node):
    "Takes input HDF5 file and outputs objects up to one object deep.
    node is the object identifier exampble : /HistOcc ,
    input need to be string"
    # opens object tree inside of .h5 file with
    the help of the python packages pytables
    h5file = tables.open_file(url,mode='r', driver="H5FD_CORE")
    # navigates to object specivied in "node"
    example: "/HistOcc" wich is CArray type objekt
    Branch = h5file.get_node(node)
    # Read/transfers CArray type object to output variable
    R = Branch.read()
    # Close file to release Memory
    h5file.close()
    # Output Return Variable "R"
    return(R)

def searchAryData_E(url,outputpath,Target):
    Array=h5_node_reader(url, "/HistOcc")
```

```
    url1=url.split(os.sep)
    url1=url1[len(url1)-1].split("_")
    url1.pop(len(url1)-1)
    url1="_".join(url1)+".h5"
    Mask=h5_node_reader(url1, "/configuration/Enable")
    size=Array.shape
    F1=open(outputpath,"w")
    F1.write("X/L:Y/L:Z/L\n")
    F1.write("-1 -1 {}\n".format(size[0]))
    F1.write("-1 -1 {}\n".format(size[1]))
    for x in range(size[0]):
        for y in range(size[1]):
            Z=Array.item(x,y,0)
            #print(Z)
            if Z==Target and Mask.item(x,y)==1:
                F1.write("{} {} {}\n".format(x,y,Z))
    F1.close()

def searchAryData_L(url,outputpath,Target):
    Array=h5_node_reader(url, "/HistOcc")
    url1=url.split(os.sep)
    url1=url1[len(url1)-1].split("_")
    url1.pop(len(url1)-1)
    url1="_".join(url1)+".h5"
    Mask=h5_node_reader(url1, "/configuration/Enable")
    size=Array.shape
    F1=open(outputpath,"w")
    F1.write("X/L:Y/L:Z/L\n")
    F1.write("-1 -1 {}\n".format(size[0]))
    F1.write("-1 -1 {}\n".format(size[1]))
    for x in range(size[0]):
        for y in range(size[1]):
            Z=Array.item(x,y,0)
            #print(Z)
            if Z<=Target and Mask.item(x,y)==1:
                F1.write("{} {} {}\n".format(x,y,Z))
    F1.close()

def GetMaskData(url,outputpath,Target):
    Array=h5_node_reader(url,"/HistOcc")
    url1=url.split(os.sep)
    url1=url1[len(url1)-1].split("_")
    url1.pop(len(url1)-1)
```

```
    url1="_".join(url1)+".h5"
    Mask=h5_node_reader(url1, "/configuration/Enable")
    size=Array.shape
    F1=open(outputpath,"w")
    F1.write("X/L:Y/L:Z/L\n")
    F1.write("-1 -1 {}\n".format(size[0]))
    F1.write("-1 -1 {}\n".format(size[1]))
    for x in range(size[0]):
        for y in range(size[1]):
            Z=Array.item(x,y,0)
            #print(Z)
            if Mask.item(x,y)==Target:
                F1.write("{} {} {}\n".format(x,y,Z))
    F1.close()
def GetData(url,outputpath):
    Array=h5_node_reader(url, "/HistOcc")
    size=Array.shape
    F1=open(outputpath,"w")
    F1.write("X/L:Y/L:Z/L\n")
    F1.write("-1 -1 {}\n".format(size[0]))
    F1.write("-1 -1 {}\n".format(size[1]))
    for x in range(size[0]):
        for y in range(size[1]):
            Z=Array.item(x,y,0)
            #print(Z)
            F1.write("{} {} {}\n".format(x,y,Z))
    F1.close()
def datReader(Path):
    #open dat file
    f1=open(Path,"r")
    #skipping header
    f1.readline()
    #reading Size data fron file
    X=f1.readline()
    Y=f1.readline()
    #casting string data to int
    X=X.split("\n")[0]
    X=int(X.split(" ")[2])
    Y=Y.split("\n")[0]
    Y=int(Y.split(" ")[2])
    #generate Ary with size X,Y fild with Zeros
    Ary=np.zeros((X,Y))
    #input pixel location from dat file
```

```
    for line in f1:
        temp=line.split("\n")[0].split("␣")
        Ary.itemset((int(temp[0]),int(temp[1])),1)
    #closing file
    f1.close()
    #returning Mask like Ary
    return(Ary)
def ModuleCompare(inpath,Namelist,Outpath):
    #compare two Data sets and output the difference
    for x in range(4):
        ary1=datReader(inpath+"/module_{}".format(x)+Namelist[0])
        ary2=datReader(inpath+"/module_{}".format(x)+Namelist[1])
        Ary=ary1-ary2
        #generate ouputfile and writing header
        f1=open(inpath+"/module_{}".format(x)+Outpath,"w")
        f1.write("X/L:Y/L:Z/L\n")
        f1.write("-1␣-1␣{}\n".format(Ary.shape[0]))
        f1.write("-1␣-1␣{}\n".format(Ary.shape[1]))
        for X in range(Ary.shape[0]):
            for Y in range(Ary.shape[1]):
                if Ary.item(X,Y)>0:
                    f1.write("{}␣{}␣-1\n".format(X,Y))

        f1.close()
def ModuleCount(inpath,Name):
    #loop over all 4 chips
    count=0
    for x in range(4):
        f=open(inpath+"\module_{}".format(x)+Name)
        #counts pixels in File
        for line in f:
            count+=1
        #correction for Headers in datafiles
    count=count-(3*4)
    return(count)
```

Plotter.py

```
import Utils.IO as IO
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.colors import ListedColormap
```

```
def genQuadAryfromData(DATAList):
    Temp={}
    for x in DATAList:
        Mask=x[1].split("_interpreted.h5")
        Mask="".join(Mask)+".h5"
        Temp[int(x[0].split("_")[1])]=
        (IO.h5_node_reader(x[1], "/HistOcc"),
        IO.h5_node_reader(Mask, "/configuration/Enable"))
    size=Temp[0][0].shape
    size=(size[0]*2,size[1]*2)
    Ary=np.zeros(size)
    for z in Temp:
        Arytuple=Temp[z]
        for x in range(Arytuple[0].shape[0]):
            for y in range(Arytuple[0].shape[1]):
                if Arytuple[1].item(x,y)==1:
                    Z=Arytuple[0].item(x,y,0)
                else:
                    Z=0
                if z==0:
                    Ary.itemset((int(size[0]/2-1)-x,int(size[1]/2-1)-y),Z)
                if z==1:
                    Ary.itemset((int(size[0]/2-1)-x,int(size[1]-1)-y),Z)
                if z==2:
                    Ary.itemset((x+int(size[0]/2),y+int(size[1]/2)),Z)
                if z==3:
                    Ary.itemset((x+int(size[0]/2),y),Z)
    return(Ary)
def genQuadAryfromData2(DATAList):
    Temp=[]
    for x in DATAList:
        Temp.append(IO.datReader(x))
    size=Temp[0].shape
    size=(size[0]*2,size[1]*2)
    #print(size)
    Ary=np.zeros(size)
    i=0
    for z in Temp:
        Arytuple=z
        for x in range(Arytuple.shape[0]):
            for y in range(Arytuple.shape[1]):
                Z=Arytuple.item(x,y)
```

```
                    if i==0:
                        Ary.itemset((int(size[0]/2-1)-x,int(size[1]/2-1)-y),Z)
                    if i==1:
                        Ary.itemset((int(size[0]/2-1)-x,int(size[1]-1)-y),Z)
                    if i==2:
                        Ary.itemset((x+int(size[0]/2),y+int(size[1]/2)),Z)
                    if i==3:
                        Ary.itemset((x+int(size[0]/2),y),Z)
            i+=1
    return(Ary)
def plotAry(Ary,Outputpath):
    Max=np.amax(Ary)
    viridis = cm.get_cmap('viridis', int(Max))
    newcolors = viridis(np.linspace(0, 1, int(Max)))
    white = np.array([1, 1, 1, 1])
    newcolors[:1, :] = white
    newcmp = ListedColormap(newcolors)
    #F1.figimage(X=array,resize=True)
    plot=plt.imshow(X=Ary,aspect="auto", interpolation='none',cmap=newcmp)
    plt.colorbar(plot)
    plt.savefig(Outputpath)
    plt.close()
    print("Quadplot Done")
def plotAry2(Ary,Outputpath):
    Max=np.amax(Ary)
    white = np.array([1, 1, 1, 1])
    blue=np.array([0,0,1,1])
    newcmp = ListedColormap([white,blue])
    #F1.figimage(X=array,resize=True)
    plot=plt.imshow(X=Ary,aspect="auto", interpolation='none',cmap=newcmp)
    plt.colorbar(plot)
    plt.savefig(Outputpath)
    plt.close()
    print("Quadplot Done")
def plotQuadfromDataList(DATAList,Path):
    Ary=genQuadAryfromData(DATAList)
    plotAry(Ary, Path)
def plotbarM3(Outputpath,labels,List0,List1=[],List2=[],
Names=["Input1","Input2","Input3","Y-Axis","Title"]):
    x = np.arange(len(labels))  # the label locations
    width = 0.25  # the width of the bars
    fig, ax = plt.subplots()
    if len(List1)==len(List0) and len(List2)==len(List0):
```

```
        rects1 = ax.bar(x - width, List0, width, label=Names[0])
        rects2 = ax.bar(x , List1, width, label=Names[1])
        rects3 = ax.bar(x + width, List2, width, label=Names[2])
    else:
        if len(List1)==len(List0) and not len(List2)==len(List0):
            rects1 = ax.bar(x - width/2, List0, width, label=Names[0])
            rects2 = ax.bar(x + width/2, List1, width, label=Names[1])
        else:
            rects1 = ax.bar(x, List0, width, label=Names[0])

    # Add some text for labels, title and custom x-axis tick labels, etc.
    ax.set_ylabel(Names[3])
    ax.set_xlabel("Reconnection iteration")
    ax.set_title(Names[4])
    ax.set_xticks(x)
    ax.set_xticklabels(labels)
    ax.legend(loc="center right")
    def autolabel(rects):
        """Attach a text label above each bar
        in *rects*, displaying its height."""
        for rect in rects:
            height = rect.get_height()
            ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')
    if len(List1)==len(List0) and len(List2)==len(List0):
        autolabel(rects1)
        autolabel(rects2)
        autolabel(rects3)
    else:
        if len(List1)==len(List0) and not len(List2)==len(List0):
            autolabel(rects1)
            autolabel(rects2)
        else:
            autolabel(rects1)

    fig.tight_layout()
    plt.savefig(Outputpath)
    plt.close()

def plotIV(RNlist,DirList,Inpath,Outputpath):
```

```
    # Ploting Parameters
    fig = plt.gcf()
    fig.set_size_inches(18.5, 10.5)
    DN=-1
    plt.rcParams['xtick.labelsize']=24
    plt.rcParams['ytick.labelsize']=24
    #Section for Preassembly IV Data
    #X=GR("C:\git\OutTest\Glasgow.txt")
    #Y=X[1]
    #X=X[0]
    #plt.scatter(X,Y,s=100,alpha=0.8,marker="o",label="Bare Module")
    #IV Data Plotting for ALL Test Stepps
    for x in RNlist:
        DN=DN+1
        Temp=Inpath+"/{}/module_0/{}
_________module_0_iv_scan.h5".format(DirList[DN],x)
        #print(Temp)
        WH=IO.h5_node_reader(Temp, '/IV_data')
        #print(WH)
        X,Y=zip(*WH)
        X=np.array(X)
        #print(X)
        Y=np.array(Y)
        plt.scatter(X,Y*1000000000,s=100,alpha=0.8,marker="X",
        label=DirList[DN])
    #Setting Labels on Plot and Saving Picture
    plt.ylabel("Leakeage current [nA]",fontsize=24)
    plt.xlabel("Voltage",fontsize=24)
    fig.legend(loc="lower right",fontsize=24)
    fig.savefig(Outputpath)
    plt.close()
```

Config.py

```
def Config():
    config = {
        # Path of output directory, if no folder with specified name
          exits it will be generated
        "OutputDir": "E:/Output",
        # Path to input data folders
        "InputDir" : "E:/pybar",
        # Data Folders for Analysis
        "Data": {"Run0": {
```

```
        "Name"              : "G_R4"  ,
        "RN_Digital"        : 2  ,
        "RN_Analog"         : 3  ,
        "RN_Source"         : [24,16,18,21]  ,
        "RN_CONSource"      : 26  ,
        "RN_IV"             : 27
    },
    "Run1": {
        "Name"              : "G_R5"  ,
        "RN_Digital"        : 2  ,
        "RN_Analog"         : 3  ,
        "RN_Source"         : [21,10,12,16]  ,
        "RN_CONSource"      : 24  ,
        "RN_IV"             : 25
    },
    "Run2": {
        "Name"              : "G_R6"  ,
        "RN_Digital"        : 2  ,
        "RN_Analog"         : 3  ,
        "RN_Source"         : [12,10,16,14]  ,
        "RN_CONSource"      : 19  ,
        "RN_IV"             : 21
    },
    "Run3": {
        "Name"              : "G_R7"  ,
        "RN_Digital"        : 2  ,
        "RN_Analog"         : 3  ,
        "RN_Source"         : [9,10,11,12]  ,
        "RN_CONSource"      : 15  ,
        "RN_IV"             : 18
    },
    "Run4": {
        "Name"              : "G_R8"  ,
        "RN_Digital"        : 2  ,
        "RN_Analog"         : 3  ,
        "RN_Source"         : [9,10,11,12]  ,
        "RN_CONSource"      : 15  ,
        "RN_IV"             : 16
    },
    "Run5": {
        "Name"              : "G_R9"  ,
        "RN_Digital"        : 2  ,
        "RN_Analog"         : 3  ,
```

```
                "RN_Source"       : [9,10,11,12]  ,
                "RN_CONSource"    : 14,
                "RN_IV"           : 15
            },
            "Run6": {
                "Name"            : "G_R10"  ,
                "RN_Digital"      : 2  ,
                "RN_Analog"       : 4  ,
                "RN_Source"       : [10,12,13,14]  ,
                "RN_CONSource"    : 15  ,
                "RN_IV"           : 16
            }
        }
}
return(config)
```

B. Selbstständigkeitserklärung

Ich versichere, dass ich die schriftliche Ausarbeitung selbständig angefertigt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle Stellen, die dem Wortlaut oder dem Sinn nach (inkl. Übersetzungen) anderen Werken entnommen sind, habe ich in jedem einzelnen Fall unter genauer Angabe der Quelle (einschließlich des World Wide Web sowie anderer elektronischer Datensammlungen) deutlich als Entlehnung kenntlich gemacht. Dies gilt auch für angefügte Zeichnungen, bildliche Darstellungen, Skizzen und dergleichen. Ich nehme zur Kenntnis, dass die nachgewiesene Unterlassung der Herkunftsangabe als versuchte Täuschung gewertet wird.

Datum	Nico Malinowski