INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Masterarbeit

Eignungsfeststellung von Flash-Speichern im Automobilbereich

Daniel Haas

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Masterarbeit

Eignungsfeststellung von Flash-Speichern im Automobilbereich

Daniel Haas

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Nils gentschen Felde Jan Schmidt Pascal Jungblut Uwe Henkel (BMW)
Abgabetermin:	6. August 2018

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 4. August 2018

.....
(Unterschrift des Kandidaten)

Abstract

Die Komplexität der Elektronik in modernen Fahrzeugen wächst stetig an. Damit wachsen auch die Anforderungen der persistenten Datenspeicher in den diversen Steuergeräten an. Für die Auswahl des richtigen persistenten Speichers (meist NAND-Flash) stehen verschiedene Speicherstandards und folglich deren konkrete Implementierungen verschiedener Speicherhersteller zur Verfügung. Diese Speicher gilt es auf ihre Eignung für den Automobilbereich zu testen. Die Eignung wurde in dieser Arbeit als ein Zusammenschluss aus Performance, Effektivität und Funktionsangebot definiert. Für die Eignungsfeststellung wurde eine Testmethodik entwickelt, die auf der *Performance Test Specification* der *Storage Networking Industry Association* basiert und durch Anpassungen an den Automobilbereich ergänzt wurde. Anschließend wurde ein Software-Prototyp implementiert und diverse Testreihen an verschiedenen NAND-Flash-Speichern (*eMMC* und *UFS*) durchgeführt. Die Ergebnisse zeigten, dass sich Speicher des gleichen Standards und der gleichen Speicherkapazität enorm in ihrer Performance und Effektivität unterscheiden können. Dies ist unter anderem auf die unterschiedliche Strategie der Speicherfirmware zurückzuführen, jedoch auch auf andere Faktoren. Es hat sich also gezeigt, dass die Forschung auf diesem Gebiet durchaus notwendig ist und auch in Zukunft weiter ergänzt werden sollte, um die bestmögliche Applikation im Automobilbereich zu ermöglichen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADAS Advanced Driver Assistance Systems

HDD Hard Disk Drive

OEM Original Equipment Manufacturer

eMMC embedded Multimedia Card

UFS Universal Flash Storage

DRAM Dynamic Random Access Memory

SRAM Static Random Access Memory

ROM Read Only Memory

MOS Metal Oxide Semiconductor Transistor

CMOS Complementary Metal Oxide Semiconductor

XiP Execute in Place

MLC Multi Level Cell

SLC Single Level Cell

TLC Triple Level Cell

QLC Quad Level Cell

P/E Program and Erase

BiCS Bit Cost Scalable Technology

V-NAND Vertical NAND

CT Charge Trap

CHE Channel-Hot-Electron

FN Fowler-Nordheim

FTL Flash Translation Layer

SoC System-on-a-Chip

ECC Error Correcting Code

LPN Logical Page Number

LBN Logical Block Number

LSN Logical Sector Number

LBA Logical Block Adress

OOB Out-Of-Band

NOP Number of Partial Programming

ECN Erase Count Number

ECB Erase Count Block

FFS Flash File System

API Application Programming Interface

DBMS Datenbank-Management-System

SNIA Storage Networking Industry Association

SSS Solid State Storage

IOPS Input/Output Operations per Second

DUT Device Under Test

OIO Outstanding I/O Operations

FOB Fresh out of the box

RW Read-Write

BS Block Size

WAF Write Amplification Factor

JEDEC Joint Electron Device Engineering Council

OLTP Online Transaction Processing

PTS Performance Test Specification

ONFI Open NAND Flash Interface

DDR Double Data Rate

MIPI Mobile Industry Processor Interface

LU Logical Unit

OCR Operation Conditions Register

CID Device Identification Register

CSD Device Specific Data Register

EXTCSD Extended CSD Register

RCA Relative Device Adress Register

DSR Driver Stage Register

QSR Queue Status Register

RPMB Replay Protected Memory Block

SCSI Small Computer System Interface

WSAT Write Saturation Test

SS Steady State

UML Unified Modeling Language

XML Extensible Markup Language

JSON JavaScript Object Notation

GUI Graphical User Interface

CLI Command Line Interface

1. Einleitung

1.1. Ausgangssituation

Der zunehmende informationstechnische Fortschritt im Kraftfahrzeug erleichtert das Leben des Fahrers in vielen Belangen. Die stetige Weiterentwicklung im Bereich Entertainment bringt immer leistungsfähigere Systeme hervor (z. B.Navigation, Sprachassistenz) oder ermöglichen es, den Fond-Passagieren im Fahrzeug beispielsweise einen Film in Blu-Ray-Qualität anzusehen, während sie sich auf dem Weg zu ihrem Ziel befinden. Auch ein großes Thema der Zukunft im Automobilbereich sind die *Advanced Driver Assistance Systems (ADAS)*, die im Deutschen mit „Fahrassistenzsystem“ übersetzt werden können. Dies reicht von kleinen Erleichterungen, wie einem Park-Lenk-Assistenten bis hin zum vollen autonomen Fahren, bei dem das Fahrzeug den Verkehr vollkommen autonom bewältigen kann.

All diese neu entstehenden Funktionen haben eines gemeinsam: Sie werden von einer Vielzahl an zusehend leistungsfähigeren und vernetzten Steuergeräten getrieben. Abbildung 1.1 zeigt, dass die Anzahl der Funktionen pro Fahrzeug auch in Zukunft stark ansteigen wird. Interessant ist, dass sich der Trend zu einer schwächer ansteigenden Anzahl an Steuergeräten jedoch mit mehr Funktionen pro Steuergerät entwickelt. Hier sollen in Zukunft Kosten eingespart werden. Dazu werden bauraumorientiert kleinere funktionsärmere Steuergeräte zu größeren funktionsreicheren Steuergeräten zusammengefasst [SZ16].

Eine logische Konsequenz ist, dass auch die Speichermedien dieser Steuergeräte immer größere Kapazitäten und höhere Leistungsfähigkeit benötigen. Neben Festplatten (*Hard Disk Drive (HDD)*) sind NAND-basierte Flash-Speicher im automotiven Umfeld als Speichermedium sehr verbreitet. Sie können dabei als eingebettete Chips oder als herausnehmbare Speicherkarten implementiert werden [Cou16].

Flash-basierte NAND-Speicher finden ihre Verwendung neben dem Automobilbereich auch im Industrie- und im Konsumbereich, z. B. als Speichermedium in einem Smartphone. Standardisiert werden diese von der Organisation JEDEC Solid State Technology Association. Der JEDEC-Ausschuss entwickelt offene Standards für Halbleiterspeicher. Diese Standards dienen dazu ein Produkt von der Innovation zur Massenware zu transformieren, während Qualität und Zuverlässigkeit aufrecht erhalten werden. Dies führt auf der einen Seite dazu, dass Unternehmen stärker versuchen an innovativen Technologien zu arbeiten und es ermöglicht auf der anderen Seite ein variableres Angebot für den Käufer. Anstatt immer von Grund auf neu mit dem Entwicklungsprozess eines Produktes zu beginnen, können Speicherhersteller ihr Design auf Standards aufbauen und sich gezielter auf neue Innovationen fokussieren [1].

Solange nicht alle Vorgaben und Spezifikationen eines gewissen Standards erfüllt werden, darf sich ein Produkt nicht JEDEC-konform nennen [JED15]. Daraus resultierend sind diese Standards für Speichertechnologien die Ausgangsbasis dieser Arbeit. In den Standards befinden sich alle wichtigen Vorgaben und vom Speicher zu liefernde Informationen, die schließlich

[1] Why JEDEC Standards Matter - https://www.jedec.org/standards-documents/about-jedec-standards Stand 15.01.2018

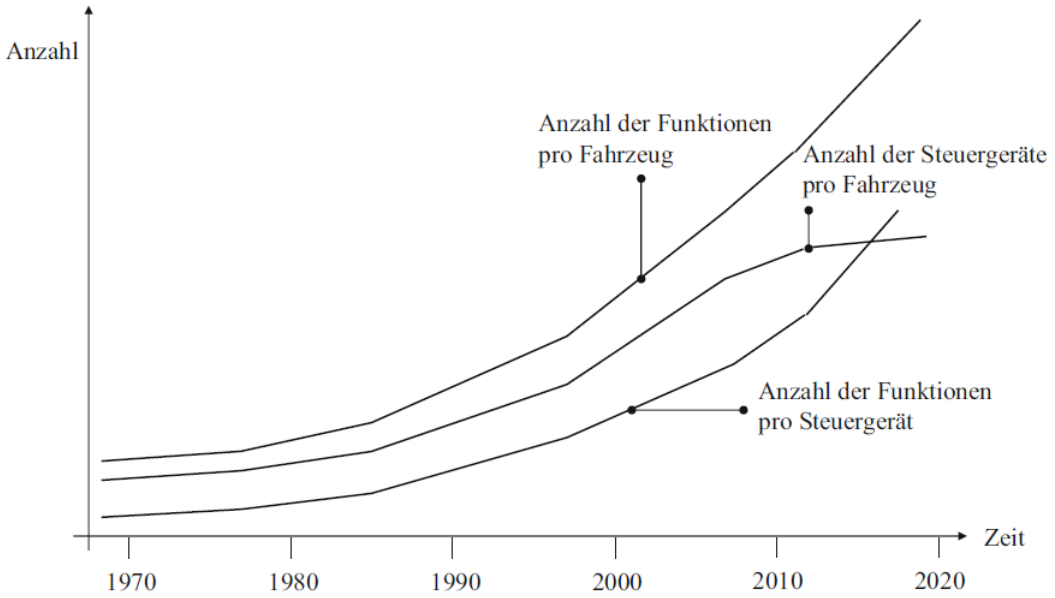


Abbildung 1.1.: Funktionen und Steuergeräte pro Fahrzeug [SZ16]

vom Speicherhersteller komplett implementiert werden müssen. Da der JEDEC-Standard oft verschiedene Varianten eines Speichers enthält (z. B. verschiedene Übertragungsgeschwindigkeiten) und die Hersteller diese und zusätzliche Informationen in ihren Datenblättern preisgeben, gehören auch die Datenblätter zur Ausgangsbasis.

1.2. Zielsetzung

Bei Veröffentlichung eines neuen Speicherstandard ist zu prüfen, ob die entwickelten Produkte auch für den Automotive-Bereich geeignet sind, da der JEDEC-Standard kein spezieller Automotive-Standard ist. Ziel der Arbeit ist folglich die Entwicklung einer Methodik und eines Softwarekonzeptes zur Feststellung der Automotive-Eignung neuer Flash-Speichertechnologien. Da es keinen herstellerübergreifenden Automotive-Standard gibt, der alle Anforderungsbereiche eines Datenspeichers definiert, muss die Situation differenzierter betrachtet werden. Für einen speziellen Automobilhersteller, auch oft *Original Equipment Manufacturer (OEM)* genannt, gelten eigene Standards (bei BMW: Group Standards). Diese befassen sich hauptsächlich mit den Fertigungs-, Qualitätssicherungs- und Qualifizierungsprozessen. Diese werden im Rahmen dieser Arbeit nicht behandelt, da einzelne Tests keine statistisch aussagekräftigen Hinweise über die Qualität eines Produktes liefern.

Bei den Funktionalitätsanforderungen verweisen die Standards darauf, dass die im Lastenheft spezifizierten Anforderungen erfüllt werden müssen. In einem Kraftfahrzeug gibt es, wie schon erwähnt, verschiedene Bereiche, in denen Flash-Speicher verwendet werden. Dazu zählen das Infotainmentsystem[2] oder das ADAS. Diese Bereiche haben zum Teil sehr unterschiedliche Anforderungen (z. B. Temperatur, Performance). Definiert sind diese für einen Bereich in einem speicherspezifischen Lastenheft, welches konkrete Grenzwerte zu verschiede-

[2]Zusammenschluss aus Funktionen wie Autoradio, Navigationssystem, Freisprecheinrichtung und weitere Funktionen zu einer zentralen Bedieneinheit

nen Funktionalitäten beinhaltet. Was folglich untersucht werden kann, sind die Eigenschaften (Funktionalitäten) eines Speichers unter verschiedenen Bedingungen. Um nun die Eignung für den Automobilbereich festzustellen, müssen die Eigenschaften eines Speichers für die verschiedenen Lastenhefte eines Automobils validiert werden. Als Ergebnis kann sich somit die Eignung für verschiedene oder für alle Bereiche ergeben. Der Fokus liegt hierbei auf der Erfüllung gewisser Leistungsanforderungen und nicht der Qualitätssicherung. Nicht außer Acht gelassen werden darf, dass diese Eignung für jeden Automobilhersteller unterschiedlich sein kann und somit für jeden Einzelfall separat festgestellt werden muss.

1.3. Methodik und Vorgehen

Um eine auf NAND-Flash-Speicher zugeschnittene Testmethode zu entwickeln, sollen in Kapitel 2 die nötigen theoretischen Grundlagen dafür geschaffen werden. Aus jedem Grundlagenunterkapitel soll hervorgehen, wie sich der jeweilige Aspekt auf die Methode auswirken kann. Zuerst wird eine Klassifikation von Halbleiterspeichern vorgenommen, damit der Einsatzbereich des NAND-Speichers besser eingeordnet werden kann. Anschließend werden die wichtigsten technischen Grundlagen zur Flash-Funktionsweise erläutert. Daraus ergibt sich der entscheidende NAND-Performance-Charakter. Darauf folgend sollen die verschiedenen Problemfelder der Zuverlässigkeit dieses Speichertyps und die Auswirkungen auf die Performance untersucht werden. Ein kurzer Einblick in die 3-D-Architektur soll die starken Unterschiede zwischen Speichern des gleichen Standards zeigen und folglich die Notwendigkeit vergleichbare Tests mit verschiedenen Speicher-Technologien durchzuführen. Im nächsten Schritt erfolgt die Einordnung des Flash-Speicherchips in sein systemisches Umfeld, um zu erkennen in welchem Maße dieses Einfluss auf die Funktionalität des Speichers besitzt. Schließlich werden die beiden Flash-Standards embedded Multimedia Card (eMMC) und Universal Flash Storage (UFS) auf ihre Speicherschnittstellen und Kommunikationsmodelle untersucht. Hier können grundlegende Unterschiede in der Konzeption erkannt werden, die auf die Leistung des Speichermediums essenziellen Einfluss haben können.

Kapitel 3 zeigt themenverwandte Arbeiten auf. Eine Performance-Testspezifikation der Storage Networking Industry Association (SNIA) wird als Basis der zu erstellenden Testmethode deklariert. Die Spezifikation beinhaltet eine komplette Testmethode für SSD-Speicher. Damit jedoch eingebettete Speicher in kleineren Speichergrößen und mit einem anderen Anforderungsprofil methodisch korrekt getestet werden können, werden dieser Testspezifikation noch Anpassungen auf das automotive Umfeld hinzugefügt. Die Konzepte dafür sollen aus ergänzenden Quellen erarbeitet werden. Schließlich werden noch weitere verwandte Themen gezeigt, die keinen zusätzlichen Aspekte für die eigene Testmethodik enthalten. So können beispielsweise Fehler in deren Herangehensweise analysiert werden und somit für die eigene Methode ausgeschlossen werden.

In Kapitel 4 wird das vollständige Testkonzept erarbeitet. Zuerst werden dazu die Informationen, die die beiden Flash-Standards eMMC und UFS zur Verfügung stellen, kategorisiert. Anhand dieser Kategorien wird dann gezeigt, welche Eigenschaften die beiden Standards für die Diagnose und Konfiguration zur Verfügung stellen. Des weiteren werden die bereits erarbeiteten einzelnen relevanten Komponenten der Performance-Testspezifikation der SNIA wiederholt, da diese Basis der eigenen Methode ist. Anschließend werden die konkreten Anpassungen an das automotive Umfeld wiederholt und zusätzliche Tests und Testparameter konzipiert und definiert. In einer darauffolgenden Zusammenfassung werden dann der

endgültige Katalog an Messgrößen für beide Speicherstandards und die fertige Messmethode mit konkreten Tests, sowie deren Input und Output festgelegt.

Die anschließende Implementierung eines Prototypen soll in Kapitel 5 beschrieben werden. Erster Schritt hierfür ist die Erstellung eines Datenformates für den Input des Prototypen. Dazu soll ein Standardformat festgelegt werden, das alle Optionen und Metadaten bereitstellt, um ein konkretes Lastenheft darauf umzusetzen. Im Anschluss sollen die Sollzustände für BMW festgelegt werden. Diese ergeben sich aus den konkreten Grenzwerten eines beispielhaften Speicherlastenheftes. Im nächsten Schritt wird die Implementierung der Testsoftware für die in Kapitel 5 konzipierten Messreihen erläutert. Dazu werden die Entwicklungsumgebung bestehend aus Hard- und Softwareumgebung, sowie die entwickelte Softwarearchitektur beleuchtet. Schließlich werden noch die wichtigsten Implementierungsschritte für die einzelnen Testkomponenten aufgezeigt.

In Kapitel 6 soll die konkrete Durchführung einer kompletten Messreihe für einen UFS-Speicher gezeigt werden. Zunächst wird dafür die konkrete verwendete Input-Datei erläutert. Darauf folgen die notwendigen Schritte, die für die Durchführung einer Testreihe notwendig sind. Anschließend wird noch die Ordnerstruktur des Ergebnisordners beschrieben, der vom Prototypen als Resultat eines Testdurchlaufs erstellt wird.

Schließlich soll in Kapitel 7 eine Ergebnisinterpretation stattfinden. Es werden die Testergebnisse für das BMW-Fallbeispiel und einen Benchmark verschiedener eMMC- und UFS-Speicher analysiert. Beim BMW-Fallbeispiel wird erläutert, warum die abgeprüften Anforderungswerte erfüllt oder nicht erfüllt wurden und was dies an Schlussfolgerungen über die verwendete Testmethode zulässt. Für die Benchmark-Interpretation sollen die Diagnose, alle Einzeltests, sowie der Temperatureinfluss separat diskutiert werden. Erst sollen Auffälligkeiten gefunden und anschließend versucht werden, diese durch das erlangte Wissen über Flash-Speicher zu begründen. In einer anschließenden Zusammenfassung werden die Haupterkenntnisse der Testreihen zusammengefasst und es findet eine Bewertung der eigenen Testmethode statt. Abschließend folgt ein Ausblick auf zukünftige Weiterentwicklungsmöglichkeiten der Testmethode und der daraus resultierende Mehrwert für die Entwicklung von Applikationen im automotiven Umfeld.

2. Grundlagen Flash

In diesem Kapitel werden die Grundlagen zu Flash-Speichern erläutert. Aus jedem Unterkapitel geht hervor, welchen Einfluss das jeweilige Thema auf die Performance oder auf die Messmethode hat. Zudem soll das Kapitel dabei helfen, die später im Testprozess erzeugten Ergebnisse auch richtig interpretieren zu können. Anfangs sollen in den Unterkapiteln 2.1 bis 2.2 die physikalischen Eigenschaften des NAND-Flash-Speichers ohne Berücksichtigung des Umfelds (z.B. Controller, Schnittstelle, usw.) analysiert werden. Dadurch soll ein grundlegendes Verständnis für die speziell Flash betreffenden Performancecharakteristika erworben werden. Zu Beginn soll eine grobe Klassifikation von Halbleiterspeichern die Einordnung der Flash-Speicher ermöglichen und ihr Einsatzgebiet zeigen und eingrenzen. Im nächsten Schritts soll der grundsätzliche Aufbau und Funktionsweise einer Flash-Zelle aufgezeigt werden, wobei auf die zwei Flash-Typen NOR und NAND eingegangen wird. Weiter soll das Thema Zuverlässigkeit und mögliche Auswirkungen auf die Performance analysiert werden. Darauf folgt ein tieferer Einblick in den technologischen Fortschritt der NAND-Speichertechnologie. Es handelt sich dabei um einen Wandel von der 2D- zu der 3D-Flash-Architektur. Dem Leser soll ein Eindruck vermittelt werden, welche Hürden bereits zu welchem Preis überwunden wurden und welche Stärken und Schwächen daraus resultieren. In Unterkapitel 2.5 soll das systemische Umfeld im Hinblick auf die Performance und grundlegende Performance-Einflüsse auf den Speicher analysiert werden, die sich durch sein eingebettetes Umfeld und die beteiligten Systemkomponenten ergeben. Eine Analyse der Flash-Standards (eMMC und UFS) auf ihre Speicherschnittstellen und ihre Kommunikationsmodelle soll in Unterkapitel 2.6 grundlegende Unterschiede in der Konzeption zeigen. Daraus können wiederum Schlüsse über die Performance der Speicher gezogen werden.

2.1. Klassifikation von Halbleiterspeichern

Dieses Unterkapitel soll aufzeigen, welche Arten von Speichermedien grundsätzlich vorhanden sind, wo der NAND-Flash Speicher dabei einzuordnen ist und welches Einsatzgebiet für diese Speicher daraus resultiert. Im Allgemeinen lassen sich Speichermedien in zwei Klassen aufteilen:

- Flüchtige und
- Nichtflüchtige Speicher

Auf Abbildung 2.1 ist eine grobe Übersicht über die etablierten Speichertypen zu sehen.

2.1.1. Flüchtige Speicher

Flüchtige Speicher können ihre Information nur höchstens so lange halten, wie sie mit Spannung versorgt werden (teiweise muss Information sogar kontinuierlich aufgefrischt werden).

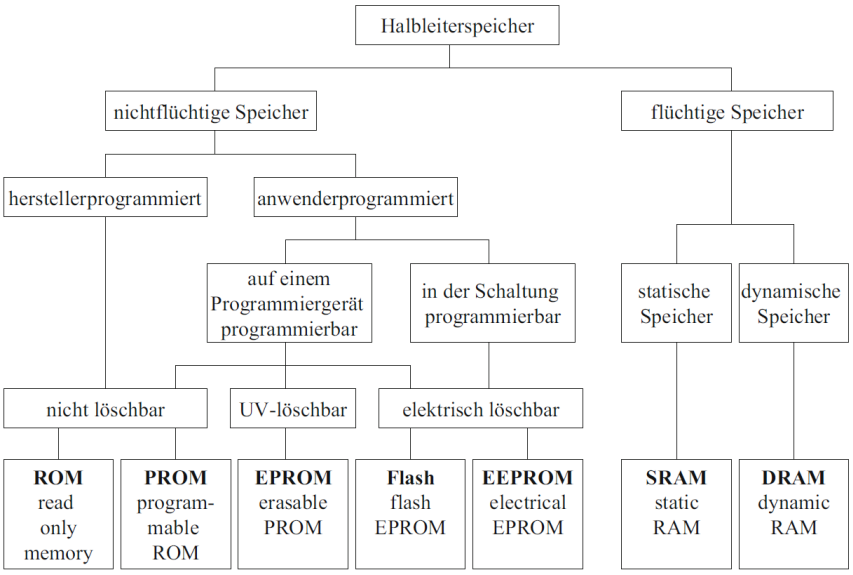


Abbildung 2.1.: Übersicht von Speichertechnologien [SZ16]

Wird die Spannungsversorgung entfernt, so verliert der Speicher allmählich seine Informationen. Diese Eigenschaft qualifiziert diese Speicherklasse folglich für spezielle Anwendungsbereiche. So werden diese als Cache (Puffer-Speicher) oder als Arbeitsspeicher verwendet [ARI16].

Eine weiteres Unterscheidungskriterium bietet die Einteilung in statische und dynamische Speicher. Der *Dynamic Random Access Memory (DRAM)*, der Vertreter der dynamischen Speicher muss seine Informationen periodisch auffrischen, um die Datenintegrität aufrecht zu erhalten. Dies stellt einen dynamischen Vorgang dar, auf welchen die Namensbebung zurückzuführen ist. Der *Static Random Access Memory (SRAM)* hingegen muss bei konstanter Spannungsversorgung seine Informationen nicht auffrischen, um seine Datenintegrität zu garantieren. Auf Abbildung 2.2 ist der Schaltungsaufbau der beiden Speicherzellen gezeigt. Zu erkennen ist hierbei der Unterschied in der Komplexität. Die DRAM-Zelle auf der linken Seite besteht aus nur einem Transistor und einem Kondensator. Die SRAM-Zelle auf der rechten Seite benötigt sechs Transistoren und zwei Inverter. Aufgrund der komplexeren Struktur ergibt sich für den SRAM der Vorteil, einfach steuerbar zu sein und er ist vergleichsweise schneller, da er keinen extra Datenbus benötigt, wie es bei DRAM der Fall ist. Auf der anderen Seite ist DRAM deutlich günstiger, was auf die niedrige Anzahl an Bauteilen und die höhere Energieeffiziez zurückzuführen ist. Aufgrund des einfachen Aufbaus ist es beim DRAM möglich eine viel höhere Speicherdichte verglichen mit SRAM zu erzielen. Deshalb dominiert DRAM auch den Markt der flüchtigen Speicher. Jedoch befindet sich fast in jedem Logik- oder Speicherchip SRAM in Form des erwähnten Pufferspeichers oder in eingebetteten Systemen, in denen nur bis zu zehntausenden an Kilobytes benötigt werden. DRAM-Speicher werden immer dort eingesetzt, wo große Datenmengen verarbeitet werden. Ein Beispiel wäre der Arbeitsspeicher für Desktops, da hier eine sehr große Speichermenge erforderlich ist [MSCT14].

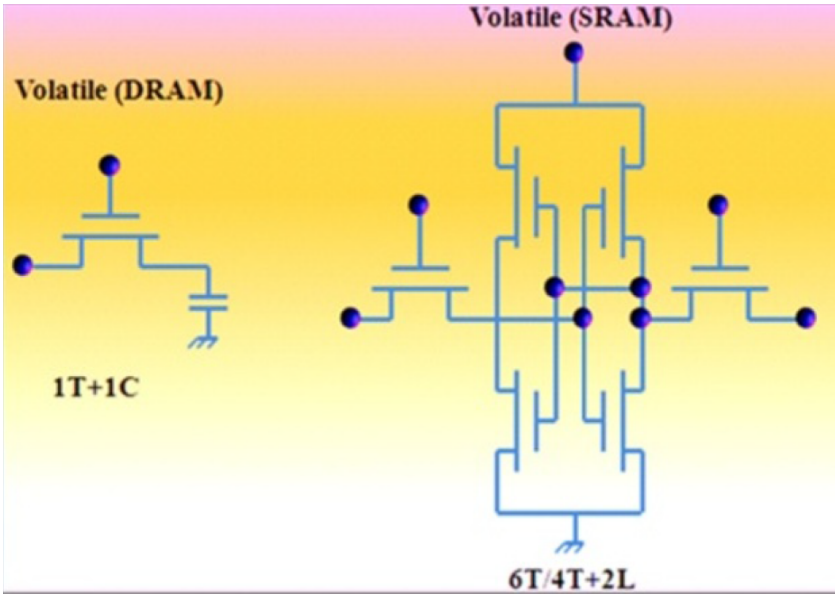


Abbildung 2.2.: Schaltungsaufbau von DRAM und SRAM [MSCT14]

2.1.2. Nichtflüchtige Speicher

Nichtflüchtige Speicher behalten ihre Information auch wenn sie von der Spannungsversorgung getrennt werden [MSCT14]. Daraus resultieren zwei verschiedene Anwendungsfälle. Speicher mit einer hohen Speicherdichte, d.h möglichst viel Speicherplatz auf geringer Platinenfläche und mit großem Schreibdurchsatz werden als Daten- oder Massenspeichermedium verwendet, wie beispielsweise eine SSD-Festplatte oder ein Flash-Speicherstick. Das zweite Anwendungsgebiet finden Speicher, die einen schnellen wahlfreien Zugriff benötigen und Code auf dem Speicher ausführen können. Das bedeutet, dass ein schneller Zugriff auf Speicherareale an verschiedenen Speicheradressen gegeben ist. Der schnelle Zugriff auf kleine Speicherbereiche wird durch eine stark granulare Adressierung erreicht. Diese Speicher können beispielsweise das Betriebssystem eines Mikrocontrollers speichern, wobei das Betriebsystem direkt auf dem Speicher ausgeführt werden kann [BCMV03]. Ein Beispiel hierfür wäre ein NOR-Flash-Speicher, der später noch näher erklärt wird.

Wie auf Abbildung 2.1 zu sehen ist, lassen sich auch nichtflüchtige Speicher weiter unterteilen. Die wichtigsten Unterscheidungskriterien sind hier auf der einen Seite wie ein Speicher programmiert wird und auf der anderen Seite *wie* und *ob* er gelöscht werden kann. Es gibt hersteller- und anwenderprogrammierte Speicher, die wiederum entweder nicht löschbar, durch ultraviolettes Licht löschbar oder elektrisch löschbar sind. Ein wichtiger Vertreter der herstellerprogrammierten Speicher ist der *Read Only Memory (ROM)*, der nicht löschbar ist. Dieser eignete sich in der Vergangenheit beispielsweise, um das BIOS eines Computers zu speichern, da es nicht verändert werden darf und nur Lesezugriff bieten soll. Zum heutigen Zeitpunkt wurde jedoch der ROM durch Flash-Speicher ersetzt, da das BIOS sich nun auch gelegentlichen Updates unterziehen muss. Für nichtflüchtige Speicher ergeben sich wichtige allgemeine Anforderungen, die zur Beurteilung eines Speichers dienen [BCMV03][ARI16]:

- Niedrige Bit-Kosten

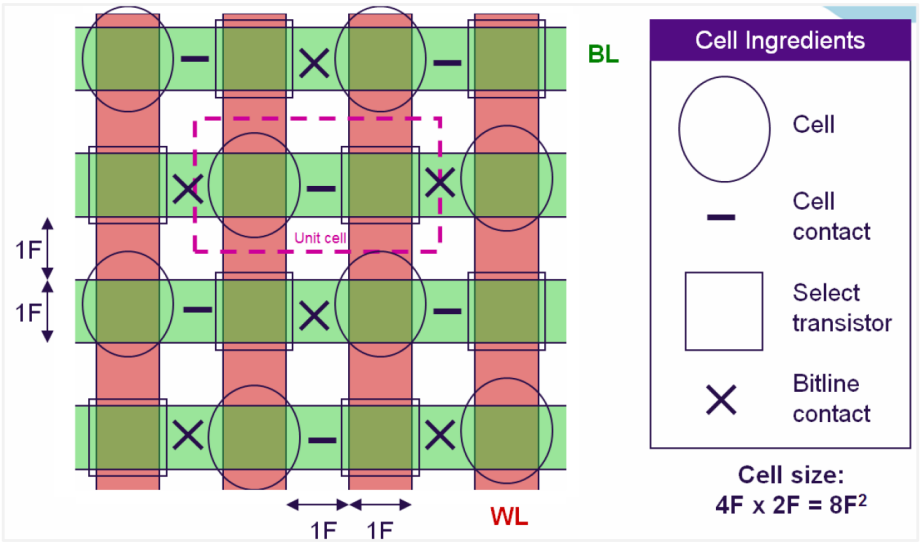


Abbildung 2.3.: Array-Layout mit einer $8F^2$ Strukturgröße

- Hohe Flexibilität
- Programmierung mit möglichst hoher Geschwindigkeit
- Hohe Zuverlässigkeit

Die Anforderung nach niedrigen Kosten ist die wichtigste. Der Preis wird hauptsächlich durch die Die-Size (Größe eines einzelnen, ungehäußten Halbleiterchips) des Speichers und die Kosten des Herstellungsprozesses bestimmt. Daher ist es notwendig eine möglichst kleine Die-Size mit einem möglichst kostengünstigen Fertigungsprozess zu kombinieren. Um die Größe des einzelnen Chips zu reduzieren, muss die Größe der einzelnen Speicherzelle verringert werden, da die Speicherzellen einen Großteil der benötigten Fläche ausmachen. Der Begriff der sogannten „Feature-Size“ ist für dieses Thema wichtig und wird auch im Verlauf dieser Arbeit öfter erwähnt, wenn es um das Thema Skalierung einer Speicherzelle geht. Sie ist eine essentielle Größe in der Halbleitertechnik und schafft Vergleichbarkeit der verschiedenen Technologien. Die Feature-Size ist die kleinste Strukturbreite, die zuverlässig fotolithographisch hergestellt werden kann. Dabei entspricht diese kleinste Strukturbreite einem „F“. „Features“ sind dabei beispielsweise die Flash-Einzelzelle, ein Zellkontakt, ein Auswahltransistor oder ein Bitline-Kontakt und nehmen jeweils die Strukturbreite eines Fs in Anspruch vgl. Abbildung 2.3. Das Ziel ist es eine $4F^2$ Struktur (F steht für Feature), bei der die planare Ausrichtung (nur in der Ebene) von Auswahltransistor und Zelle den jeweils geringsten Platzbedarf aufweist und damit die maximal mögliche Dichte erreicht wird. In den Anfängen der Flash-Technologie war dies jedoch noch nicht möglich. Auf Abbildung 2.3 ist ein solches Array-Layout mit einer $8F^2$ Strukturgröße abgebildet. Die Fläche wird hier durch die Ausdehnung und Anordnung der Einzelkomponenten innerhalb einer Einzelzelle bestimmt. Die Anforderungen für die Isolierung sind bei NAND-Flash höher als bei anderen Speichern, da diese mit einer vergleichsweise hohen Spannung konfrontiert werden. Da es durch die hohe Spannung zu Schäden an der Isolierung kommen könnte, gestaltete sich das Verringern des Durchmesser der Isolierungschicht äußerst schwierig [ARI16].

Mit Flexibilität ist die Fähigkeit gemeint, wiederholt gelöscht sowie wieder neu programmiert zu werden und dies mit einer minimalen Granularität. Dies bedeutet der Speicher muss auch in der Lage sein, kleine Speicherbereiche löschen und programmieren zu können. Die Anforderung an eine möglichst schnelle Programmierung, die in einer maximalen Schreibgeschwindigkeit des Speichers resultiert ist selbsterklärend [BCMV03].

Die Zuverlässigkeit ist auch eine essenzielle Anforderung an einen nichtflüchtigen Speicher. Dies bedeutet, dass der Speicher die Information lange genug und mit der verlangten Datenintegrität aufbewahrt. Genauere Informationen zum Thema Zuverlässigkeit folgen im Unterkapitel 2.3, nachdem der Aufbau der Flash-Zelle erläutert wurde.

Die Flash-Technologie bot dabei den besten Kompromiss zwischen den genannten Anforderungen, aufgrund einer Ein-Transistor-Zelle die öfter als 100000 mal elektrisch beschrieben werden konnte. Dabei konnte man in der Einheit Byte den Speicher beschreiben und in Pages (KiloBytes) löschen [BCMV03]. Folglich wurde die Flash-Technologie sehr stark in den folgenden Forschungsfeldern weiterentwickelt [MSCT14]:

- Verkleinerung der Zellgröße
- Verringerung des Stromverbrauchs
- Vergrößerung der Speicherdichte (Mehr Bits auf der selben Fläche)

2.2. Flash-Technologie

2.2.1. Aufbau einer Flash-Zelle

Aus Punkt 2.1.2 geht hervor, dass die Flash-Technologie eine enorm wichtige Technologie ist und ein sich stetig weiterentwickelndes Forschungsfeld. Deshalb wird sie als Exempel für nichtflüchtige Speicher in dieser Arbeit weiter betrachtet. Um die Vor- und Nachteile dieser Technologie zu verstehen, werden im Folgenden die technischen Grundprinzipien erläutert. Auf Abbildung 2.4 ist die Architektur einer einzelnen Flash-Zelle abgebildet und auf Abbildung 2.5 das daraus resultierende Schaltbild. Im Prinzip ist eine solche Zelle ein *Metal Oxide Semiconductor Transistor (MOS)*, also ein Halbleiter-Bauelement zum Steuern elektrischer Spannungen. Ein derartiger Transistor ist identisch zur gezeigten Flash-Zelle auf Abbildung 2.4. Bei der Flash-Zelle jedoch wird ein zusätzliches Gate, das Floating Gate (FG), in das Zentrum des Transistors eingefügt. Dieses ist elektrisch komplett mit Dielektrikum isoliert und „schwebt (floating)“ deswegen isoliert im Zentrum des Transistors. Aus diesem Grund ist es in der Lage Ladung zu speichern, ohne diese wieder zu verlieren. Das bedeutet aus einem reinen Steuerungselement wurde durch das Hinzufügen eines isolierten FGs eine Speicherzelle. Angesteuert (ausgewählt) wird die Speicherzelle über das Control Gate (CG) [BCMV03].

Zur Injektion und Emission von Ladung in das FG muss das sogenannte Tunneloxid, das das FG vom Rest des Transistors isoliert, überwunden und Ladung durch diese Isolationsschicht transportiert werden. Dafür gibt es generell zwei bekannte physikalische Verfahren [ARI16]:

- *Fowler-Nordheim (FN)*-Tunneln für Injektion und Emission
- *Channel-Hot-Electron (CHE)* Injection

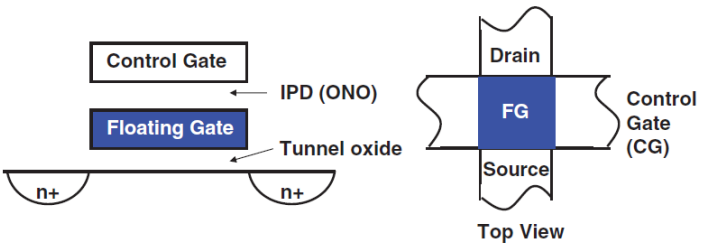


Abbildung 2.4.: Architekur einer Flash-Einzelzelle mit Floating Gate [ARI16]

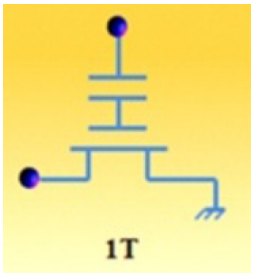


Abbildung 2.5.: Schaltbild einer Flash-Speicherzelle [MSCT14]

Wie sich diese Effekte genau erklären, soll kein Bestandteil dieser Arbeit sein. Grundsätzlich wird beim Fowler-Nordheim-Tunneln wird die Durchbruchsspannung der Isolierungsschicht herbeigeführt. Bei dieser Spannung wird also der Isolator leitend und es können Elektronen in das FG fließen. Der Effekt kann mit der Entstehung eines Blitzes verglichen werden, bei der Wolken bei einer gewissen Durchbruchsspannung in einem Blitz entladen werden. Bei der CHE ist nicht die Energie der Spannung ausschlaggebend. Die Elektronen, die ins FG injiziert werden sollen, werden so stark beschleunigt, dass die kinetische Energie der Ladung (Bewegungsenergie) ausreicht, die Isolationsschicht zu überwinden. Bei beiden Effekten kommt es zu Kollisionen von Elektronen mit dem Kristallgitter des Tunneloxids. Diese Tatsache führt zu einer allmählichen Abnutzung des Speichers. Ein wichtiger Faktor ist deshalb die Dicke des Tunneloxids. Dieses muss auf der einen Seite dünn genug sein, um die erwähnten Effekte möglich zu machen, auf der anderen Seite aber auch dick genug sein, um sich nicht zu schnell abzunutzen. Genaueres zur Zuverlässigkeit und somit auch den Abnutzungserscheinungen der Flash-Zelle wird in Unterkapitel 2.3 erläutert.

Zusammenfassend lässt sich feststellen, dass der grundlegende Aufbau der Flash-Zelle mit seiner Hochspannungsprogrammierung zu einer allmählichen Abnutzung führt, im Gegensatz zu anderen Speichern. Auf diese Abnutzung muss, wie später noch erklärt wird, reagiert werden. Der Speicher muss also auf gewisse Weise *schonend* betrieben werden, was wiederum einen essentiellen Einfluss auf die Performance bedeutet.

2.2.2. Architekturen

Nachdem die Grundlagen der einzelnen Flash-Zelle dargelegt wurden, ist es interessant sich die aus den Einzelzellen zusammengeschlossenen Speicher-Arrays anzusehen. Dabei werden zwei Zugriffstypen unterschieden. Es kann entweder parallel oder seriell auf das Speicherarray zugegriffen werden. Auf Abbildung 2.6 können diese beiden Hauptkategorien erkannt werden.

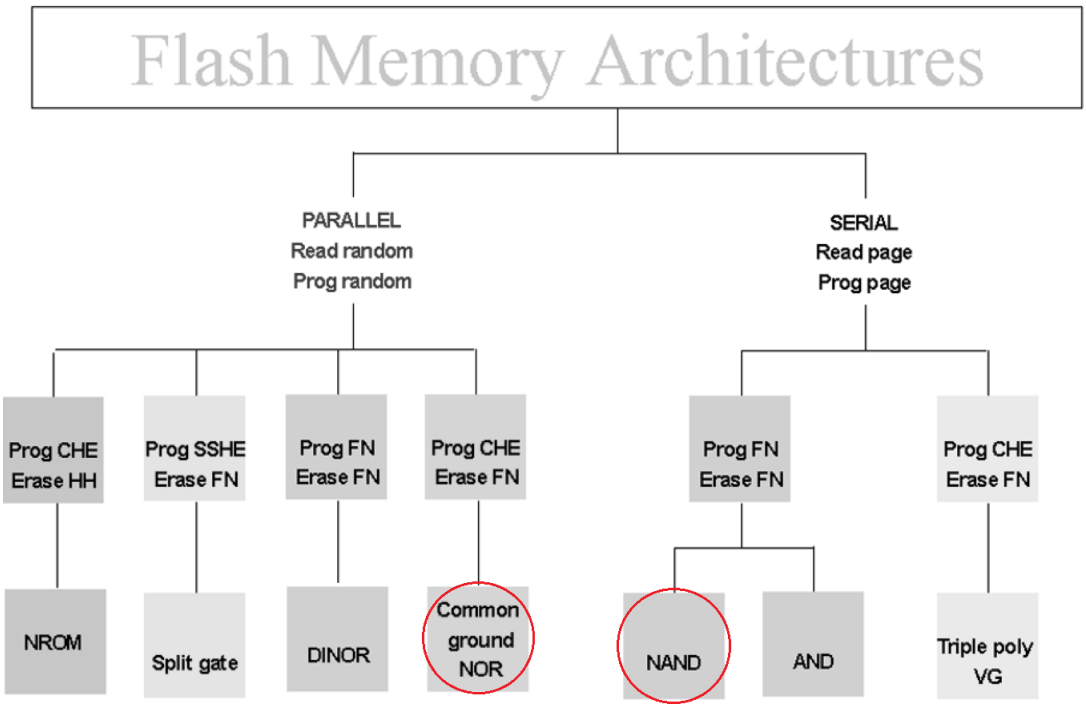


Abbildung 2.6.: Flash Architekturen [BCMV03]

Die zwei Vertreter sind der NOR-Flash-Speicher beim parallelen Zugriff und der NAND-Flash-Speicher beim seriellen Zugriff [BCMV03].

Die Namensgebung bezieht sich wiederum auf die Zugriffsstrukturen der beiden Speicher und kommt aus der *Complementary Metal Oxide Semiconductor (CMOS)*-Technik. Die CMOS-Technik ist die Logikfamilie, die hauptsächlich für integrierte Schaltkreise verwendet wird, d.h die Struktur für die elementaren Logikgatter (z.B. AND-Gatter) stellt. Auf Abbildung 2.7 sind die beiden Architekturen von NOR und NAND Speicherarrays zu sehen. Die parallele Verbindung des NOR-Arrays erinnert an die Verbindung der Transistoren in einem logischen NOR-Gate aus der CMOS-Technik. Die serielle Verbindung des NAND-Arrays erinnert wiederum an ein logisches NAND-Gate. Intel ist das Unternehmen, das 1988 den ersten NOR-Speicherchip auf den Markt gebracht hatte. Toshiba war der Vorreiter in Sachen NAND und brachte den ersten auf dieser Technologie basierten Speicherchip 1989 auf den Markt [MSCT14].

Die Vor- und Nachteile der beiden Technologien können auch von ihren jeweiligen Zugriffstypen abgeleitet werden. Da die Zellen des NOR parallel mit der Bitline (siehe Abbildung 2.7) verbunden sind, können diese Zellen individuell gelesen und programmiert werden. NOR bietet Adress- und Datenbusse, um wahlfrei auf Speicherbereiche zuzugreifen. Es handelt sich also um eine Random-Access-Schnittstelle. Dies ermöglicht neben einem schnellen Lesen und Schreiben von einzelnen Bytes auch einen *Execute in Place (XiP)*, d.h ein Ausführen des Codes direkt auf dem Speicher, anstatt den Code in den Arbeitsspeicher zu kopieren und dann auszuführen. Grund dafür ist die besagte Random-Access-Schnittstelle mit ihrer feinen Adressierung, bei der schnell auf kleine Speichereinheiten zugegriffen werden kann. Beim NAND hingegen kann nur auf vergleichsweise große Bereiche zugegriffen werden und

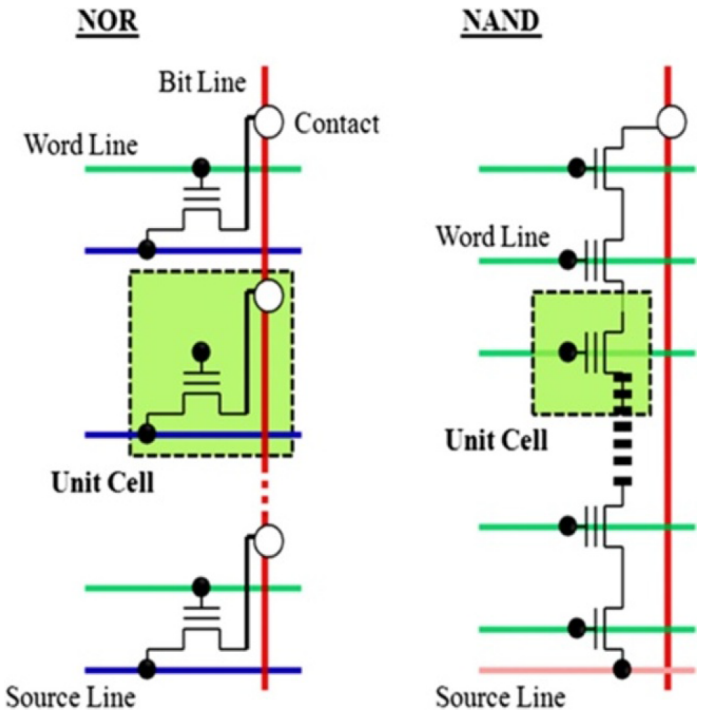


Abbildung 2.7.: Vergleich von NOR- und NAND-Speicherarrays [MSCT14]

deswegen wäre ein NAND zu langsam in der wahlfreien Lesegeschwindigkeit für diesen Anwendungsfall. Der XiP ist bei eingebetteten Applikationen beispielsweise erforderlich. Es ergeben sich aus dieser Architektur allerdings auch Nachteile. So ist die Performance bei Schreib- und Löschoperationen bei größeren Dateien (nicht individuelle Bytes) im Vergleich eher langsam. Dies kann durch den erzeugten Overhead begründet werden, der durch die Addresierung von vielen kleinen Einheiten für die Programmierung (Löschung) entseht. Außerdem verbraucht der parallele Ansatz deutlich mehr Fläche und erzeugt somit geringere Speicherdichten. Aus diesem Grund sind die wirtschaflich produzierten NOR-Speicher immer vergleichsweise klein (Speicherkapazität). Typischerweise haben NOR-Speicher eine Größe von mehreren KB bis hin zu einigen Gbit. Folglich werden NOR-Speicher als Codespeicher gerade für eingebettete Systeme verwendet, sind aber auch perfekt als Ersatz für ROM-Speicher geeignet, um das BIOS/Firmware zu speichern. Auch in Low-End-Mobiltelefonen oder Konsumgeräten werden diese verbaut [MSCT14].

Aufgrund der seriellen Verbindungsstruktur des NAND-Speichers, ergeben sich mehrere Vorteile gegenüber dem NOR. Die serielle Verbindung benötigt weniger Fläche, was eine höhere Speicherdichte zur Folge hat. Diese wiederum bewirkt, dass die Kosten pro Bit niedriger sind und außerdem aufgrund des niedrigeren Platzverbrauchs höhere Speichergrößen möglich sind. Ein klarer Vorteil liegt auch in der höheren Geschwindigkeit bei Schreib- und Leseoperationen größerer Datenmengen, da bei der seriellen Schnittstelle mehrere Zellen gleichzeitig angesprochen werden. Es ergeben sich auch Nachteile aus dem seriellen Ansatz. Performancenachteile entsehen beim NAND-Speicher bei zufälligem Zugriff auf Speicherbereiche, da dieser keine Random-Access-Schnittstelle sondern eine sequentielle Schnittstelle besitzt. Dies kann nur auf Systemebene erzielt werden, nicht jedoch auf Hardware-Ebene.

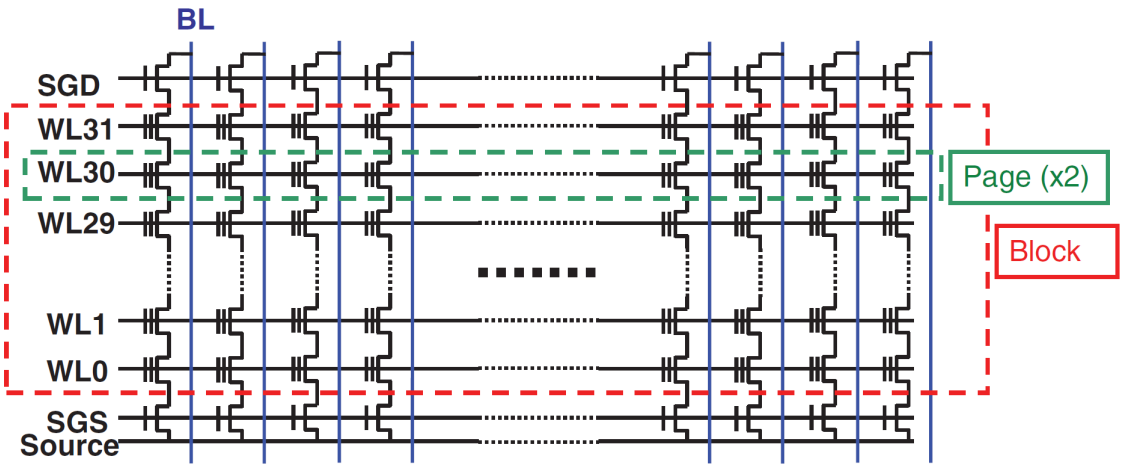


Abbildung 2.8.: Array-Architektur von NAND-Speicherzellen [ARI16]

Dazu kann der Vergleich zu einem Betriebsystem herangezogen werden, bei dem es scheinbar möglich ist gleichzeitig mehrere Prozesse auszuführen. Es findet jedoch keine echte Gleichzeitigkeit statt, sondern eine geschickte zeitliche Planung, die den Eindruck der Gleichzeitigkeit erweckt. Aus diesen Eigenschaften ergibt sich, dass NAND-Speicher hervorragend als größere Datenspeicher für beispielsweise Memory Cards geeignet sind um große Dateien, wie Video- oder Audiodateien zu speichern [MSCT14].

Unter Betrachtung der vorhergehenden Vor- und Nachteilen kann geschlussfolgert werden, dass der NAND-Speicher ideal als Datenspeicher für den Automotive-Bereich geeignet ist, wenn Speichergröße und Lese- und Schreibgeschwindigkeit betrachtet werden. Somit werden die Analysen und Tests in dieser Arbeit anhand von NAND-Speichern durchgeführt. Auf Abbildung 2.8 ist die spezielle NAND-Array-Architektur noch etwas genauer dargestellt. Um später in Unterkapitel 2.2.3 den Lese- und Schreibvorgang zu verstehen, müssen die Grundzüge der Array-Architektur bekannt sein.

Die blauen vertikalen Verbindungen stellen die Bitlines(BL) dar. Eine BL verbindet alle Transistoren innerhalb eines NAND-Zellen-Strings. Dieser String besteht hier aus 32 Zellen sowie zwei Auswahltransistoren. Die 32 Zellen sind in Serie verbunden und die zwei Auswahltransistoren (SGD, SGS) existieren, um diese Serie gegenüber Source und Drain isolieren zu können, soweit dies benötigt wird. Die horizontalen Leitungen nennen sich Wordlines (WL) und verbinden immer die Zellen mit der selben Nummer von verschiedenen Bitlines (z.B. WL0 schließt alle Zellen mit der Nummer null in horizontaler Richtung zusammen). Die Zellen einer WL bilden eine sogenannte Page (auf der Abbildung in grün). Diese ist typischerweise 2KB bis 16 KB groß. Ein Block (auf der Abbildung rot) wiederum hat die Größe: *Page * 32 Zellen * 2 = 128 KB bis zu 1024 KB*. Der Faktor zwei kommt hinzu, da 64 Pages in einem Block sind. Gelesen und geschrieben wird auf Page-Ebene und gelöscht auf Block-Ebene.

Zusammenfassend lässt sich hinsichtlich der Architektur festhalten, dass NAND-Flash mit seinem seriellen Zugriff auf die Einzelspeicherzellen die Flash-Variante für große Speicherdichten repräsentiert. Die Array-Architektur zeigt, wie die physikalischen Zellen auf logische Einheiten (Pages, Blöcke) abgebildet und gesteuert (BL, WL) werden. Mit diesen Informationen können nun der Lese-, Schreib- und Löschvorgang im Hinblick auf die physische

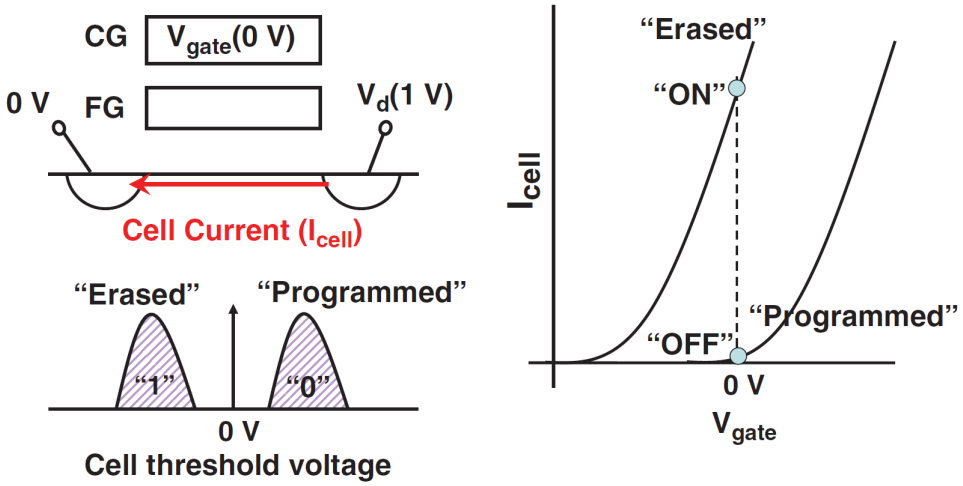


Abbildung 2.9.: Prinzip des Lesevorgangs einer Flash-Einzelzelle [ARI16]

und logische Auswirkung verstanden werden und somit auch die sich daraus ergebenden grundlegenden Performancecharakteristka dieser Operationen.

2.2.3. Lese- und Schreibvorgang

Da bereits die Architektur der Einzelzelle und des NAND-Zellen-Arrays bekannt sind, wird nun erklärt, wie diese Technologie das Lesen und Schreiben ermöglicht. Der Effekt, der das Lesen von Information ermöglicht, ist das schon erwähnte Speichern und Entladen des FG. Dies verdeutlicht Abbildung 2.9. Das grundlegende Funktionsweise der Flash-Zelle ist wie folgt: Wenn Ladung im FG gespeichert wird, so ändern sich die elektrischen Eigenschaften des gesamten Bauteils, da die aufgebrachte Ladung zusätzlich isolierend wirkt. Dies hat den Effekt, dass die Flash-Zelle am CG mit einer höheren Spannung aktiviert werden muss, damit die Zelle wieder Strom (Ic) leiten kann. Man nennt diese Spannung *Zellgrenzspannung* und die Verschiebung eine *Grenzverschiebung*. Jene Eigenschaft macht man sich nun zu nutze und setzt das CG auf eine gewisse Kontrollspannung (Vgate = 0V), wie in der Abbildung auf der rechten Seite zu sehen. Prüfsensoren kontrollieren dabei, ob ein Stromfluss entsteht. Ergibt sich kein Stromfluss, so kann schlussgefolgert werden, dass die Zelle programmiert ist, d.h. es befindet sich zusätzlich isolierende Ladung im FG. Kein Stromfluss wird also als eine logische 0 oder „OFF“erkannt. Fließt unter der angelegten Kontrollspannung ein Strom durch den Transistor, so kann geschlussfolgert werden, dass die Zelle gelöscht ist und damit einer logischen 1 oder dem Zustand „ON“entspricht [ARI16].

Aus Abbildung 2.10 lässt sich schließen, wie die einzelnen Elemente gesteuert werden, sodass gewisse Zellen ausgelesen werden können. Gelesen wird immer eine Page, also eine Reihe von Zellen, die durch eine einzige WL angesprochen wird. Die WL steuert dabei alle CGs in dieser Line. An der zu lesenden WL werden die genannten Vgate = 0V zur Lesekontrolle angebracht. Alle anderen nicht ausgewählten WLs werden auf VpassR = 6V gesetzt. Bei einer derart hohen Spannung werden alle Transistoren leitend und agieren somit als sogenannte *Durchlasstransistoren*. Die Informationen werden folglich weitergeleitet. An den zu lesenden Einzelzellen wird die jeweilige BL, die immer die Source des Transistors steuert auf 1V gesetzt, um eine Spannung zwischen den 0V Drain und der Source zu erzeugen.

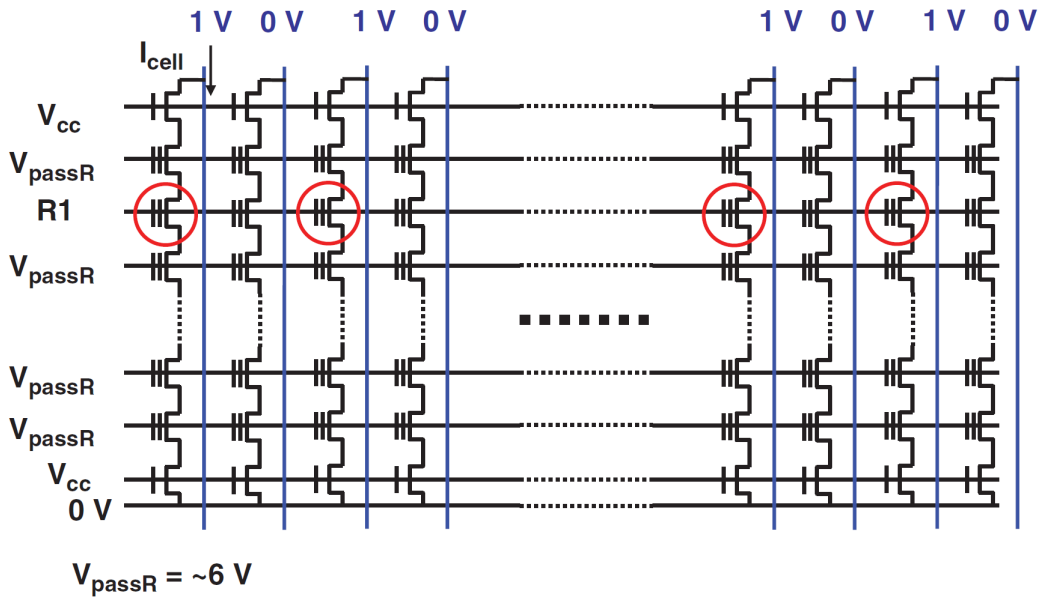


Abbildung 2.10.: Schaltung einer Leseoperation bei einem NAND-Array [ARI16]

Die BLs der nicht auszulesenden Einzelzellen werden somit auf 0V geschaltet. Es lässt sich folgende Erkenntnis schließen: die nicht ausgewählten Zellen, beziehungsweise die Zellen, deren WL auf 6V (hohe Spannung) geschaltet wurden, bekommen eine hohe Spannung ab um leitend zu werden. Dieser Umstand kann dazu führen, dass diese Transistoren gestört werden. Eine detaillierte Erklärung gibt Unterkapitel 2.3 [ARI16].

Eine Einzelzelle wird programmiert und eine Grenzspannungsverschiebung erzeugt. Durch eine hohe Spannung am CG und gleichzeitig niedrige Spannung am Substrat wird Ladung durch das Tunneloxid ins FG injiziert (programmiert). Um den Speicher zu löschen, wird genau das Gegenteil gemacht. Dies ist auf Abbildung 2.11 anschaulich verdeutlicht.

Auf Abbildung 2.12 ist die Programmierung eines NAND-Flasharrays zu sehen. Es wird immer eine WL tiefer von oben angefangen programmiert. Die WL der nicht zu programmierenden Zellen werden, ähnlich wie bei der Leseoperation, mit einer hohen Spannung durchlässig geschaltet. Die BL der zu programmierenden Zelle wird dann auf 0V gesetzt, um ein maximales Spannungsgefälle zwischen FG und Substrat zu erzeugen. So kann Ladung in das FG injiziert werden. Beim Löschvorgang wird das Spannungsgefälle einfach umgedreht. Die Löschoperation wird jedoch nicht mit einzelnen WLs ausgeführt, sondern gleichzeitig auf einem gesamten Block. Sowohl Programmier- als auch Löschoperation sind andauernde Vorgänge und keine einmaligen Operationen. Das bedeutet, nachdem Elektronen bewegt wurden, muss immer kontrolliert werden, ob die Aktion erfolgreich war. Deshalb wird in Pulsen programmiert und gelöscht. Es wird also ein Spannungspotential für eine kurze Dauer (z.B. 200μs) angelegt und von einer kurzen Kontrollspannung (*Verify*) gefolgt, die den Erfolg der Aktion kontrolliert. Dauer und Potenial des Spannungspulses sind hier die essentiellen Faktoren, die Löschen vom Schreiben unterscheiden. Je „mächtiger“der Puls (Spannung, Dauer), desto schneller bewegen sich genug Elektronen, um den erwünschten Zustand zu erzielen. Damit wird jedoch die Aktion gleichzeitig destruktiver für den Speicher. Es muss also ein Gleichgewicht zwischen Performance und Zuverlässigkeit gefunden wer-

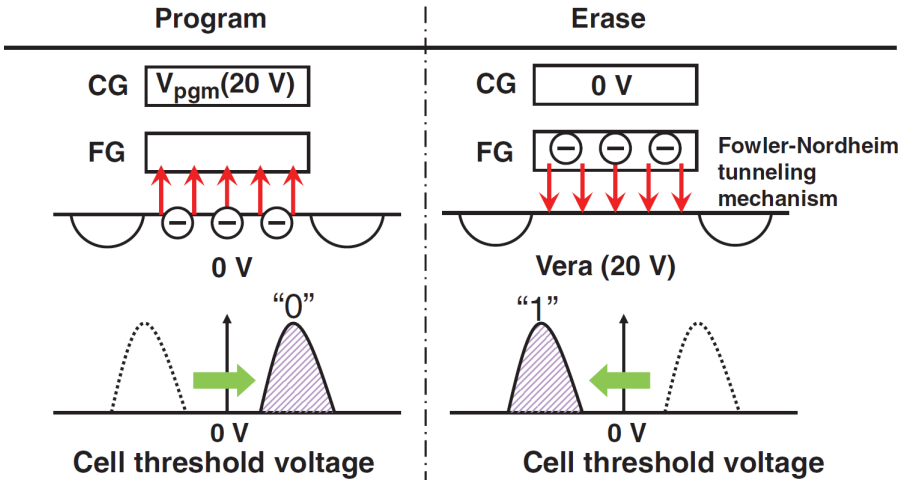


Abbildung 2.11.: Löschen und Programmieren einer Flash-Einzelzelle [ARI16]

den. Programmiert kann dementsprechend deutlich schneller als gelöscht werden, da beim Löschen alle Zellen in einem Speicherarray *gleichzeitig* unter hoher Spannung stehen und deswegen besonders stark abgenutzt werden. Deshalb werden die Pulse hier kürzer und über einen längeren Zeitraum ausgeführt. Folglich ergibt sic die Tatsache, dass eine Schreibaktion im μs-Bereich und eine Löschaktion im ms Bereich liegt, also deutlich langsamer ist[ARI16].

Zusammenfassend kann hier festgehalten werden, dass sich aus dem physikalischen Vorgang des Elektronentransports in den Flash-Zellen unterschiedliche Performance- und Haltbarkeitseigenschaften ableiten lassen. Leseaktionen sind im Vergleich am schnellsten (ca. 15μs) und am wenigsten destruktiv für die Speicherzelle, aufgrund niedriger Kontrollspannung. Schreib- und Löschaktionen sind deutlich langsamer (Schreiben 200μs, Löschen 2ms), wobei die Löschaktion die Zelle am deutlichsten strapaziert. Löschoperationen müssen folglich so gut es geht vermieden werden. Dies ist Aufgabe des Controllers und wird in Unterkapitel2.5.2 behandelt.

2.2.4. Multilevel-Zellen

Die Idee bei diesem Ansatz ist es, die Speicherdichte zu vergrößern, ohne dabei die physikalische Größe der Zelle zu verändern. Möglich ist das, in dem ein FG die Fähigkeit besitzt, verschiedene Mengen an Ladung aufzunehmen. Folglich bewirken dann auch verschiedene Grenzspannungen den Stromfluss der Flash-Zelle. Abbildung 2.13 zeigt den Multilevelansatz mit seinen vier verschiedenen Grenzspannungen. Diese Spannungen korrespondieren jeweils mit einer bestimmten Bitfolge. Bei dem ersten Multilevel-Ansatz, der sogenannten *Multi Level Cell (MLC)*, werden hierbei zwei Bit an Information darstellbar. Außerdem gibt es noch die *Triple Level Cell (TLC)*, bei der 3 Bits zu Verfügung stehen und die *Quad Level Cell (QLC)*, bei der sogar 4 Bit an Information innerhalb einer einzigen Flashzelle gespeichert werden können. Die Art von Einzelzelle, die vorher besprochen wurde und nur ein einzelnes Bit speichern kann, wird *Single Level Cell (SLC)* genannt. Die Struktur der Einzelzelle und der Herstellungsprozess sind im Prinzip bei SLC und MLC gleich. Neben dem offensichtlichen Vorteil der erhöhten Speicherdichte pro Zelle bringt dieser Ansatz jedoch auch Hürden mit sich. Damit mehrere Zustände gelesen werden können, müssen die Span-

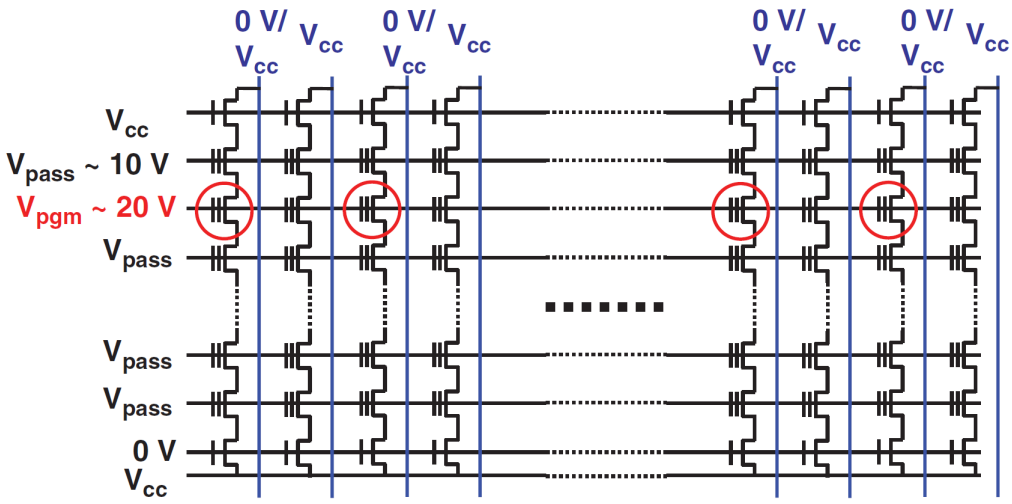


Abbildung 2.12.: Programmierung eines NAND-Flasharrays [ARI16]

nungskurven enger werden und näher zusammenrücken, wie auf Abbildung 2.13 zu erkennen ist. Auf dieser Darstellung wird allerdings „nur“eine MLC mit 2 Bit Information gezeigt. Dies lässt erahnen, wie genau diese Verteilung bei einer QLC werden muss. Natürlich kann es bei einer solchen Zelle mit schmäleren Kurven und somit einem kleineren Bereich, der vom Sensor als die richtige Information ausgelesen wird, schneller dazu kommen, dass eine falsche Information ausgelesen wird. Gerade im Hinblick auf Abnutzung, sind diese Zellen deshalb deutlich schlechter, da die Zustände schneller nicht mehr erkannt werden können. Außerdem müssen diese Zellen bei der Schreiboperation viel genauer programmiert werden. Somit steigt die Schreibdauer pro Bit und damit sinkt die Schreibgeschwindigkeit. Um diesen negativen Effekten entgegenzuwirken, wurde auf Systemebene der „Moving Read Algorithmus“ entwickelt. Dieser passt intelligent die Auslesespannung am ausgewählten CG den von physikalischen Effekten beeinflussten Grenzspannungen an. Dies erhöt deshalb die Haltbarkeit. Aus den negativen Eigenschaften kann geschlussfolgert werden, dass Multilevel-Zellen zwar größere Speichermengen ermöglichen, jedoch in der Qualität Defizite mit sich bringen. Dabei gilt, je mehr Bit pro Zelle gespeichert werden sollen, desto schlechter wird die Qualität. Deshalb werden TLC-Speicher beispielsweise für USB-Speichersticks verwendet, da hier ausschlaggebend ist möglichst viel Daten auf kleinstem Raum abzusichern, jedoch ein geringer Datenverlust tolerierbar ist [ARI16].

Die aufgeführten Punkte bringen die Erkenntnis, dass mit einer steigenden Anzahl an Bits pro Zelle die Speicherdichte des Flash-Speichers steigt, jedoch die Zuverlässigkeit und die Performance sinken. Die Performanceeinbusen werden durch einen umfangreicheren Kontrollprozess bei den Operationen (Lesen, Schreiben, Löschen) auf der Flash-Zelle bedingt, da hier mehrere Spannungszustände als ursprünglich zwei (SLC) unterschieden werden müssen und somit genauer gearbeitet werden muss. Die Zuverlässigkeit sinkt, da es schneller durch Abnutzung zu dem Situation kommt in der nicht mehr alle Zustände verlässlich unterschieden werden können, da es bereits ausreichend ist, wenn wenige Elektronen verloren gehen, um eine dauerhafte Grenzspannungsverschiebung zu bewirken.

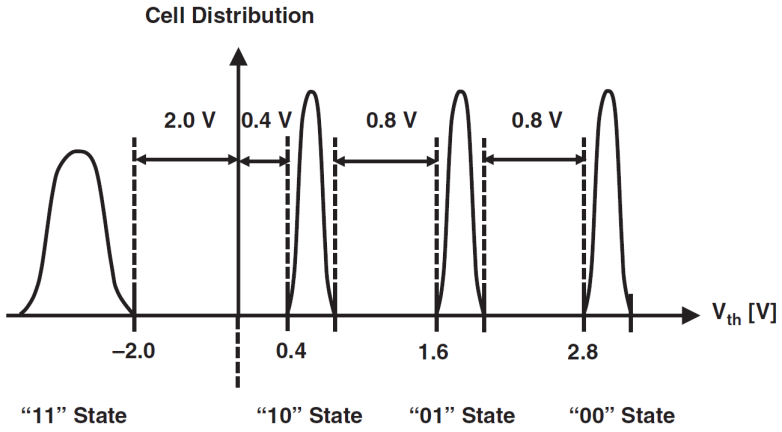


Abbildung 2.13.: Verteilung der Grenzspannungen auf vier Zustände bei Multilevel-Zelle [ARI16]

2.3. Zuverlässigkeit

In diesem Unterkapitel werden die diversen Hürden der Flash-Technologie besprochen, die beispielsweise den Prozess der Erhöhung der Speicherdichte oder die Steigerung der Performance beeinträchtigen.

2.3.1. Data Retention

Der Begriff „Data Retention“ definiert sich als die Fähigkeit eines Speichers seine Information über einen längeren Zeitraum speichern zu können. Wie auch andere nichtflüchtige Speicher, sind Flash-Produkte spezifiziert ihre Information über zehn Jahre zu behalten. Das bedeutet, dass der Verlust an Ladung im FG über die Jahre hinweg minimal bleiben muss, um noch die richtigen Zustände auslesen zu können. Bei modernen Flash-Speichern ist die Zellgröße sehr stark geschrumpft und damit auch die Kapazität der Zelle. Verliert die Zelle 20 % ihrer Ladung (einige Elektronen pro Monat), kann dies schon zum Auslesen eines falschen Wertes führen. Die größten Probleme bei der Data Retention haben Zellen, die ihren programmierten Zustand beibehalten müssen, da der programmierte Zustand mit einem elektronenbefüllten FG korrespondiert [BCMV03].

Es gibt mehrere Gründe für den Verlust der Ladung [BCMV03]:

- Defekte im Tunneloxid
- Defekte in der Isolationsschicht (Dielektrikum)
- Kontamination durch mobile Ionen
- Das Freilassen von Ladung aus der Isolationsschicht rund um das FG

Die Data Retention ist abhängig von mehreren Charakteristiken. Es gibt sogenannte „Program and Erase (P/E) Schemes“. Diese Schemata beinhalten im Prinzip die Information, mit welchen Verfahren Programmier- und Löschoperationen durchgeführt werden. Beispielsweise kann solch ein Schema festlegen, dass anhand des Fowler-Nordheim-Effekts programmiert wird und ein anderer Effekt für das Löschen verwendet wird. Welches Schema

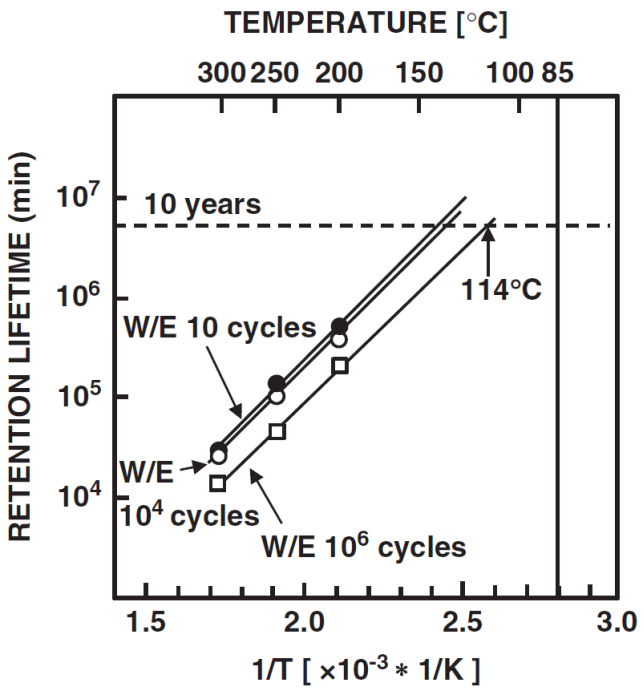


Abbildung 2.14.: Schätzung der Data Retention-Zeit im Bezug auf die Betriebstemperatur [ARI16]

nun verwendet wird hat enormen Einfluss auf die Dauer der Data Retention. Aus der Forschung ging hervor, dass ein spezielles Schema, das „Uniform Program and Erase Schema“die Data Retention enorm steigern kann [ARI16].

Ein weiterer Faktor ist der Gebrauchszustand des Speichers im Hinblick auf P/E-Cycles. In Unterkapitel 2.2.3 wurde der Lese- und Schreibvorgang bereits erläutert. Es wird in der Einheit Block (bestehend aus mehreren Pages) gelöscht und in der kleineren Einheit Page geschrieben. Ein P/E-Cycle ist nun der Vorgang des einmaligen Löschens eines Blockes mit anschließender Beschreibung aller Pages innerhalb dieses Blockes, bis dieser sozusagen voll ist und wieder gelöscht werden müsste. Es zeigt sich, dass P/E-Cycling die Data Retention-Zeit verkürzt. Jedoch wirkt das Uniform P/E Schema diesem Effekt in einem gewissen Maße entgegen und lässt auch noch bei einer P/E-Cycle-Anzahl von einer Millionen eine akzeptable Data Retention von zehn Jahren zu. Der negative Effekt bleibt jedoch erhalten [ARI16].

Der nächste Faktor ist Temperaturabhängigkeit. Eine erhöhte Temperatur (150-300 Grad Celsius) führt zu einer deutlich verkürzten Data Retention-Zeit. Je höher dabei die Temperatur ist, desto stärker ist dabei dieser Effekt zu beobachten. Dies wird durch das vorher besprochene P/E-Cycling noch verstärkt. Mit dem Wissen über die Temperaturabhängigkeit kann die Data Retention über Zeit durch eine erhöhte Temperatur simuliert werden. Auf Abbildung 2.14 wurde dieser Ansatz verwendet und eine Schätzung der Data Retention Zeit im Bezug auf die Betriebstemperatur und die vorher getätigten P/E-Cycles dargestellt. Hier zeigt sich, dass bei einer Betriebstemperatur von 100°C die besagten zehn Jahre Data Retention-Zeit auch bei 1 Millionen P/E-Cycles garantiert werden können [ARI16].

Ein weitere wichtige Einflussgröße bildet die Dicke des Tunneloxids. Auf Abbildung 2.15 ist ersichtlich, dass diese Beziehung nicht linear und nicht intuitiv ist. Für die dünnen Tunne-

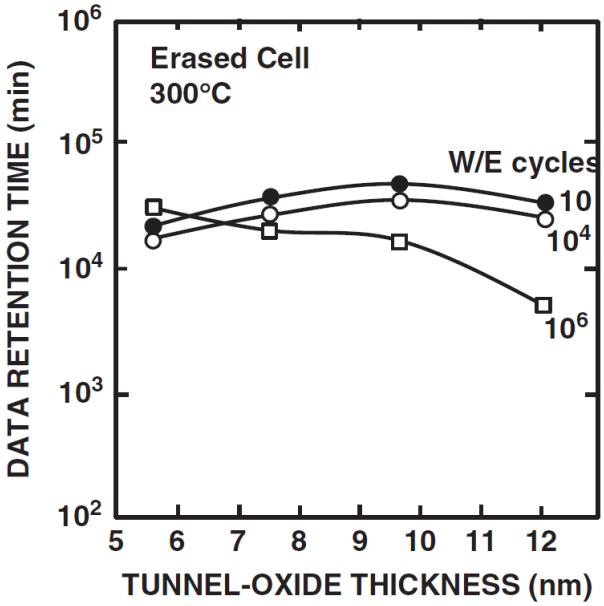


Abbildung 2.15.: Data Retention-Zeit bei 300°C in Abhängigkeit der Tunneloxiddicke und den P/E-Cycles[ARI16]

loxidschichten verkürzt sich die Data Retention-Zeit bei 10-10.000 P/E-Cycles, also im niedrigen bis mittleren Abnutzungsbereich. Jedoch bei 1 Millionen P/E-Cycles, also einer enormen Abnutzung verlängert sich diese wieder erkennbar. Dafür verantwortlich sind spezielle physikalische Effekte, deren Erläuterung den Rahmen dieser Arbeit überschreiten würden. Als Schlussfolgerung lässt sich demnach feststellen, dass die Data Retention-Zeit im Extremfall der starken Abnutzung (1 Million P/E-Cycles) nicht durch die Dicke der Tunneloxidschicht limitiert wird. Jedoch wirkt sich eine sehr dünne Oxidschicht sehr wohl im nutzerrelevanten Abnutzungsbereich negativ auf die Data-Retention-Zeit aus.

2.3.2. Read/Program Disturb

Bei Read/Program Disturbs handelt es sich um einen Fehlermechanismus, bei dem eine Zelle beim Schreib- oder Lesevorgang ungewollt in ihrer elektrischen Eigenschaft (Grenzspannung) verändert wird. Die Veränderung wird durch elektrische Beanspruchung bedingt, die auf eine Zelle wirkt, während andere Zellen programmiert oder gelesen werden. Der Grund hierfür ist die Architektur der Flash-Zelle und wurde bereits in Kapitel 2.2.3 beschrieben. Da alle nicht beteiligten Zellen beim Lese- oder Schreibvorgang durchlässig geschaltet werden müssen und dafür eine relativ hohe Spannung notwendig ist, werden diese Zellen abgenutzt. Es gibt zwei Arten von Disturbs (Störungen):

- Row Disturbs
- Column Disturbs

Abbildung 2.16 zeigt diese beiden Beanspruchungssarten. Row Disturbs werden durch elektrische Beanspruchung am CG bedingt. Dieser entsteht, wenn andere Zellen in der selben WL programmiert oder gelesen werden. Wenn eine hohe Spannung am CG angelegt

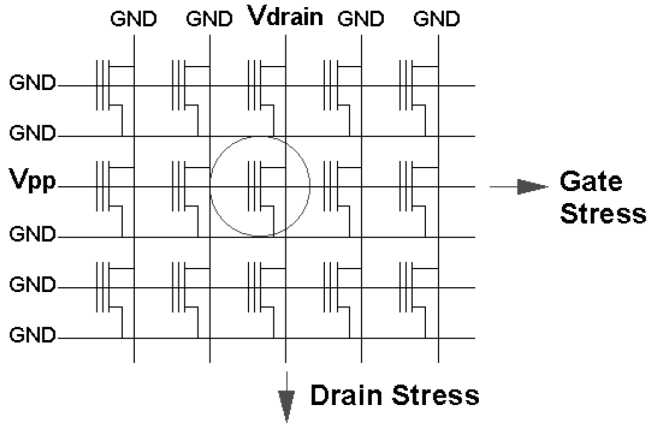


Abbildung 2.16.: Schema eines Flash-Arrays, das durch P/E-Cycles entstandene Row und Column Disturbs zeigt[BCMV03]

wird, müssen alle anderen Zellen in dieser Reihe dem elektrischen Stress trotzen und ihre Ladung behalten. Abhängig von den Daten, die in der Zelle aufbewahrt werden, können diese entweder durch einen Ladungsverlust am FG oder durch einen Verlust am Dielektrikum verloren werden [BCMV03].

Column Disturbs werden durch elektrischen Stress an der Drain einer Zelle beim Lese- oder Schreibvorgang einer anderen Zelle auf der selben BL bedingt. Somit können programmierte Zellen ihre Ladung vom FG an die Drain verlieren. Dies nennt sich „Soft Erasing“ und erfolgt durch FN-Tunneln. Diese Störung hängt stark von der Anzahl an Zellen innerhalb einer BL und einer WL ab [BCMV03].

2.3.3. Program/Erase Standhaftigkeit

Generell sind Flash-Produkte für ca. 100.000 P/E-Cycles spezifiziert (kann für verschiedene Industriestandards abweichen). Dabei ist es bekannt dass das P/E-Cycling eine gleichmäßige Verschlechterung der Zell-Performance bewirkt. Dafür verantwortlich ist hauptsächlich die Abnutzung des Tunneloxids, welche die Standhaftigkeit der Zelle limitiert. Auf Abbildung 2.17 ist zu erkennen, wie sich über die fortschreitenden P/E-Cycles die Programmier- sowie die Löschgrenzspannung gegenseitig annähern und somit ein immer kleiner werdendes „Fenster schließen“. Die Fensterbildung ist ein Maß für die Alterung des Tunneloxids und somit einer Speicherzelle. Bei Implementierungen von Flash-Produkten werden intelligente Algorithmen verwendet, um diese Fensterbildung zu verhindern. Dabei entsteht der Nebeneffekt, dass sich sowohl die Programmier- als auch die Löschzeiten der Zelle verlängern. Die Verlängerung jener Zeiten, ist ein wichtiger Hinweis auf die Performanceverminderung bei gealterten Speichern [BCMV03].

Sehr wichtig für dieses Thema ist auch die Erkenntnis, dass es mehrere Trade-Off Beziehungen im Bezug auf das P/E-Cycling gibt. Die Abbildungen 2.18 und 2.19 zeigen diese Trade-Off-Beziehungen. Die Data Retention-Zeit ist immer ein Kompromiss mit der Ausdauer bei den P/E-Cycles. Das bedeutet bei der Erstellung eines NAND-Flash-Produktes muss genau festgelegt werden, welche Prioritäten wichtig für den Kunden sind. Also entweder eine sehr lange Data Retention-Zeit oder eine sehr hohe Ausdauer oder ein Kompromiss

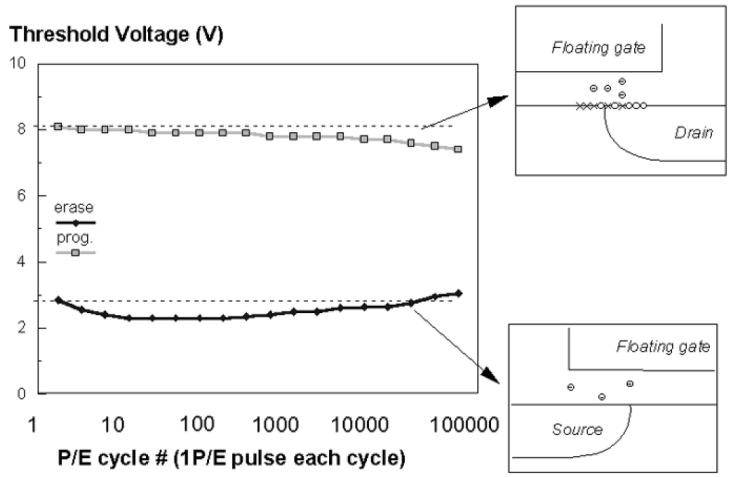


Abbildung 2.17.: Bilden eines Spannungsfensters als eine Funktion von P/E-Cycles bei einer Flash-Einzelzelle [BCMV03]

mit Mittelwerten aus beiden Feldern. Dieselbe Beziehung gilt auch für die Programmiergeschwindigkeit und die Ausdauer beim P/E-Cycling. Je höher die Performance bei der Schreibgeschwindkeit, umso niedriger wird die Ausdauer [ARI16].

Hinsichtlich der Zuverlässigkeit ist zusammenfassend festzustellen, dass dieses Thema besonders die Schreib- und Löschaktionen (z.B. P/E-Cycles) betrifft, auch wenn das Lesen durchaus negative Nebeneffekte (Read Disturb) erzeugen kann. Die Zuverlässigkeit hat nur einen indirekten Einfluss auf diese Arbeit, der sich jedoch massiv auswirken kann. Da die Zuverlässigkeit des Flash-Speichers in vielen Applikationen (gerade im automotiven Umfeld) hohe Priorität besitzt, muss dieses Thema von den Speicherherstellern adressiert werden. Es ist vereinfacht gesagt notwendig zusätzlichen Aufwand zu betreiben, um Daten aufzufrischen und somit die Data Retention zu verlängern. Vor allem muss effizient mit Löschaktionen umgegangen werden. Folglich entsteht zusätzlicher Mehraufwand im Speicher, den der Benutzer (Host) nicht bewusst steuert und der einen essentiellen Einfluss auf die Performance hat. Es ist also wichtig zu wissen, dass gewisse Eigenschaften, die sich positiv auf die Zuverlässigkeit des Speichers auswirken, gleichzeitig einen negativen Einfluss auf die Performance haben. Eine detailliertere Beschreibung folgt im Unterkapitel2.5.2.

2.4. 3-D-Architektur

Dieser Teil der Arbeit dient dazu, ein Verständnis für den technologischen Wandel in der NAND-Flash-Technologie vermittelt werden. Der treibende Faktor für diesen Wandel war das Erreichen einer höheren Speicherdichte, da dadurch der Preis pro Bit sinkt und somit Kosten gespart werden können.

2.4.1. 2D zu 3D

Die bisher beschriebene Architektur der Flash-Technologie wird als 2-D-Architektur bezeichnet, da die Einzelzellen in einer planaren Struktur angeordnet sind. Diese Architektur stößt jedoch aktuell schon an ihre Skalierungsgrenzen. Um eine höhere Speicherdichte zu erlangen

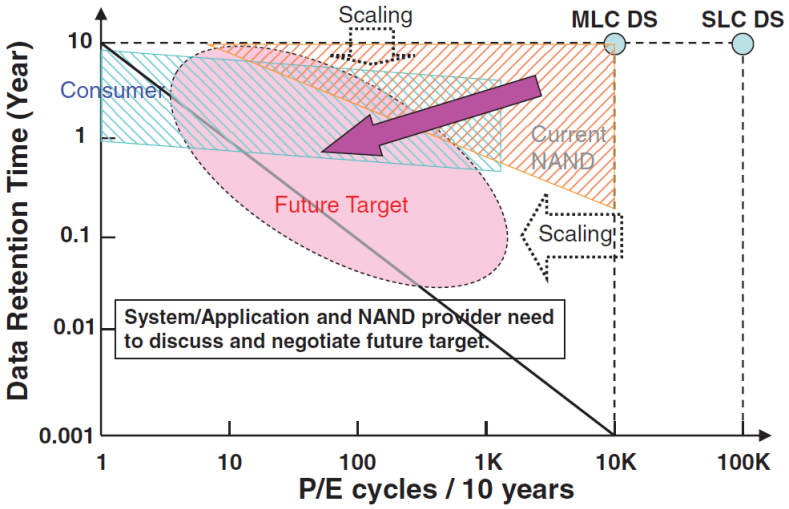


Abbildung 2.18.: Trade-Off zwischen Data Retention-Zeit und P/E-Cycles[ARI16]

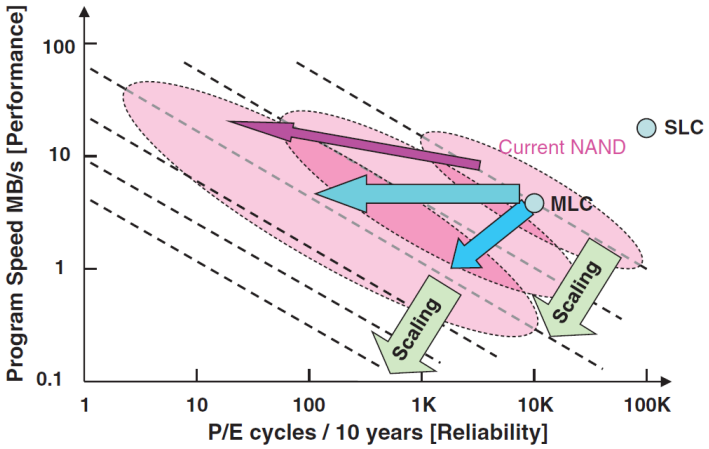


Abbildung 2.19.: Trade-Off zwischen Prgrammiergeschwindigkeit und P/E-Cycles [ARI16]

Number of electrons per logic level

Node	45nm	32nm	22nm	16nm	11nm	8nm
SLC (2 levels)	400	200	100	50	25	13
MLC (4 levels)	140	70	35	18	10	5
TLC (8 levels)	60	30	15	8	4	2
QLC (16 levels)	30	15	8	4	2	1

Abbildung 2.20.: Gespeicherte Elektronenanzahl bei verschiedenen Strukturgrößen und Bittiefen pro Zelle [ARI16]

gab es nach der Entwicklung der MLC, bei der mehrere Bits in einer Zelle gespeichert werden konnten, nur noch die Möglichkeit, die Feature Size der einzelnen Zelle weiter zu verkleinern. Das Kapitel 2.3 erläuterte bereits, dass diese Schrumpfung, besonders die Reduzierung der Tunneloxidicke, zu Problemen in der Zuverlässigkeit führen kann und gewissermaßen an eine Grenze des technisch Realisierbaren kam. Abbildung 2.20 verdeutlicht die Skalierungsprobleme der 2D-Technologie. Hier wird gezeigt, wie viele Elektronen eine Flash-Einzelzelle, bei gewissen Strukturgrößen und verschiedenen Bittiefen pro Zelle speichert. Die eingezeichnete Diagonale zeigt, ab wann die Elektronenanzahl pro Zelle zu niedrig wird, um die Technologie zu realisieren. Eine SLC-Zelle mit einer 8nm-Technolgie kann beispielsweise nicht mehr realisiert werden, da die 13 Elektronen pro Zelle nicht mehr ausreichen, um die zwei verschiedenen Zustände der Zelle darzustellen und auszulesen. Dabei wird klar, dass die Strukturgröße eine klare Grenze für die 2D-Technologie darstellt. Aus diesem Grund wurde seit 2007 die Forschung an einem gänzlich innovativem Ansatz begonnen. Die 3-D-Architektur wurde vorgeschlagen. Die erste vorgeschlagene Architektur war die sogenannte *Bit Cost Scalable Technology (BiCS)* . Diese Speicher konnten nun in der Vertikalen wachsen und somit musste die einzelne Zelle (Feature Size vgl. Unterkapitel 2.1.2) an sich nicht mehr verkleinert werden [ARI16].

Aufgrund der deutlich vorteilhafteren Architektur wurden enorme Investitionen in der 3-D-Forschung getätigt und ab 2013 wurde schließlich der erste 3-D-Speicher in den Markt eingeführt. Dieser war ein MLC mit 24 Schichten. Die verwendete 3-D-Technologie war dabei der sogenannte *Vertical NAND (V-NAND)*. Folglich setzte sich 3D auch auf dem Markt immer stärker durch und überholte 2D im Bezug auf verkaufte Bits Mitte des Jahres 2017, wie auf Abbildung 2.21 zu erkennen ist [1]. Hier ist der Cross-Over-Punkt der beiden Technologien zu erkennen [ARI16].

Die Abbildung 2.22 stellt die zeitliche Entwicklung ab 2013 von 2D und 3D dar. Hier können die zwei verschiedenen Entwicklungstrends beobachtet werden. Während die 2D-Technologie auf immer kleinere Zellstrukturen setzt, wird im 3D-Bereich die Anzahl der

[1] Präsentation von Sandisk zu 3D-NAND https://www.slideshare.net/sandisk/storage-class-memory-learning-from-3d-nand Stand 18.01.2018

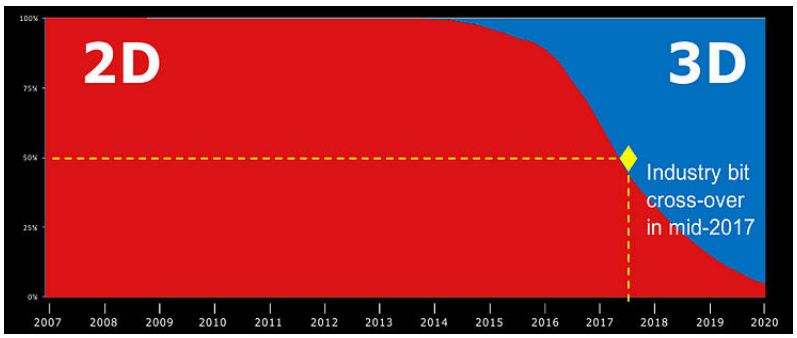


Abbildung 2.21.: Die veraltete 2D-Technologie wird von der zukunftsträchtigen 3D-Technologie im Rennen um verkaufte Bits überholt [ARI16]

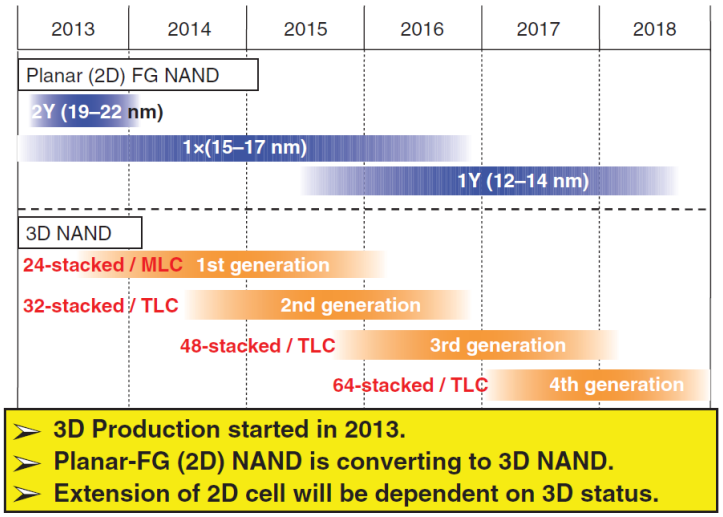


Abbildung 2.22.: Übergang von der planaren 2D-NAND-Zelle zu einer 3D-NAND-Zelle[ARI16]

Speicherschichten, die übereinandergestapelt werden, erhöht [ARI16].

2.4.2. Grundprinzip

Bei der 3D-Architektur wurden die NAND-Strings (BLs) von einer horizontalen in eine vertikale Ebene gebracht. Dieser Vorgang wird auf Abbildung 2.23 verdeutlicht.

Daraus resultiert, dass der NAND-String nun in die vertikale „wächst“ und somit die Erhöhung der Speicherdichte nicht mehr von der Verkleinerung der Feature Size (horizontale Ebene) abhängt. Zusätzlich wurde die Speicherdichte durch das Stapeln von Speicherschichten übereinander erhöht, wobei die Chipgrundfläche dabei konstant bleiben konnte. Auf Abbildung 2.24 ist die erste Umsetzung eines 3D-Speichers, die BiCS abgebildet. Auf der linken Seite sieht man den realen Speicher unter einem Elektronen-Mikroskop vergrößert und auf der rechten Seite die dazugehörige Schaltung. Wichtig zu erkennen ist hier, wie die vorher planaren BLs nun in die vertikale Ebene gedreht wurden. Das Grundprinzip ist weiter

[2]Quelle: Interne Präsentation von BMW

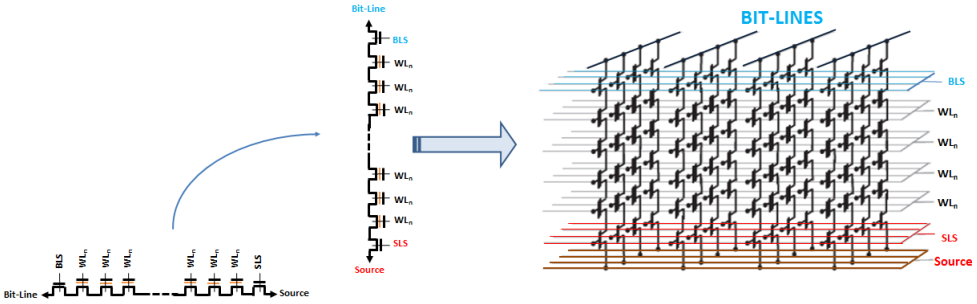


Abbildung 2.23.: Übergang des NAND-Strings von horizontaler Ebene in die Vertikale[2]

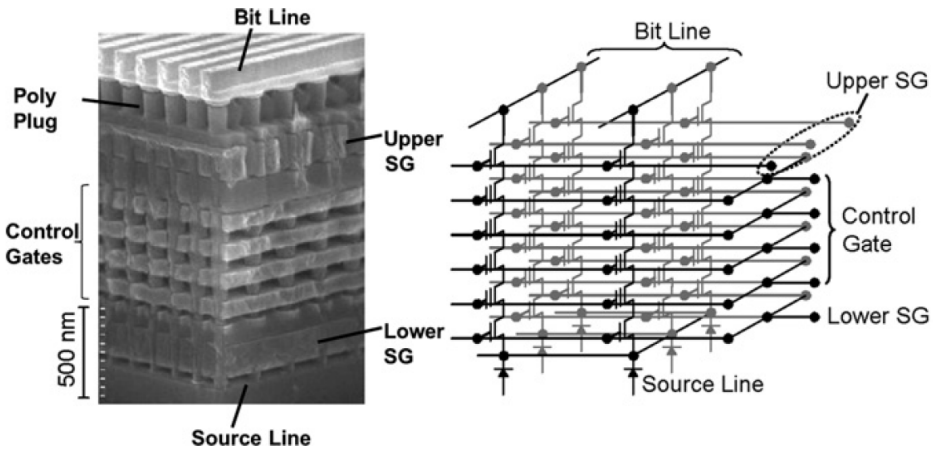


Abbildung 2.24.: BiCS-Zelle unter Rasterelektronenmikroskop und äquivalente Schaltung[ARI16]

nicht sehr verschieden von der 2D-Architektur, deswegen ist die konkrete Implementierung dieser Technik das große Forschungsthema. Einen großen Unterschied gibt es zur grundsätzlichen 2D-Flash-Zelle, zumindest bei den meisten Implementierungen von 3D. Während die 2D-Flash-Zelle ihre Ladung in einem Floating Gate abspeichert, wird die Ladung bei den meisten 3D-Implementierungen in sogenannten *Charge Traps (CTs)* abgespeichert. Diese Technologie ist nicht neu, trotzdem wurde sie bei den 2D-Flash-Speichern kaum verwendet. Beim CT werden die Ladungen nicht im Floating Gate gespeichtert, sondern es werden Elekronen in einer Schicht aus Siliciumnitrid „gefangen“. Es gibt allerdings auch eine Implementierung, bei der noch ein Floating Gate verwendet wird. Im nächsten Unterkapitel gibt es dazu eine Übersicht [ARI16].

2.4.3. Übersicht Architekturen

Auf Abbildung 2.25 sind die vier wichtigsten Implementierungen des 3D-Ansatzes mit ihren verschiedenen Eigenschaften dargestellt. Schon beim alleinigen Betrachten der konkreten Architektur fällt auf, dass hier fundamentale Differenzen vorhanden sind. Die 3D-Architekturen unterscheiden sich viel gegenseitig, im Vergleich zu verschiedenen 2D-Implementierungen untereinander. Auffällig ist auch, dass nur eine der vier Technologien die FG-Zelle verwendet. Eine weitere Ungleichheit ist die Geometrie des NAND-Strings. Während zwei der Techno-

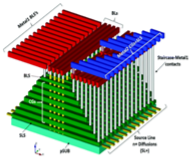
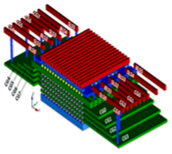
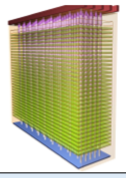
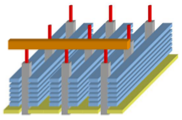
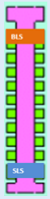
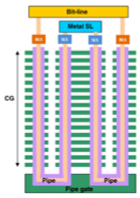
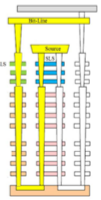
	TCAT	P-BiCS	3DFG	SMArT
				
Marke	V-NAND	BiCS Flash	3D-NAND	3D-NAND
Architektur	TCAT Terabit Cell Array Transistor	P-BiCS Pipe-Bit Cost Scaling	3DFG 3D Floating Gate	SMArT Stacked Memory Array Trans.
Eigenschaften	512Gb, 64L	512Gb, 64L	384Gb, 64L	128Gb, 36L
Zelleneigenschaft	CT, TLC	CT, TLC	FG, TLC	CT, MLC
NAND-String U-String vs. I-String				

Abbildung 2.25.: Wichtigste 3D-Architekturen [BCMV03][ARI16][Mic16][Pri14]

logien einen String in I-Form besitzen, haben zwei Strings eine U-Form. Hier verdeutlicht sich ein weiteres mal, wie inhomogen die Implementierungen sind. Das Gleiche gilt auch für die Zuverlässigkeit. Bei diesem Thema unterscheiden sich generell 2D- und 3D-Technologien, sowie 3D-Technologien untereinander in Konzept, Material und Fertigung [ARI16].

Das Thema 3D ist sowohl technologisch als auch prozesstechnisch hochkomplex und eine genauere Betrachtung würde den Rahmen dieser Arbeit sprengen. Bei weiterem Interesse sind die folgenden Quellen zu empfehlen: [BCMV03][ARI16][Mic16][Pri14]. Für diese Arbeit können dennoch wichtige Informationen aus dieser Thematik gewonnen werden. Dadurch dass die Feature-Size bei der 3-D-Technolgie wieder deutlich steigt (von 11nm auf 40nm), wird das Problem der zu wenigen Elektronen in der Einzelzelle behoben. Folglich können die Operationen (Lesen, Schreiben, Löschen) wieder schneller ausgeführt werden, da die Unterscheidungsbereiche der abzulesenden Grenzspannungen wieder größer werden. Somit wird der Verifizierungsprozess leichter und damit zeitsparender. Diese Tatsache sollte auch eine verbesserte Zuverlässigkeit *ohne* zusätzlichen Aufwand für den Speichercontroller bedeuten. Es ergeben sich jedoch nicht nur Vorteile. Durch das Aufeinanderstapeln von bis zu 64 NAND-Schichten, wachsen die NAND-Strings rapide an (siehe Abbildung2.23). Die Strecke, die die Elektronen innerhalb einer Bitline zurücklegen müssen, werden somit auch stark verlängert. Es können Performanceprobleme entstehen, weil es immer schwieriger wird die verlängerten Laufzeiten und die vermehrt spürbare Elektronenträgheit zu koordinieren[Ent].

Es gibt Punkte, die für eine verbesserte Performance sprechen und Punkte, die dagegen sprechen. Folglich muss die Performance von 3D-Speichern mit der von 2-D-Speichern verglichen werden, um praktische Ergebnisse und somit mehr Informationen zu gewinnen.

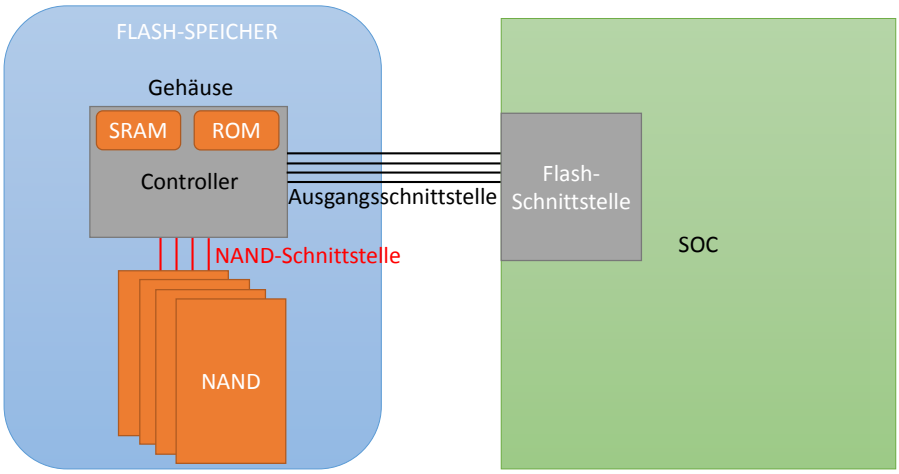


Abbildung 2.26.: Hardwareumfeld eines Flash-Arrays mit Speicher-Package und SoC

2.5. Systemisches Umfeld des NAND-Speichers

Dieser Teil der Arbeit setzt sich mit der Fragestellung auseinander, wie sich der Flash-Speicher im Bezug auf sein Hardwareumfeld integriert und wie er sich ins Softwareumfeld integriert. Das Umfeld, welches aus Hardware und Systemkomponenten besteht, hat einen maßgeblichen Einfluss auf die Performance der Speicher und muss deshalb bei der Testmethodik später beachtet werden.

2.5.1. Hardwareintegration

Im bisherigen Verlauf der Arbeit wurde der Aufbau einer Flash-Einzelzelle, die Verbindung dieser zu NOR- oder NAND-Strings, sowie die Bildung eines Speicher-Arrays aus Blocks und Pages erläutert. Es wurde damit die Struktur eines „Raw NANDs“ erarbeitet, d.h. eines NAND-Speicherchips ohne zusätzliche Hardwarekomponenten. Um einen NAND-Chip zu nutzen, muss dieser allerdings in ein entsprechendes Hardwareumfeld gebracht werden, damit dieser mit den anderen Komponenten im System kommunizieren kann und somit nutzbar wird.

Hardwareumfeld

Abbildung 2.26 zeigt eine vereinfachte beispielhafte Darstellung einer Hardwareintegration. Die linken Seite stellt den Flash-Speicher mit seinen verschiedenen Unterkomponenten dar. Der Gesamtspeicher besteht dabei aus den NAND-Chips, die die eigentlichen Speicherelemente darstellen. Damit der Speicher mit der „Außenwelt“ kommunizieren kann, benötigt

dieser einen speziellen Flash-Controller. Das genaue Aufgabenfeld des Controllers, das als *Flash Translation Layer (FTL)* bezeichnet werden kann, wird im nächsten Unterkapitel 2.5.2 erläutert. Die vom Controller auszuführende Firmware benötigt zwei zusätzliche Hardwarekomponenten. Zum einen muss persistenter Code abgelegt werden. Dafür wird ein ROM-Speicher verwendet. Zum anderen müssen sich zur Laufzeit ändernde Daten gespeichert werden. Hier kommt beispielsweise ein SRAM zum Einsatz. Die Kontrolleinheit besteht also neben dem Controller an sich, noch aus einem flüchtigen und einem nichtflüchtigen Speicher, mit allerdings vergleichbar niedrigen Speicherkapazitäten im Vergleich zu den NAND-Speicherchips. Sowohl Kontrolleinheit als auch einer oder mehrere Speicherchips befinden sich in einem Gehäuse und bilden somit den Gesamtchip.

Auf der rechten Seite der Abbildung befindet sich der sogenannte *System-on-a-Chip (SoC)* (alternativ CPU), mit dem der Speicherchip über einen Bus verbunden ist. Es handelt sich hierbei um die zentrale Recheneinheit des Systems. Der Unterschied zwischen SoC und CPU liegt im Allgemeinen darin, dass der SoC alle wichtigen Controller-Chips (GPU, USB-Controller, etc.) neben der CPU auf einem Silikon-Chip integriert hat und die CPU diese Controller-Chips ausgelagert hat und über Busse ansteuert. Dabei werden SoCs vor allem im eingebetteten Bereich verwendet, während CPUs aufgrund des größeren Platzangebots vor allem in Laptops oder PCs ihre Verwendung finden. Die Kommunikation zwischen Flash-Speicher und zentraler Recheneinheit findet dann auf einem Bus zwischen Speichercontroller und Flash-Schnittstelle der zentralen Recheneinheit statt.

Nun sind noch zwei verschiedene Schnittstellen zu betrachten, die die Kommunikation mit dem Speicher ermöglichen:

- Ausgangsschnittstelle
- NAND-Schnittstelle

Auf Abbildung 2.26 ist die Ausgangsschnittstelle in schwarz dargestellt. Es handelt sich dabei um die Schnittstelle, über die der Datentransfer zwischen dem Speicher-Controller und dem SOC (Flash-Schnittstelle) läuft. In diesem Abschnitt wird die Schnittstelle im Sinne eines physikalischen Datenbusses betrachtet. Es wird hier der grundsätzliche Leitungsaufbau und die grundsätzliche Art der Datenübertragung analysiert, wobei sich serielle von parallelen Schnittstellen unterscheiden lassen. Zusätzlich muss ein System im Controller mit einem speziellen Übertragungsprotokoll den Datentransfer regeln. Dies wird in Unterkapitel 2.6.2, bei den speziellen Speicher-Standards, weiter behandelt. Der Controller kann die vom SOC ankommenden Daten vorher verarbeiten, um diese optimiert an die tatsächlichen NAND-Bausteine weiterzuleiten.

Die NAND-Schnittstelle ist auf der Abbildung rot dargestellt. Sie regelt den Datentransfer zwischen NAND-Chips und Controller, d.h es fließen die vom Controller aufbereiteten Bits, die tatsächlich physikalisch in den Flash-Speicherarrays programmiert werden (siehe Unterkapitel 2.5.1) [MFL14]. Wichtig ist dabei festzustellen, dass beide Schnittstellen essentiellen Einfluss auf die Performance des Speichers haben. Dieser wird dadurch begründet, dass alle Daten beide Datenbusse überwinden müssen, um eine Lese-, Schreib- oder Löschoperation erfolgreich auszuführen.

Auf Abbildung 2.27 ist noch eine zweite Möglichkeit der Hardwareintegration auf der linken Seite dargestellt. Auf der rechten Seite wird am Beispiel eMMC(managed NAND) die vorher erläuterte Variante abgebildet, bei der sich der Controller und die Speicher-Chips in einem Gehäuse befinden. Bei der alternativen Variante auf der linken Seite besitzt der

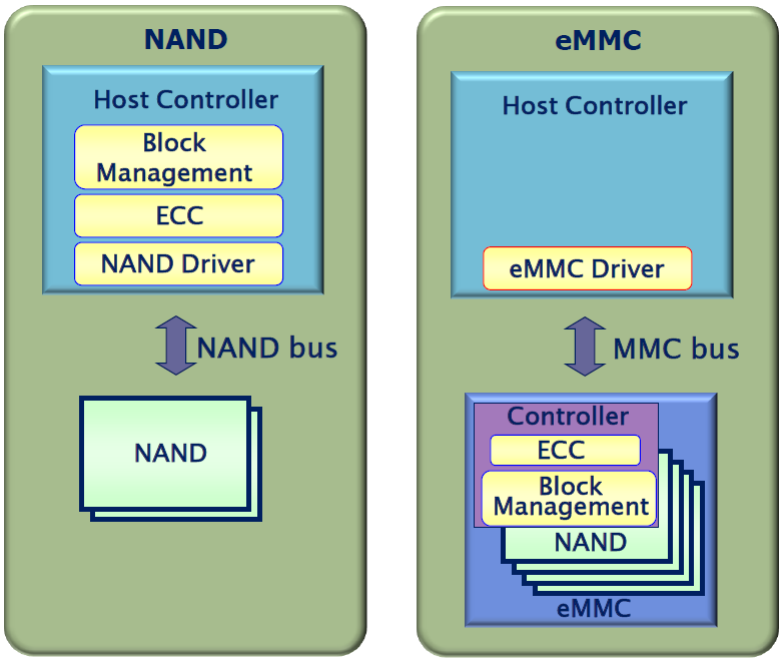


Abbildung 2.27.: Aufbau von unmanaged NAND und managed NAND(eMMC)

Flash-Speicherchip keinen extra Controller. Stattdessen werden alle wichtigen Management-Funktionen, wie z.B. der *Error Correcting Code (ECC)* zum Erkennen und Korrigieren von Übertragungsfehlern, vom Host Controller, also vom SoC übernommen. Hier spricht man vom Raw NAND oder Unmanaged NAND [KRKC11]. Für diese Arbeit ist jedoch nur die Variante des managed NAND von Bedeutung, bei der mit dem Speicher über eine einheitliche, standardisierte (Ausgangs-)Schnittstelle kommuniziert wird.

Serielle vs. Parallele Schnittstelle

Um die Datenübertragung von Ausgangs- und NAND-Schnittstelle besser verstehen zu können, muss festgestellt und verstanden werden, welche Art von physikalischer Schnittstelle zur Datenübertragung verwendet wird. Dieser Abschnitt setzt sich den zwei Varianten von Schnittstellen (parallele und serielle) auseinander, wobei hier nur auf die physikalische Datenübertragung eingegangen wird. Es wird also bei den übertragenen Bits nicht unterschieden, ob diese Teil der zu übertragenden Daten sind oder zu den Metadaten (Header) eines Protokolls gehören. Im Anschluss folgt eine Erläuterung der Speicherprotokolle (siehe Unterkapitel 2.6.2.

Geräte, die eine Schnittstelle zu einem Mikrocontroller besitzen werden allgemein Ein-/Ausgabeeinheiten oder IO-Ports genannt. Im Zusammenhang mit Speichern werden diese oft einfach Schnittstellen genannt. Zu differenzieren sind dabei grundsätzlich zwei verschiedene Arten von Schnittstellen, die aufgrund ihrer physikalischen Art und Weise Daten zu übertragen, unterschieden werden: parallele und serielle Schnittstellen [BU10].

Erstere werden generell durch folgende Eigenschaften charakterisiert [BU10]:

- Anzahl parallel übertragener Bits (Datenbreite)

- Übertragungsgeschwindigkeit
- Ein-/Ausgaberichtung

Die Datenbreite (Anzahl parallel übertragener Bits) ist in der Regel eine Zweierpotenz, also z.B. 4, 8, oder 16 Bit. Pro Leitung wird ein Bit parallel zu den anderen Leitungen übertragen. Aus diesem Grund wird bei größerer Bitzahl (z.B. 8) eine sehr große Anzahl an physikalischen Leitungen benötigt. Um nun beispielsweise 1 Byte zu übertragen wird innerhalb eines Clock-Pulses auf 8 Leitungen gleichzeitig jeweils ein Bit übertragen. Die Ein-/Ausgaberichtung ist hierbei zu vernachlässigen, da Speicher generell beide Richtungen für Lese- und Schreiboperationen nutzen. Aus diesem physikalischem Aufbau ergibt sich ein klarer Nachteil: hohe Kosten aufgrund einer hohen Anzahl an physikalischen Leitungen [BU10].

Bei seriellen Schnittstellen werden die Teile eines Datenpakets nacheinander über den Bus übertragen. Es werden also anstatt auf vielen Leitungen gleichzeitig auf einer oder sehr wenigen Leitungen nacheinander bitweise die Daten übertragen. Bei dieser Art von Schnittstelle werden somit physikalische Leitungen eingespart. Mit dieser Übertragungsweise kommen allerdings auch zusätzliche Aufwände hinzu. Da die Daten so stark aufgeteilt übertragen werden, muss die Synchronisation zwischen Sender und Empfänger viel stärker gesichert werden als bei der parallelen Schnittstelle. Es ist somit auch notwendig die Daten zu deserialisieren, also das Datenpaket wieder zusammenzusetzen. Hinzu kommt noch, dass bei seriellen Schnittstellen im Allgemeinen aufgrund des Deserialisierungsprozesses und der Synchronisierung ein anspruchsvolleres Protokoll notwendig wird [BU10].

Sowohl Sender als auch Empfänger besitzen einen Taktgenerator, der die selbe Taktfrequenz erzeugt. Diese laufen jedoch mit der Zeit nicht mehr exakt synchron und müssen deswegen nachsynchronisiert werden. Es gibt dabei zwei verschiedene Synchronisationsverfahren bei der seriellen Übertragung: asynchron und synchron. Bei der asynchronen Übertragung findet eine sogenannte Zeichensynchronisation statt. Ein Zeichen entspricht dabei einem Datenpaket von 5-8 Bits. Ist dieses übertragen wird der Takt beim Empfänger wieder angepasst. Das bedeutet, dass die zwei Taktfrequenzen nur sehr kurz synchron (genauer nicht mehr als eine halbe Taktperiode) sein müssen, nämlich nur während der Übertragung eines Zeichens. Allerdings muss dementsprechend oft synchronisiert werden und somit entsteht ein relativ hoher Synchronisationsaufwand, wobei die Bandbreite des Busses nur teilweise zur Nutzdatenübertragung verwendet wird. Bei der synchronen Übertragung findet eine Rahmensynchronisation statt, die die Synchronisationskosten verringert. Dies wird dadurch erzielt, dass seltener synchronisiert werden muss, da der Datenblock deutlich größer sein kann als das einzelne Zeichen bei der asynchronen Übertragung. Damit dies funktioniert, müssen dafür die Taktgeneratoren hochwertiger sein [BU10].

Diese Art von Aufteilung (parallel vs. seriell) zeigt die unterschiedliche Funktionsweise der Ausgangsschnittstellen und ist somit Grundlage für das Verständnis der Übertragungsprotokolle beziehungsweise Übertragungskosten (zusätzliche Leitungen, zusätzlicher Overhead) der Speichervarianten.

Schnittstelle zwischen NAND und Controller

Dieser Abschnitt fokussiert sich auf die NAND-Schnittstelle, also die Schnittstelle zwischen NAND-Chips und Speichercontroller (siehe Abbildung 2.26). Wichtig ist dies, da diese Schnittstelle die theoretische Performance (Übertragungsgeschwindigkeit) eines Speichers

definiert, da sie direkt auf die NAND-Zellen zugreift. Die Ausgangsschnittstelle muss dabei versuchen, die NAND-Schnittstelle maximal mit möglichst wenig erzeugten Latenzen auszureizen. Auf der anderen Seite kann bei einer zu schnellen Ausgangsschnittstelle die NAND-Schnittstelle zum Bottleneck werden. Die Daten können also nicht so schnell weggeschrieben werden, wie sie von der Ausgangsschnittstelle zur Verfügung gestellt werden.

Auch die NAND-Schnittstelle ist standardisiert. Es soll in dieser Arbeit nicht zu tief in auf diese Standards eingegangen werden, da sie ausschließlich vom Controller angesprochen werden können, und somit eine nicht veränderbare (nicht durch ein eigenes Programm ansprechbare) Stellschraube im Hinblick auf die Performance darstellen. Trotzdem müssen die Grundzüge verstanden werden, um die harte Performanceobergrenze zu verstehen, die durch diese Standards festgelegt werden.

Es gibt grundsätzlich zwei verschiedene Standards [Sam]:

- Toggle
- Open NAND Flash Interface (ONFI)

Toggle wird dabei von Samsung und Toshiba verwendet, während alle andere Speicherhersteller den ONFI-Standard implementieren. ONFI ist ein offener Standard, wohingegen Toggle nicht öffentlich einsehbar ist. Die genauen Unterschiede sollen in dieser Arbeit keine Rolle spielen. Es soll jedoch bemerkt werden, dass der jeweilige Standard eine maximale Übertragungsgeschwindigkeit vorgibt. Wichtig dabei ist, dass die aktuellsten Versionen beider Standards immer die selbe Übertragungsgeschwindigkeit garantieren (z.B. ONFI 2.0 und Toggle 1.0 mit jeweils 133 MB/s) und es nicht in den JEDEC-Standards vorgeschrieben wird, welcher Standard verwendet wird [JED15][JED][Sam].

Am Beispiel ONFI sollen nun die Performance-Grundlagen der Schnittstelle erläutert werden. Grundsätzlich ist wichtig zu verstehen, dass es mehrere ONFI-Schnittstellen pro Speicher gibt. Genauer, eine parallele Schnittstelle mit 8 Datenleitungen pro Speicherchip (NAND-Die). Deshalb wird im Standard die Übertragungsrate in Übertragungen pro Sekunde angegeben. Diese parallelen Schnittstellen , werden zusätzlich parallel nebeneinander betrieben. Das heißt bei zwei Schnittstellen wird die doppelte Übertragungsgeschwindigkeit erzielt. Anschaulich wird dies am aktuellen Standard ONFI 4.1. Der maximale Übertragungstakt beträgt 600 MHz, was bei der Nutzung von Double Data Rate (DDR) einen effektiven Übertragungstakt von 1200 MHZ ergibt. DDR bedeutet, dass sowohl bei der steigenden, als auch bei der fallenden Flanke eines Taktimpulses eine Übertragung stattfindet, also eine doppelte Übertragung pro Takt. Im Standard wird folglich eine Übertragung von 1200 MT/s (Mega Transitions / Second) angegeben. Da ein ONFI-Bus ein Byte pro Takt übertragen kann, ergeben sich 1200 MHz * 8 Bit = 1200 MB/s für einen Bus (2400 MB/s für zwei Busse). Besitz ein konkreter Speicher also nur einen ONFI-Bus, so kann die Übertragungsrate des Gesamtspeichers nie den Wert von 1200 MB/s übersteigen [Int17].

Abbildung 2.28 zeigt zusammenfassend auf, welche Schnittstellen vorhanden sind und wie sich die rein physikalischen Übertragungsgeschwindigkeiten zusammensetzen. Der Bus zwischen SOC und Controller bildet die Ausgangsschnittstelle (z.B. UFS), dessen maximale Übertragungsgeschwindigkeit (hier 2400 MB/s) sich aus den einzelnen NAND-Schnittstellen zwischen Controller und NAND-Chip (jeweils 1200 MB/s) zusammensetzt. Genaue Datenraten der speziellen Speicherstandards werden in Unterkapitel 2.6.1 erläutert. Der errechnete Wert ist allerdings rein theoretisch und kann in der Realität nicht erzielt werden, da zusätzliche Latenzen z.B durch die Serialisierung und Deserialisierung von Daten zwischen

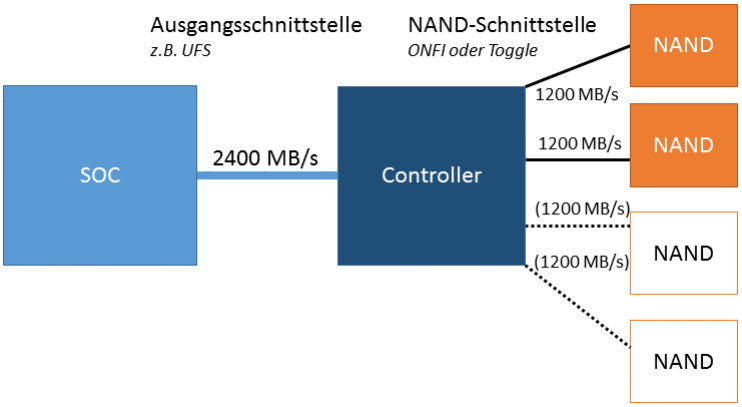


Abbildung 2.28.: Überblick zu Schnittstellen im Managed NAND

den Schnittstellen dazu führen, dass der physikalische NAND-Speicher nicht voll ausgelastet werden kann. Es ist aus diesem Wert also folglich keine Mindestperformance des Gesamtspeichers abzuleiten und zeigt somit die Wichtigkeit einer praktischen Performance-Messung.

2.5.2. Systemische Integration

Zum weiteren Verständnis der Systemintegration steht die Definition des Begriffs *FTL* noch aus. In Unterkapitel 2.3 wurde erläutert, wie sich fortschreitende P/E-Cycles negativ auf das Verhalten des Flash-Speichers auswirken. Außerdem wird vor der Schreibaktion eines kleinen Bereichs erst ein großer Bereich gelöscht. Dieser Vorgang kann ineffizient werden, wenn beispielsweise nicht vollkommen gefüllte Blöcke gelöscht werden. Aus diesen und weiteren Gründen ist ein Softwaremodul notwendig, das diese Probleme möglichst intelligent löst und somit eine effiziente Nutzung des NAND-Flashs ermöglicht. Genannt wird dieses Softwaremodul Flash Transition Layer und kann dabei vom Host oder vom Speicher selbst übernommen werden [KRKC11].

Die zwei Hardwareintegrationsvarianten Managed und Unmanaged NAND können auch etwas abstrakter auf Systemebene betrachtet werden. Auf Abbildung 2.29 sind diese beiden Architekturen im Bezug auf das gesamte System, von Applikation bis hin zum Raw Flash-Speicher, dargestellt. Auf Darstellung a) übernimmt das Betriebssystem die Aufgabe, das File System zu hosten und zusätzlich übernimmt es die Implementierung der FTL. Auf Darstellung b) verschiebt sich die Implementierung der FTL in den Aufgabenbereich des Speichers und würde hier von einem Controller bearbeitet [MFL14].

Bevor noch genauer auf die Aufgaben der FTL eingegangen wird, soll noch erläutert werden, wie die FTL in die Flash-Speicherarchitektur integriert wird. Außerdem sollen noch weitere grundliegende Eigenschaften zum Flash-Speicher erläutert werden. Die Abbildung zeigt 2.8 die grundsätzliche NAND-Array-Architektur und muss nun noch durch Abbildung 2.30 ergänzt werden, um die Speicherbereiche etwas genauer zu verstehen. Dargestellt ist hier ein sogenanntes „Flash Package“ also im Prinzip ein Raw NAND-Speicherchip. Dieser besteht aus einem Die(Silikonchip) auf dem mehrere Planes vorhanden sind. Planes sind wiederum lediglich Ansammlungen einer festen Anzahl an Blocks. Auf der rechten Seite der Abbildung ist zu erkennen, wie sich der einzelne Block aus verschiedenen Pages zusammensetzt. Wie schon mehrmals erwähnt, wird auf Blockebene gelöscht und auf Pageebene gelesen

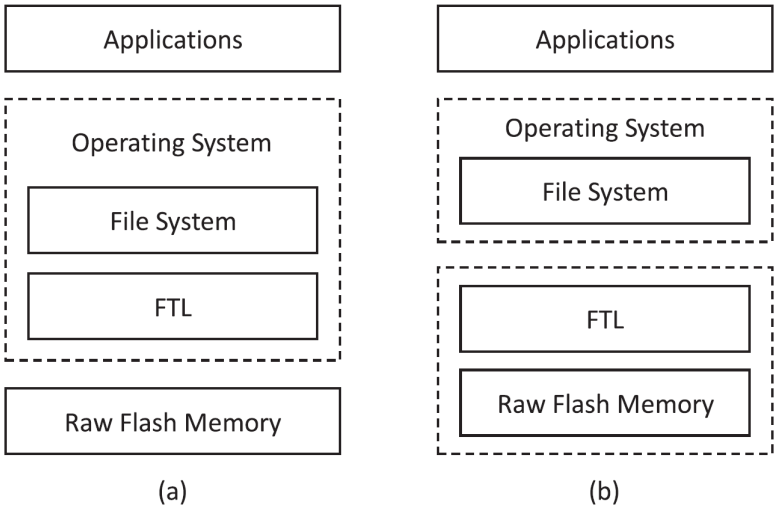


Abbildung 2.29.: FTL-Architekturen: a) Betriebssystem mit FTL-Funktionen b) FTL in der Firmware des Speichers integriert [MFL14]

und geschrieben. Die wichtige neue Information, die hier auf dieser Grafik jedoch entnommen werden sollte, ist der Aufbau einer Page. Diese wird in zwei Bereiche aufgeteilt: ein großer Bereich (Data Area) für die Speicherung von Daten und ein kleiner Bereich (Spare Area) für das Speichern von Metadaten, die für die Organisation des Speichers durch die FTL genutzt werden. Diese Metadaten werden *Out-Of-Band (OOB)*-Daten genannt und beinhalten die *Logical Page Number (LPN)*, *Logical Block Number (LBN)*, ECC sowie verschiedene Zustands-Flags (State Flags) der Page [MFL14].

Small und Large Block Flash-Speicher

Ein NAND-Flash besitzt eine feste Anzahl an Blöcken, wobei jeder Block aus 32 oder 64 Pages besteht (abhängig vom Produkt). NAND lässt sich hinsichtlich der Page-und Blockgröße noch weiter klassifizieren:

- Small Block Flash-Speicher
- Large Block Flash-Speicher

Gängige Filesysteme besitzen eine Sektorgröße von 512 Bytes. Bei einem Small Block Speicher besitzt der Datenbereich der Page genau diese Größe und zusätzlich 16 Bytes Spare Area. Bei einem Large Block Speicher jedoch hat der Datenbereich einer Page die Größe von 2 KB und die Spare Area 64 Bytes. In diesem Fall werden somit 4 Sektoren innerhalb einer einzigen Page untergebracht. Diese Information ist wichtig, da bestimmte NAND-Features für beide Speicher unterschiedlich implementiert werden [KRKC11].

Um die Prioritäten der FTL zu verstehen, ist die Information wichtig, wie lange die verschiedenen Flash-Operationen dauern. Bei einem konkreten Flash-Speicher der Firma Samsung ergeben sich folgende Werte [KRKC11]:

- Leseoperation: 15 μs

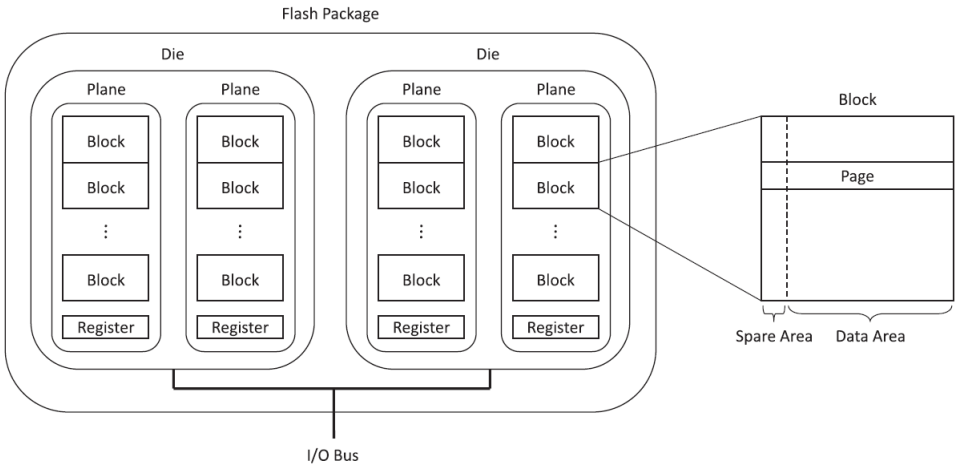


Abbildung 2.30.: Organisation eines Flash Packages[MFL14]

- Schreiboperation: 200 μs
- Löschoperation: 2 ms

Diese Werte sollen nicht die Leistungsfähigkeit dieses Speichers zeigen, sondern das Verhältnis der Zeiten für verschiedene Operationen zueinander. Die Löschoperation ist klar langsamer als die beiden anderen. Aus diesem Grund ist es für die FTL wichtig die Anzahl, auch aus Performancegründen, möglichst niedrig zu halten. Das typische Problem des NAND-Flashs ist der sogenannte „erase-before-write“, das blockweise Löschen, bevor auf Pageebene geschrieben werden kann. Um diesem Problem entgegenzuwirken, wurde das Partial Programming implementiert. Hierbei kann nicht in der Größeneinheit Page sondern sogar in der Einheit Byte innerhalb einer Page geschrieben werden. Dabei kann auf eine Page wieder zugegriffen werden, die schon teilweise beschrieben wurde und es können Daten im noch freien Bereich hinzugefügt werden. Deshalb kann die FTL in der Spare Area einer Page, auch nachdem die Data Area schon vollgeschrieben ist, noch die Metadaten für diese Page schreiben, ohne diese löschen zu müssen. Allerdings ist Partial Programming nicht unbegrenzt möglich, sondern wird durch die *Number of Partial Programming (NOP)* festgelegt, welche sich bei Small Block und Large Block Speichern unterscheidet. Bei Small Block Speichern ist die NOP für die Data Area 1 bis 2, für die Spare Area allerdings 3 bis 4. Die Spare Area kann demnach öfter in kleinen Einheiten programmiert werden. Bei einem Large Block Speicher liegt die NOP für beide Bereiche bei 1 bis 2. Das bedeutet, dass diese Speicher weniger flexibel im Bezug auf die FTL sind, da die Metadaten in der Spare Area nicht flexibel genutzt werden können [KRKC11].

Neben der NOP stellt die „Sequential Write Restriction“ eine weitere Einschränkung für den Schreibvorgang des Flash-Speichers dar, welche jedoch nur auf Large Block Speicher zutrifft. Diese Einschränkung legt fest, dass die Pages eines Blocks sequentiell von der LSB-Page zur MSB-Page geschrieben werden müssen. Auf der linken Seite der Abbildung 2.31 wird die Einschränkung für Large Block Speicher gezeigt. Hier müssen die Pages von 0 bis 63 nacheinander geschrieben werden. Wobei hingegen auf der rechten Seite (Small Block Speicher) die Pages in einer beliebigen Reihenfolge geschrieben werden können. Der Grund

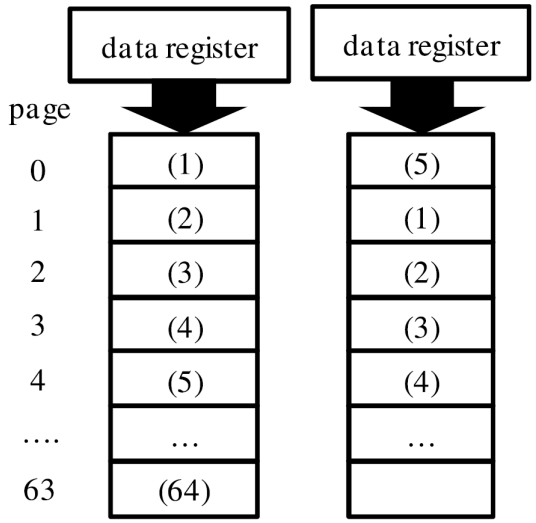


Abbildung 2.31.: Sequentielle Schreibrestriktion [KRKC11]

für diese Einschränkung liegt darin, dass es bei Large Block Speichern sonst zu Program Disturbs (siehe Kapitel2.3) aufgrund der größeren Struktur kommen würde [KRKC11].

Ein Vorteil für Large Block Speicher ist die Möglichkeit des sogenannten „Random Data Out/In“, welches für Small Block Speicher nicht möglich ist. Im Prinzip handelt es sich hierbei um die Fähigkeit, innerhalb einer Page die 4 Sektoren wahlfrei auszulesen (Random Data Out) oder zu schreiben (Random Data In). Dabei ist der Zugriff auf den einzelnen Sektor schneller als beim Small Block Speicher, der eine sequentielle Operation für die einzelnen Pages ausführt. Die Performance wird durch Random Data Out/In um das Vierfache gesteigert, verglichen mit einem sequentiellen Vorgang [KRKC11].

Flash Transition Layer

Weniger als 4% des Speichervolumens sind vom Verbraucher unsichtbar und als Nutzungsbereich der FTL-Algorithmen reserviert. Dieser Bereich setzt sich aus den Spare Areas der einzelnen Pages und den Spare Blocks zusammen. Spare Blocks sind zusätzliche Blocks, die für die FTL verwendet werden. Weil verschiedene Hersteller verschiedene FTLs verwenden, unterscheidet sich auch die Anzahl der verwendeten Spare Blocks. Diese kann auch einen Einfluss auf die Performance haben. Je mehr Spare Blocks die FTL zur Verfügung hat, desto höher ist die Pufferkapazität des Speichers. Dies wird klarer, wenn die Mapping-Algorithmen verstanden werden. Das Host-System sieht den Flash-Speicher ähnlich wie eine Festplatte. Das Filesystem fordert Lese-und Schreiboperationen an den Flash-Speicher an und verwendet dabei logische Adressen. Die physischen Adressen, die den logischen Adressen im Speicher entsprechen, werden von Mapping-Algorithmen der FTL festgelegt. Dabei findet das Mapping von logischen auf physische Adressen auf Sektor- oder Blockebene statt. Beim Überschreiben des Inhalts eines Sektors wird die physikalische Adresse des alten ungültigen Sektors auf einen neuen unbeschriebenen Bereich umgeleitet. So wird eine zeitaufwändige Löschaktion gespart. Für das Mapping befindet sich eine Mapping-Tabelle im SRAM. Die jeweiligen Adressen werden dann angepasst und der alte Block kann später gelöscht werden. Der beschriebene Vorgang wird auch „Basic Mapping“ genannt. Mehr Spare Blocks bedeuten

nun, dass öfter statt zu Löschen einfach auf einen leeren Block umgemappt werden kann. Durch die gesparten Löschzeiten erhöht sich dadurch die Performance [KRKC11].

Neben dem naiven und intuitiven Basic Mapping gibt es auch fortschrittlichere FTL-Algorithmen, welche zum einen die Performance und zum anderen die Haltbarkeit erhöhen sollen. Diese werden vom Basic Mapping abgeleitet und durch zusätzliche Eigenschaften, wie dem Partial Programming ergänzt. Beide Arten von Algorithmen können kombiniert werden und überschneiden sich auch teilweise in ihrer Wirkung.

Um die Performance zu erhöhen, muss grundsätzlich die Anzahl an Löschaktionen reduziert werden. Des Weiteren kann in verschiedenen Granularitäten gemapped werden, nämlich in Blocks, Pages und Sektoren. Je kleiner dabei die Einheit des Mappings, desto besser wird die Performance, da genauer auf die gewünschten Daten zugegriffen werden kann. Eine genauere Tabelle im RAM ist allerdings verbunden mit zusätzlichen RAM-Kosten, da mehr Einträge notwendig sind. Das bedeutet, es muss ein Kompromiss zwischen höherer Performance und gleichzeitig relativ niedrigen RAM-Kosten gefunden werden. Deshalb müssen die Basic Mapping-Algorithmen so angepasst werden, dass sie die Spare Blocks effizient nutzen, um wiederum die Gesamtzahl an Schreib-, Lese- und Löschoperationen zu reduzieren. Ein möglicher Ansatz wäre zum Beispiel das „Log-Sector Scheme“. Hierbei soll die Anzahl an Löschoperationen reduziert werden, die durch die Wiederholung von identischen Logical Sector Numbers (LSNs) verursacht wird. Das Konzept besteht darin, dass jeder Sektor einen oder mehr Sektoren (Log Sector) als Puffer zugewiesen bekommt. Soll ein Sektor überschrieben werden, so wird dieser nicht inklusive seines Blocks gelöscht und neu beschrieben, sondern die Daten werden in den Puffer geschrieben. Es wird folglich kein Sektor mit der selben LSN und dem geupdateten Inhalt erzeugt, sondern der neue Inhalt in einem „Reservesektor“ gespeichert. Der Puffer kann sich dabei auch in einem anderen Block befinden [KRKC11].

Die Steigerung der Haltbarkeit kann auf zwei verschiedene Arten gelöst werden. Die erste Möglichkeit ist das sogenannte „Wear Leveling“, bei dem die Abnutzung möglichst gleichmäßig auf den ganzen Speicher verteilt werden soll. Dieser Algorithmus arbeitet auf Blockebene und Basis dafür ist die *Erase Count Number (ECN)* jedes Blockes, welche die Metainformation beinhaltet, wie oft dieser bereits gelöscht wurde. Übersteigt diese Anzahl einen gewissen wert so wird er als „Hot Block“ erkannt, der schon einen gewissen Grad an Abnutzung erfahren hat. Daraufhin findet ein Austausch mit einem im Vergleich ungenutzten „Cold Block“ statt. Dafür wird dessen logische Adresse auf eine andere physikalische Adresse gemapped. Für diesen Algorithmus ist ein periodisches Monitoring notwendig, denn es muss möglichst früh erkannt werden, wenn es zu einer ungleichmäßigen Abnutzung kommt. Einhergehend ist ein zusätzlicher Aufwand (zusätzliche periodische Leseaktionen), der die Performance negativ beeinflussen kann. Für das Wear Leveling wurde noch eine Unterstützungsmaßnahme entwickelt. Generell werden die Informationen für die FTL in der Spare Area und im SRAM abgelegt. Im Falle der Initialisierung eines Speichers, beispielsweise beim Aufstarten seines Hostsystems, werden die Block- und Sektorinformationen von der Spare Area ausgelesen und in den SRAM geladen. Das Wear Leveling benötigt z.B. die ECN. Diese Information jedoch von jedem Block auszulesen, ist ein kostenintensiver Prozess. Als Lösung kann ein sogenannter *Erase Count Block (ECB)* genutzt werden. Hier werden alle ECNs der einzelnen Blocks gemeinsam in einem Block abgespeichert. Dadurch können alle stark genutzten oder auch defekte Blocks leicht durch das Lesen eines einzelnen auf einen Blick erkannt werden. Wenn ein Block sein P/E-Cycle-Limit erreicht hat, kann er nicht mehr genutzt werden. Dieser Block wird dann mit einem „FFFFFF“ markiert [KRKC11].

Die zweite Möglichkeit der Haltbarkeitssteigerung stellt die „Cleaning Policy“, auch „Garbage Collection“ genannt, dar. Wird ein Sektor überschrieben, so wird der neue Inhalt in einen neuen Sektor geschrieben und die LSN auf diesen übertragen. Dadurch bleibt ein alter Sektor mit invaliden Daten übrig, der die selbe LSN, wie der upgedatete Sektor hat. Dieser alte Sektor wird als „dirty“ bezeichnet. Er muss möglichst schnell gelöscht werden, da er sonst unnötig Speicherplatz verbraucht. Falls dies nicht gemacht wird, füllen sich die Blöcke mit vielen dirty Sektoren schnell und verursachen dadurch eine wiederkehrende Löschaktion. Der Garbage Collector entfernt den dirty Block, indem er die validen Daten in einen neu allokierten Block kopiert und den alten löscht. Es entsteht demnach ein neuer Block der außer den validen Daten leer ist. Somit wurden dirty Sektoren eliminiert. Für die Implementierung dieses Algorithmus muss ein Monitoring auf Sektorebene betrieben werden, da die „Dirtyness“ eines Blocks durch die Anzahl an dirty Sektoren des Blocks festgelegt wird. Weil das Monitoring auf Sektorebene im Gegensatz zum Wear Leveling (Blockebene) betrieben wird, ist dieser Algorithmus im Vergleich teurer [KRKC11].

Zusammenfassend zeigt sich zum Thema Systemintegration die Erkenntnis, dass sie neben dem physikalischen Umfeld eine tragende Rolle im Hinblick auf die Performance spielt. Man könnte sagen, dass das Hardware Umfeld die theoretische Grenze setzt, wie schnell Bits physikalisch übertragen werden können. Das zugrundeliegende System hingegen legt die tatsächlichen Verhältnisse innerhalb dieses theoretischen Rahmens fest, da die Aufgabe der Bits auf die Kalkulation der Performance Einfluss nimmt (nicht alle übertragenen Bits sind Daten). Es werden grundsätzlich zwei Performance beeinflussende Phänomene erkannt. Zum einen erzeugt das Betriebssystem in Verbindung mit der Applikation, die den Speicher nutzt, zusätzlichen Aufwand in Form von Wartezeiten (Latenzen). Das kann somit zu einer verminderten Performance führen. Zum anderen spielt das Übertragungsprotokoll eine Rolle. Hier ist wichtig, wie groß der Zusatzaufwand (z.B. Länge der Übertragungsheader) im Protokoll ist, um Information zu übertragen. Daraus kann geschlussfolgert werden, dass diese Zusatzaufwände so niedrig, wie möglich gehalten werden müssen, um eine hohe Performance zu erzielen. Wird die Performance gemessen, so müssen diese Faktoren (soweit möglich) gezielt definiert und aufgezeichnet werden, um die Vergleichbarkeit zu waren. Kapitel 3 greift dieses Thema noch einmal auf.

Die zweite Erkenntnis liegt in der Tatsache, dass ein beachtlicher Mehraufwand in Form von zusätzlichen Schreib- und Löschoperationen durch die FTL erzeugt wird. In Kapitel 2.3 wurde bereits die Wichtigkeit der Zuverlässigkeit aufgrund der speziellen Flash-Charakteristika erläutert. Die FTL greift nun mit intelligentem Mapping und Garbage Collection diese Problematik auf und erzeugt gleichzeitig die erwähnten zusätzlichen Operationen. Da diese Mechanismen im Controller stattfinden und somit nicht vom Nutzer beeinflusst werden können, stellen sie einen weiteren Performance-Faktor dar. Dessen Einfluss lässt sich nur durch praktische Tests ermitteln und kann gerade bei älteren Speichern (viele P/E-Cycles) zu Performance-Verminderung führen.

2.6. Flash-Standardanalyse

Im Kapitel 2.5 wurden die NAND- und Ausgangsschnittstelle sowie die generelle systemische Integration eines Flash-Speichers erläutert. Nun soll an den zwei konkreten JEDEC-Standards eMMC und UFS gezeigt werden, welche dieser Charakteristika in den Standards festgelegt sind. Aus den verwendeten Schnittstellen sollen sich Performanceobergrenzen ab-

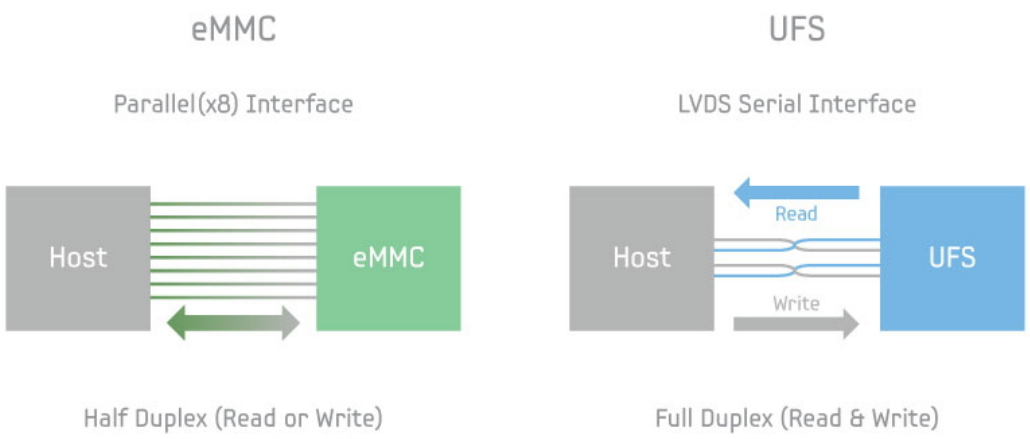


Abbildung 2.32.: Parallele (eMMC) und serielle (UFS) Schnittstellen

leiten lassen und das Systemmodell der jeweiligen Schnittstelle soll Auskunft über die standardisierte Konfigurierbarkeit und die Diagnosekapazitäten des Speichers geben.

2.6.1. Verwendete Speicherschnittstellen

Zum Verständnis eines Flash-Speicher (managed NAND) in seinem Gesamtpaket müssen die beiden Schnittstellen Ausgangs- und NAND-Schnittstelle separat betrachtet werden. Die NAND-Schnittstelle kann nur über den Controller angesprochen werden und ist für den Nutzer/Anwendungsprogrammierer eine Art Blackbox, da auch der Controller eine Blackbox nach Außen darstellt. Nur Funktionen, die in der Ausgangsschnittstelle (z.B. UFS) spezifiziert sind, können indirekt den Controller ansprechen. Hinzukommt, dass nicht spezifiziert ist, wie viele NAND-Schnittstellen ein Speicher zur Verfügung stellen muss und diese Information wird auch nicht von den Herstellern zur Verfügung gestellt. Auch welche Standard (ONFI oder Toggle), ist nicht spezifiziert [JED][JED15]. Es zeigt sich, dass es so viele NAND-Schnittstellen geben muss, dass sie die für die Ausgangsschnittstelle definierte Datenrate, ermöglichen. Im weiteren Verlauf ist deswegen nur noch die Ausgangsschnittstelle gemeint, wenn von der Speicherschnittstelle gesprochen wird, es sei denn es wird speziell angemerkt, dass dies nicht der Fall ist.

Abbildung 2.32 zeigt den grundsätzlichen Aufbau beider Schnittstellen anhand von einem UFS-Speicher auf der rechten Seite und einem eMMC auf der linken Seite. Hier kann der grundlegende Unterschied erkannt werden: Die parallele Schnittstelle(eMMC benötigt deutlich mehr physikalische Leitungen als die serielle Schnittstelle(UFS). Die Schnittstelle des eMMC besitzt hier 8 Datenleitungen und kommuniziert mit Halb-Duplex, d.h es ist nur Schreiben oder Lesen zu einem bestimmten Zeitpunkt möglich. Der UFS hingegen kann Schreib- und Leseoperationen gleichzeitig ausführen und kommuniziert somit mit Voll-Duplex. Eine Leitung besteht hier aus einem Paar an differentiellen Leitungen, wobei eine Gesamtleitung für Lesen und eine für Schreiben vorhanden ist [3].

Auf Abbildung 2.33 ist die Schnittstelle des eMMC etwas detaillierter dargestellt. Wie

[3] https://news.samsung.com/global/emmc-to-ufs-how-nand-memory-for-mobile-products-is-evolving, Stand 10.04.2018

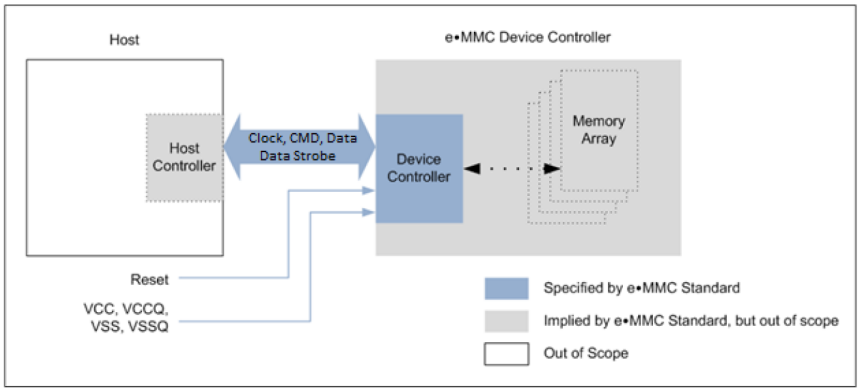


Abbildung 2.33.: Schnittstellenaufbau eMMC[JED15]

schon erwähnt, handelt es sich hierbei um eine parallele Schnittstelle mit bis zu 8 parallelen Datenleitungen. In dem eMMC-Hauptstandard 5.1 der JEDEC wird die Schnittstelle sowie das Verhalten des Speicher-Controllers definiert. Das Dokument spezifiziert hier feste physikalische Leitungen und Signale auf der Seite der Schnittstelle, jedoch nur das Verhalten des Speicher-Controllers, also keine Angaben zur Hardware des Controllers. Die Schnittstelle beinhaltet vier verschiedene Kommunikationsleitungen [JED15]:

- CLK: Taktleitung
- Data Strobe: Ein Signal, das vom Speicher erzeugt wird und die Frequenz des CLK-Signals wiedergibt
- CMD: Bidirektionaler Kommandokanal für die Speicherinitialisierung und die Übertragung von Kommandos
- DAT0-DAT7: Parallele Datenleitungen

Die Busbreite kann konfiguriert werden, sodass entweder 1, 4 oder 8 Datenleitungen für den Datentransfer genutzt werden können. Abbildung 2.34 zeigt die spezifizierten Übertragungsmodi mit den jeweiligen maximalen Übertragungsgeschwindigkeiten. Diese berechnet sich hier ganz simpel: Frequenz * Busbreite. Also eine Busbreite von beispielsweise 8 Bit, bei einer Frequenz von 26 MHz ergibt eine maximale Übertragungsgeschwindigkeit von 8 Bit * 26 MHz = 26 MB/s.

Die Schnittstelle kann in fünf verschiedenen Performancemodi genutzt werden, wobei die zwei hochperformantesten etwas genauer beleuchtet werden sollen: HS200 (max. 200 MB/s) und HS400 (max. 400 MB/s). Auf physikalischer Ebene unterscheiden diese sich nur durch das Nutzen des Data Strobe bei HS400. Bei HS200 werden sowohl Lese- als auch Schreibbefehle durch das CLK-Signal des Hosts getaktet. Bei HS400 hingegen, wird das CLK-Signal beim Lesevorgang auf Speicherseite dupliziert und in die Datenrichtung verwendet. Dieses duplizierte Taktsignal heißt Data Strobe. Es werden also hier Takt- und Datensignale in der selben Richtung verwendet: vom Speicher zum Host. Die erhöhte Übertragungsperformance bei HS400 wird dadurch erzielt, dass anstatt nur bei der ansteigenden Taktflanke nun auch die fallende Flanke als Übertragungstakt verwendet werden, also der Übertragungstakt verdoppelt wird. Zu beachten ist, dass der Performancezuwachs nur für den Lesevorgang gilt,

Mode Name	Data Rate	I/O Voltage	Bus Width	Frequency	Max Data Transfer (implies x8 bus width)
Backwards Compatibility with legacy MMC card	Single	3 V/1.8 V/ 1.2 V	1, 4, 8	0-26 MHz	26 MB/s
High Speed SDR	Single	3 V/1.8 V/ 1.2 V	1,4, 8	0-52 MHz	52 MB/s
High Speed DDR	Dual	3 V/1.8 V/ 1.2 V	4, 8	0-52 MHz	104 MB/s
HS200	Single	1.8 V/1.2 V	4, 8	0-200 MHz	200 MB/s
HS400	Dual	1.8 V/1.2 V	8	0-200 MHz	400 MB/s

Abbildung 2.34.: Übertragungsmodi eMMC[JED15]

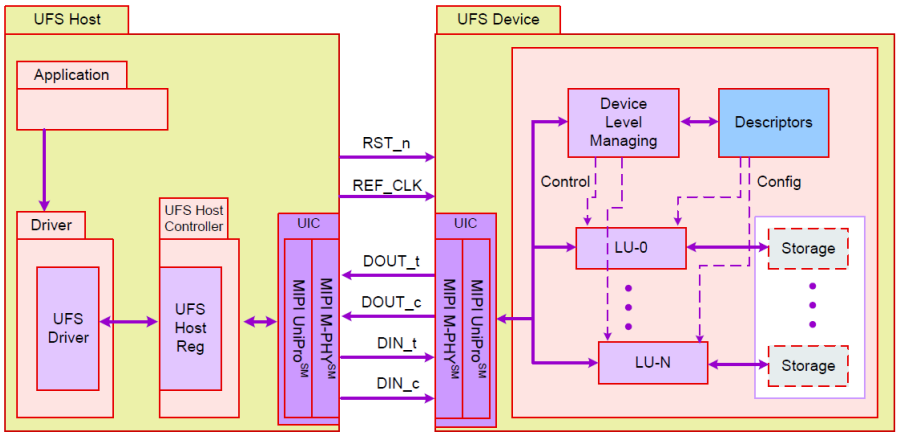


Abbildung 2.35.: Schnittstellenaufbau UFS[JED]

denn Data Strobe wird nicht für das Schreiben oder Löschen genutzt. Neben Kommunikationsleitungen gibt es noch diverse Versorgungsspannungen (VCC, VCCQ, VSS, VSSQ) sowie ein Hardware-Reset-Signal (Reset) [JED15].

Auf Abbildung 2.35 ist der Schnittstellenaufbau eines UFS-Speichers gezeigt. Wie bereits erwähnt, handelt es sich hierbei um eine serielle Schnittstelle. In diesem Abschnitt sollen nur die Signalleitungen zwischen Host und Speicher betrachtet werden. Die Schnittstelle beinhaltet fest konfigurierte sogenannte „Lanes“, die entweder fest für das Senden oder das Erhalten von Daten (Upstream, Downstream) genutzt werden. Auf der Abbildung ist jeweils eine differentielle Lane für beide Richtungen gezeigt, was der UFS-Standardkonfiguration entspricht. Es können jedoch auch jeweils zwei Lanes genutzt werden, solange die Anzahl in beide Richtungen gleich ist. Eine Lane besteht aus einem differentiellen Leitungspaar. Bei der differentiellen Übertragung wird das selbe Signal (gleiche Daten) auf zwei Leitungen übertragen, jedoch werden die Signale gegenphasig gesendet. Das bedeutet, wenn die Spannungsspitze bei einem Signal positiv ist, so ist diese bei dem anderen negativ. Die Signale werden dann vom Empfänger wieder zu einem zusammenaddiert. Beim UFS besteht die Sendeleitung aus den differentiellen Leitungen DOUT_t (t: True für „echtes“ Signal) und DOUT_c (c: complement für gegenphasiges Komplementärsignal). Analog dazu sind

HS-GEAR	Rate A-series	Rate B-series		Rate A-series (from [MIPI-M-PHY])	Rate B-series[4] (from [MIPI-M-PHY])	Unit
	f_{ref}	f_{ref}		f_{ref}	f_{ref}	
	19.2 / 38.4 / 26 / 52	19.2 / 38.4	26 / 52	19.2 / 38.4 / 26 / 52		MHz
HS-GEAR1	1248[2]	1459.2	1456.0	1248	1457.6	Mbps
HS-GEAR2	2496	2918.4	2912.0	2496	2915.2	Mbps
HS-GEAR3[3]	4992	5836.8	5824.0	4992	5830.4	Mbps

Abbildung 2.36.: Übertragungsmodi UFS[JED]

die Empfangsleitungen DIN_t und DIN_c. Außerdem besitzt die Schnittstelle ein Taktsignal (REF_CLK) und ein Hardware Reset-Signal (RST_n). Das Taktsignal ist relativ langsam (bis zu 52 MHz). Die hohen Datenraten von bis zu 5830 Mbps werden dadurch erzielt, dass der Referenztakt auf dem Speicher hoch getaktet wird und somit Takte im GHz-Bereich entstehen [JED].

Abbildung 2.36 zeigt die spezifizierten Übertragungsmodi für UFS-Speicher pro Lane. Die gezeigte Tabelle ist etwas komplexer als bei eMMC und bedarf deshalb einer genaueren Erklärung. Die Übertragungsmodi werden „High Speed Gears“ genannt, wobei drei verschiedene ausgeführt werden können. Die angegeben Frequenzen sind Referenzfrequenzen, d.h. sie werden nur als Kontrolltakt vom Host zum Speicher verwendet und intern von sogenannten „PLL-Multipliers (Frequenzmultiplikatoren)“ im Speicher auf den gewünschten Takt hoch getaktet. Das bedeutet, jeder HS-Gear kann mit jeder Referenzfrequenz betrieben werden. Zusätzlich muss noch zwischen A-Rate und B-Rate unterschieden werden. Bei der A-Rate werden die angegebenen Referenztakte verwendet, wohingegen bei der B-Rate die Referenzfrequenz um bis zu 2000 Millionstel (ppm) abweichen darf. Diese Abweichung wird angeboten, damit Modulationseinflüsse durch andere Hardware auf der Platine vermieden werden können. Um die Datenraten mit eMMC zu vergleichen, müssen nun noch die angegebenen Mbps auf MB/s umgerechnet werden. Das ergibt bei HS-Gear3 und A-Rate 4992/8 Bit = 624 MB/s für eine Lane. Diese Größe muss jedoch noch einmal relativiert werden, da beim UFS eine 8b10b-Kodierung verwendet wird. Das bedeutet von 10 Bits, die über die Schnittstelle übertragen werden, sind nur 8 Bits tatsächliche Datenbits. Dieser Leitungscode wird aus signaltechnischen Gründen verwendet, auf die in dieser Arbeit nicht genauer eingegangen werden soll. Wichtig ist, dass die berechnete Datenrate noch einmal um 20 Prozent reduziert wird. Die Vollständige Übertragungsgeschwindigkeit wird also wie folgt am vorherigen Beispiel mit 2 Lanes berechnet [JED]:

$$4992/8 \text{ (Bits pro Byte)} * 80\% \text{ (8b/10b Kodierung)} * 2 \text{ (Lanes)} = 998\text{MB/s} \quad (2.1)$$

Zusammenfassend kann zu den verwendeten Schnittstellen gesagt werden, dass die jeweiligen Übertragungsmodi die tatsächlich maximale Performance bei der Übertragungsgeschwindigkeit festlegen. Diese sind somit immer bewusst zu wählen, wenn die Performance der Speicher getestet wird. Es ist dennoch zu beachten, dass es sich hier bei beiden Speichertypen (eMMC, UFS) um die Geschwindigkeit einer reinen Bitübertragung handelt, nicht zu verwechseln mit Lese- oder Schreibgeschwindigkeit. Es werden demnach keine Kommandos,

Protokolle oder anderer systemischer Übertragungsaufwand mitberechnet. Dies zeigt, dass die Performance bei applikativen Anwendungen extrem komplex zu berechnen ist und somit praktisch für gewisse Szenarien getestet werden muss, wenn ermittelt werden soll, ob eine gewisse Performance in der eigenen Applikation gehalten werden kann.

2.6.2. Kommunikationsmodell

Dieser Abschnitt spiegelt die grundlegende Kommunikation zwischen Schnittstelle und Host für den jeweiligen Speicherstandard wieder. Die Intention liegt darin, ein Verständnis dafür zu schaffen, wie die Operationen (Schreiben, Lesen, Löschen) an den Speicher übermittelt werden. Außerdem soll gezeigt werden, auf welche Art und Weise der Speicher konfiguriert und diagnostiziert werden kann.

eMMC

Die Kommunikation findet zwischen einem Host und einem Device über ein nachrichtenbasiertes eMMC-Protokoll statt. Jede Nachricht besteht dabei aus einer der folgenden „Token“:

- Kommando: Ein Kommando-Token beginnt eine Operation und wird vom Host an das Device gesendet.
- Antwort: Ein Antwort-Token wird vom Device an den Host als Antwort auf das vorher erhaltene Kommando gesendet.
- Daten: Daten können in beide Richtungen gesendet werden (Lesen, Schreiben) und können dabei auf eine, vier oder acht Datenleitungen übertragen werden.

Eine Busoperation ist als eine Folge von Kommando und Antwort definiert und kann optional Daten enthalten. Das eMMC-Protokoll ist blockorientiert. Das bedeutet, dass nach einer Kommando-Antwort-Sequenz einer (Single-Block) oder mehrere Datenblöcke (Multi-Block) folgen können. Das Lesen eines einzelnen Blocks würde beispielsweise mit dem *Kommando* READ_SINGLE_BLOCK initiiert werden, gefolgt von einer *Antwort* (Device bereit) und schließlich gefolgt von einem *Datenblock* [JED15].

Um den Speicher zu diagnostizieren oder zu konfigurieren, stellt die eMMC-Schnittstelle sieben Register zur Verfügung. Auf diese kann nur durch entsprechende Kommandos zugegriffen werden (z.B. Lesen des Register OCR mit Kommando SEND_OP_COND). Register enthalten speicherinterne Informationen, Konfigurationsparameter oder beides. Auf die einzelnen Register und deren Inhalt wird genau in Kapitel 4.1 eingegangen. Die Diagnose und Konfiguration eines eMMC wird also ausschließlich über das Schreiben und Lesen dieser speziellen Register abgehandelt [JED15].

UFS

Betrachtet man die Kommunikation des UFS, stellt man fest, dass diese im Vergleich zum eMMC weitaus komplexer aufgebaut ist. Das Systemmodell des UFS ist in drei Schichten (Layer) unterteilt:

- UFS InterConnect Layer (UIC)
- UFS Transport Protocol Layer (UTP)

- UFS Application Layer (UAP)

Auf die InterConnect Layer wird in dieser Arbeit nicht tief-greifend eingegangen werden. Sie implementiert die zwei Standards M-PHY und das darauf aufbauende UniPro der Organisation Mobile Industry Processor Interface (MIPI)-Alliance und regelt die Verbindung und die Datenübertragung auf Bitebene über die serielle Schnittstelle [JED].

Die Transport Protocol Layer und die Application Layer werden hier zusammengefasst und in den Grundzügen beschrieben, sodass die nachrichtenbasierte Kommunikation des UFS prinzipiell verstanden werden kann.

Bei dem vorliegenden Kommunikationsmodell handelt es sich um ein Client-Server-Modell oder Anfrage-Antwort-Architektur, das vom Small Computer System Interface (SCSI) Architecture Model stammt [Int10b]. Der Client wird hier Initiator genannt und repräsentiert den Host. Der Server wird Target genannt und wird vom Speicher repräsentiert. Der Initiator sendet dabei ein Kommando oder Service-Anfrage an das Target (UFS-Speicher), der daraufhin den Service ausführt und eine Status-Antwort zurückgibt. Ein Kommando ist allerdings nicht an einen Gesamtspeicher adressiert, sondern an dessen Untereinheiten, die Logical Units (LUs) genannt werden. Diese sind vollkommen eigenständige Untereinheiten des Speichers und führen Anfragen unabhängig voneinander aus. Ein UFS kann dabei eine oder mehrere LUs besitzen.

Die Kommunikationspartner sind also jeweils ein Initiator und ein Target, die über spezielle Kommandos miteinander kommunizieren. Der Befehlssatz folgt wiederum den zwei SCSI-Standards *Primary Commands* und *Block Commands* [Int10c] [Int10a]. Grundsätzlich besteht eine Transaktion aus drei Bestandteilen: einem Kommando, keine oder mehrere Datenpakete und einer Antwort. Bei einer Schreibaktion würde beispielsweise der Initiator einen Schreibbefehl an das Target (eine LU) senden, gefolgt von mehreren Schreibdatenpaketen. Schließlich wird das Target eine Antwort zurücksenden, die entweder Bescheid gibt, dass die Aktion erfolgreich war, oder dass ein Fehler vorliegt, wobei dieser mit einer detaillierten Beschreibung übermittelt werden würde.

Für die Diagnose gibt es zwei Mechanismen. Zum einen können die SCSI-Kommandos verwendet werden, um zum Beispiel die *Vital Product Data* auszulesen. Diese geben Auskunft über den Hersteller und Produktdetails des Speichers. Zum anderen kann jedoch auch das eigene UFS-Diagnose- und Konfigurationssystem verwendet werden. Genanntes System besteht aus drei Komponenten, die sich im Prinzip nur durch die Datengröße unterscheiden: Deskriptoren, Attribute und Flags. Sie definieren und kontrollieren die Eigenschaften des Speichers wie z.B. die Anzahl der LUs oder der Übertragungsmodus. Deskriptoren beschreiben dabei einen Block an Parametern, während Attribute und Flags nur einen Parameter kontrollieren (Attribute bis zu 32-bit, Flag 1 bit). Alle drei Komponenten können teilweise gelesen und geschrieben werden [JED].

Die Diagnose und Konfiguration des UFS kann also zusammengefasst über die SCSI-Kommandos oder über die umfangreicheren UFS-spezifischen Deskriptoren, Attribute und Flags durchgeführt werden. Genauer auf den Inhalt dieser Mechanismen wird in Kapitel 4.1 eingegangen.

3. Themenverwandte Arbeiten

In diesem Abschnitt soll der aktuelle Forschungsstand zum Themengebiet Performance-Testing von Flash-Speichern gegeben werden, da die Ermittlung der Performance einen essentiellen Teil der Eignungsfeststellung von Flash-Speichern darstellt. Das Prüfen des Vorhandenseins von Features des Speichers kann durch eine Diagnose gewährleistet werden und bedarf, abgesehen von den jeweiligen JEDEC-Speicherstandards (siehe Kapitel 4), keiner zusätzliche Literatur. Deshalb wird das Thema in diesem Kapitel nicht weiter behandelt.

3.1. SNIA Performance Test Specification

Die in [Sto15] beschriebene Testspezifikation, soll als Basis für die Performance-Testmethodik dieser Arbeit dienen. In dieser Spezifikation werden zum Thema Performance folgende Eigenschaften spezifiziert:

- Zu messende Performance-Metriken
- Einflussgrößen auf die Performance (Applikationsebene)
- Grundsätzliche Konzepte zum Performance-Test-Aufbau
- Detaillierte Beschreibung von Performance Tests inklusive Pseudo-Code
- Informationen zu verwendeten und empfohlenen Tools und Plattform

3.1.1. Storage Networking Industry Association

Da [Sto15] als besonders wichtig für diese Arbeit angesehen wird, ist eine genauere Erklärung unumgänglich. Es handelt sich dabei um eine Testspezifkation der SNIA. Die SNIA ist eine Non-Profit-Handelsvereinigung, deren Mitglieder Unternehmen der Informationstechnolgiebranche sind. Das Ziel dieser Organisation ist es, die Speicherindustrie anzuführen. Ermöglicht wird dies, durch das Entwickeln und Fördern von herstellerunabhängigen Architekturen, Standards und Bildungsdiensten, was wiederum ein effizientes Informationsmanagement, den Informationsfluss und die Informationssicherheit erleichtern. Wichtig ist hier anzumerken, dass die SNIA sich mit Speicher-Netzwerken befasst, d.h im Fokus stehen große Massenspeicher und deshalb beziehen sich alle Informationen, Standards, usw. auf SSD-Speicher in großen Netzwerken. Zu den Mitgliedern zählen, wie auch schon bei der JEDEC, namhafte Speicherhersteller (Toshiba, Samsung, SanDisk, etc.). Die Organisation führt einige technische Arbeitsgruppen, darunter auch die *Solid State Storage (SSS)* Arbeitsgruppe[1].

[1] Offizielle Webseite der SNIA - https://www.snia.org - Stand 08.02.2018

3.1.2. Solide State Storage

Im Allgemeinen wird der Begriff SSS für nicht-flüchtige Speicher, wie beispielsweise alle Flash-Speicher-Varianten oder EPROMS, verwendet. Im Whitepaper zum SSS-Performance-Test wird der Begriff ähnlich, als nichtflüchtiges Speichermedium, das integrierte Schaltkreise verwendet (RAM oder Flash) definiert. Dazu zählen weiter alle Formfaktoren (Hardwarevarianten), Schnittstellen und Technologien. SSDs werden dabei als Untermenge von SSS definiert [EAW]. In der tatsächlichen Testspezifikation wird jedoch nur SSD erwähnt, d.h. die Spezifikation ist implizit eine SSD-Spezifikation. Die Charakteristiken, die ein spezifisches Testen für SSD erfordern, werden jedoch als Flashcharakteristiken beschrieben (z.B. Garbage Collection, Wear Leveling). Daraus kann geschlussfolgert werden, dass die Testmethode für managed NAND-Speicher allgemein Anwendung findet. Die verwendeten Parameter und Werte können jedoch angepasst werden, da im Automotive-Bereich gerade eingebettete Speicher mit kleineren Speichergrößen verwendet werden und zusätzliche Parameter (z.B. Temperatur) überprüft werden müssen.

3.1.3. Performance Test Specification(PTS)

[Sto15] ist nun die aktuelleste Version der Performance Test Specification (PTS), die von der Arbeitsgruppe erstellt wurde. Die Beschreibung ist zusätzlich zum Spezifikationsdokument durch das Whitepaper [EAW] und durch ein Einführungsdokument (Primer)[Kim] gegeben. Erstellt wurde die Spezifikation von 32 Ingenieuren der namhaftesten Halbleiterhersteller (Dell, IBM, Intel), darunter auch speziell Speicherhersteller (z.B. Samsung, Toshiba, SanDisk, Micron). Daraus folgt, dass hier Spezialwissen aus allen Bereichen der Halbleiterindustrie vom Hersteller bis zum Nutzer eingeflossen ist und somit die Methodik als hochwertig eingestuft werden kann. Die Spezifikation umfasst dabei eine Methodik zur Vorkonditionierung des Speichers, sowie zu verschiedenen Performance-Tests und Anforderungen für die zu erstellenden Berichte. Nicht enthalten sind Tests, die die Performance von speziellen Applikationen messen, eine vorgegebene Testplattform (Hardware/Betriebsystem/Tools), ein Zertifizierungs- oder Validierungsprozess für die Spezifikation, sowie jegliche Tests zur Haltbarkeit, Datenverfügbarkeit oder Integrität.

Performance-Metriken

Wenn von Performance eines Speichers gesprochen wird, muss dies anhand festen Metriken diskutiert werden, die später auch gemessen werden können. In Kapitel 2 wurde nur die reine Bitübertragungsrate als Metrik verwendet. Nun werden jedoch speziellere und für die tatsächliche Performance (Lesen, Schreiben) aussagekräftigere Metriken eingeführt. Es werden laut [Kim] drei Metriken unterschieden:

- Input/Output Operations per Second (IOPS)
- Durchsatz
- Latenz (Antwortzeit)

Mit *IOPS* ist die Anzahl an I/O-Operationen in einer gewissen Zeiteinheit (Sekunden) gemeint. Die Einheit für die Anzahl wird in IOPS gemessen. Eine Beispielanwendung für eine sinnvolle Einordnung ist das Booten eines System. Hier entscheiden die IOPS, wie

schnell das Betriebssystem die kleinen Lese- und Schreibblöcke laden kann. Höhere IOPS sind, wie man hier erkennen kann, somit besser.

Beim *Durchsatz*, oft auch Bandbreite genannt, wird die Datenmenge gemessen, die vom oder zum Speicher transferiert wird. Gemessen wird diese beispielsweise in MB/s, also in Datenmenge pro Zeiteinheit. Als Anwendungsfall kann hier das Laden eines großer Bild- oder Videodateien für eine Streamingapplikation genannt werden. Je höher der Durchsatz, desto schneller kann dieses große File transferiert und anschließend verwertet werden.

Die *Latenz* wird laut [Sto15] durch drei zeitliche Teilbereiche definiert:

- Anfragelatenz (z.B für Leseoperation)
- Bearbeitungszeit
- Antwortlatenz

Die Latenz kann auch mit der Rundreisezeit einer I/O-Anfrage beschrieben werden. Die Latenz wird in Millisekunden oder Mikrosekunden gemessen und oft als Durchschnitts-(AVG) oder Maximallatenz(MAX) angegeben. Diese Zeit entscheidet beispielsweise, wie lange es dauert um ein spezielles File zu finden oder zu laden, also wie viel Verzögerung dabei entsteht.

Einflussgrößen

Wird in dieser Spezifikation von Einflussgrößen gesprochen, so sind jene gemeint, die auf der Applikationsebene einwirken. In Kapitel 2 hingegen wurden größtenteils die Einflussgrößen auf die Bitübertragung und durch die Umgebung generell erläutert.

Die essentielle Einflussgröße ist das *Zugriffsmuster*(Access Pattern), d.h. die Art der Operation, die auf den Speicher einwirkt. Dabei werden folgende drei Komponenten unterschieden:

- Wahlfreie/Sequentielle Adresszugriffe
- Blockgröße - Größe der Anfrage
- Verhältnis von Lese- zu Schreiboperationen

Ein Beispiel für diese Kategorien wäre „RND 4KB 65:35 R/W“. Hier wird eine Sequenz an I/O-Operationen beschrieben, bei der jede Anfrage eine Blockgröße von 4KB besitzt, auf wahlfreie Bereiche zugreift, in einem Verhältnis von 65% Lese- zu 35% Schreiboperationen.

Auf die Performance haben alle drei Kategorien wie folgt Einfluss: Sequentielle Operationen können schneller als wahlfreie Operationen sein, dies kommt auf die FTL an und folglich wie effizient das interne Mapping durchgeführt wird. Größere Blockgrößen begünstigen einen höheren Durchsatz, wobei kleinere Blockgrößen höhere IOPS begünstigen. Generell sind Leseoperationen bei Flash-Speichern schneller aus mehreren Gründen (siehe Kapitel 2. Das bedeutet, je höher der Schreibanteil ist, umso tendenziell länger brauchen die I/O-Operationen.

Zusätzlich können noch weitere Parameter Einfluss haben. Das *Datenmuster*(Data Pattern), das bedeutet zufällige Daten oder nicht-zufällige Daten, soll laut [Sto15] standarmäßig zufällig sein, kann jedoch bei speziellem Bedarf verändert werden. Es soll zufällig sein, da verschieden Speicher für bestimmte Muster optimiert sind und es nicht möglich ist für alle

Muster zu testen. Um nun vergleichbar zu sein, sollen deshalb die Daten zufällig und für jedes Device Under Test (DUT) gleich schwer zu verarbeiten sein.

Flash-Speicher können einen flüchtigen Speicher im Package beinhalten (siehe Kapitel 2), der für die FTL verwendet wird. Dieser kann jedoch auch als Pufferspeicher genutzt werden, wobei die Möglichkeit bestehen kann, diese Eigenschaft an- und auszuschalten. Der Puffer- oder Cachespeicher beschleunigt die Performance, da Daten erst im Cache gelagert werden und später erst in den Flash-Speicher weggeschrieben werden können. Laut [Sto15], soll der „Write Cache“ für alle Tests eingeschaltet sein.

Das Thema „*OIO*“ spielt eine weitere Rolle. Dabei ergeben sich die Outstanding I/O Operationss (OIOs) als Produkt von *Thread-Anzahl* und *Queue-Tiefe*. Queue-Tiefe bedeutet die Anzahl an noch ausstehenden I/O-Operationen. Eine OIO von 2 ergibt sich beispielsweise bei einem Thread mit einer Queue-Tiefe von zwei. Genauer wird dieses Thema noch im nächsten Kapitel beschrieben. Empfohlen wird nur, dass für alle Testschritte und alle Tests die selbe OIO verwendet wird.

Workloads

Mit dem Begriff *Workload* ist generell das Zugriffsmuster, gemessen während einer bestimmten Überwachungsperiode, gemeint (z.B. zehn Minuten random 4KB 100% Schreiboperationen). Wichtig ist hier zu verstehen, dass der vom Nutzer erzeugte Workload dabei sehr unterschiedlich zum Workload, der direkt am Speicher einwirkt, sein kann. Der Grund dafür liegt darin, dass der I/O-Request den kompletten I/O-Stack durchschreiten muss. Das heißt er muss von der Applikationsschicht durch alle Schichten bis hin zur Hardwareschicht durchgereicht werden. Eine Applikation kann beispielsweise hauptsächlich kleine Blocks an wahlfreiem IO-Datenverkehr im User Space generieren. Der I/O-Stack könnte dies jedoch bis zum Eintreffen der Anfrage beim Speicher ändern. Damit I/O-Stack-Einfluss weitestgehend eliminiert wird und eine bessere Vergleichbarkeit erzeugt wird, testen Speicherhersteller und Testhäuser die Speicherperformance auf „Block IO Level“. Deshalb sollten auch bei dieser Arbeit die Performance Tests mit einer Workloaderzeugung auf Block Level durchgeführt werden.

Ein weiterer Faktor ist, ob der Workload auf User Level erzeugt wird, d.h. für eine Applikation realistische Sequenzen an I/O-Operationen. Diese Sequenzen sind sehr komplex. Ein System kann beispielsweise einen Boot durchführen (Lese- und Schreiboperationen mit kleiner Blockgröße) und dann einen gemischten Strom an I/O-Operationen erzeugen (verschiedene Programme laufen). Währendessen erzeugt das Betriebssystem seine ganz eigenen I/O-Muster. Aufgrund dieser zu großen Komplexität werden laut [Kim] „allgemein akzeptierte“ Workloads genutzt. Diese können wiederum *monoton* sein, d.h. das selbe Zugriffsmuster wird konstant verwendet oder eine Mischung von Zugriffsmustern in einem *zusammengesetzten* Zugriffsmuster. In der Testspezifikation werden diese Parameter genau angegeben und können vom Tester bei Bedarf (spezieller Anwendungsfall) variiert werden.

Der letzte Einflussfaktor wird durch den Begriff des *aktiven Bereichs* beschrieben, also den Bereich (Anteil) des Speichers, der getestet wird. Auf Abbildung 3.1 sind die beiden spezifizierten Active Ranges gezeigt, wobei bei der linken Konfiguration der gesamte Speicherbereich des DUT stimuliert wird, während bei der rechten Konfiguration nur 75% stimuliert werden.

Wichtig ist es, diese Größe festzulegen, da ein zu großer inaktiver Bereich zum Problem der *Überprovisionierung* führen kann. Konkret heißt dies, dass gewisse FTLs diesen dauerhaft

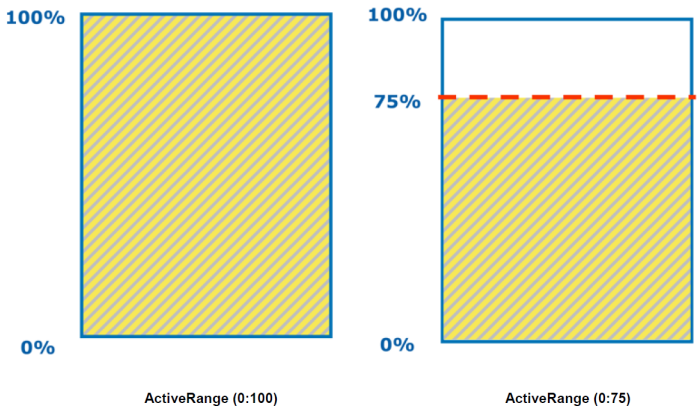


Abbildung 3.1.: Aktiver Bereich [Sto15]

freien Platz nutzen würden, um die Performance des Speichers zu erhöhen. Der Bereich kann als eine Art Puffer agieren. Es ist also wichtig vorher zu wissen, ob ein solcher freier Bereich in der realen Anwendung des Speichers vorgesehen ist. Ist dies nicht der Fall, so sollten die 100% Active Range genutzt werden, um realistische Ergebnisse zu erzielen.

Performance-Zustände

Um die Performance von Flash-Speichern korrekt zu evaluieren, muss das Konzept von Performence-Zuständen verstanden werden. In Kapitel 2 wurden bereits die Lese-, Schreib- und Löschvorgänge besprochen. Dabei wurde die grundlegende Problematik des P/E-Cycling erläutert, wobei ein Block gelöscht werden muss, um das Schreiben von Pages zu ermöglichen. Dies ist jedoch nur der Fall, wenn der Block bereits beschrieben wurde. Daraus folgt, dass Schreibaktionen im Fall eines leeren Speichers deutlich schneller verarbeitet werden können, da keinerlei Löschaktionen erforderlich sind, bis der Speicher komplett vollgeschrieben ist. Zu beachten ist hier, dass dieses Szenario für sequentielle Schreibaktionen gilt, bei denen keine Pages überschreiben werden sollen, sondern nur neue leere Pages beschrieben werden. Anders betrachtet, kann gesagt werden, dass die FTL erst allmählich beginnt zu arbeiten und somit zusätzlichen Overhead zu erzeugen, wenn der Speicher sich dem vollständigen Füllzustand nähert [EAW].

Aus den genannten Gründen und praktischen Untersuchungen, wurden drei verschiedene Performance-Zustände in [EAW] definiert, die auf Abbildung 3.2 zu sehen sind:

- Fresh out of the box (FOB)-Zustand
- Übergangszustand
- Steady State (hier werden valide Ergebnisse gemessen)

Das Diagramm zeigt, wie sich die Anzahl der IOPS von verschiedenen Flash-Speichern (SSDs) über eine Zeitspanne von 700 Minuten kontinuierlicher I/O-Operationen verändert. Zu erkennen ist dabei, dass die Performance (IOPS) drastisch über die Zeit sinkt. Im FOB-Zustand (komplett leerer Speicher) erzielen alle Speicher ihre höchste Performance. Diese Periode wird von einem Übergangszustand gefolgt, der beginnt, sobald der Speicher fast

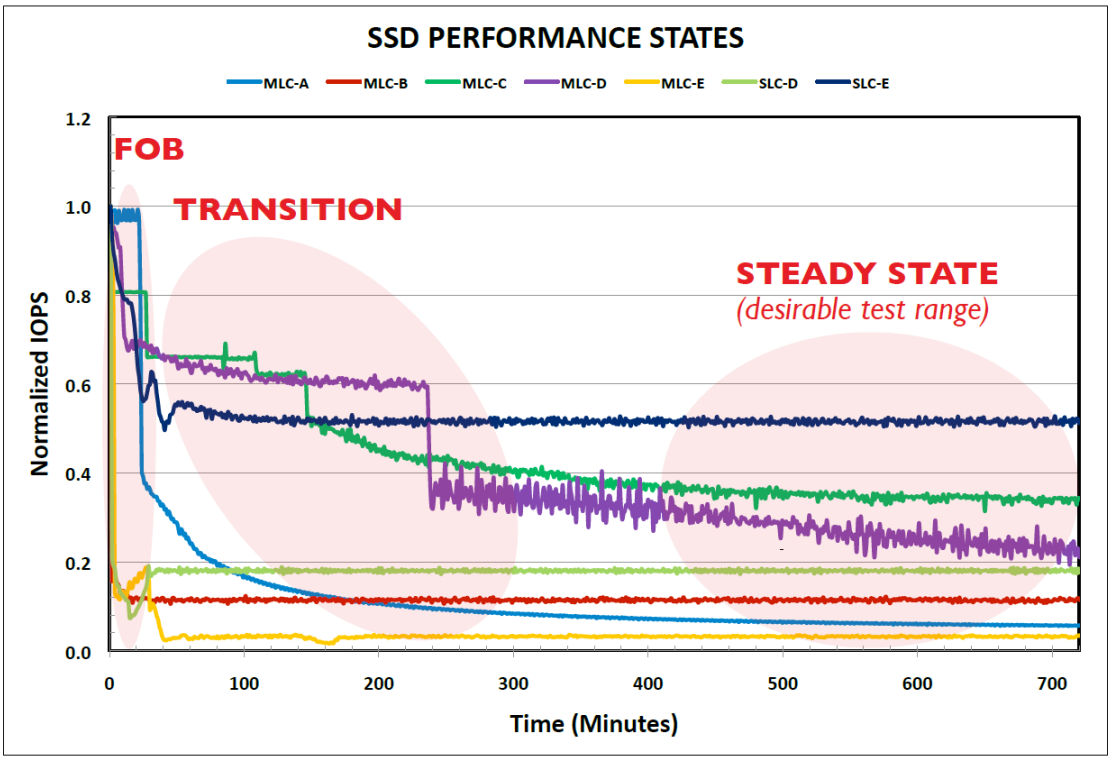


Abbildung 3.2.: Flash Performance States am Beispiel verschiedener SSDs [EAW]

voll ist. Hier fängt die FTL an zu arbeiten und verschiedene Algorithmen wie Wear Leveling und Garbage Collection (siehe Kapitel 2.5.2) erzeugen zunehmend Overhead. Dieser Zustand wird von dem sogenannten „Steady State“ abgelöst. Hier ist der Speicher sozusagen „eingeschwungen“ und die FTL hat ihr höchstes gleichbleibendes Arbeitspensum erreicht. Da konstant Performance-Samples (hier IOPS) gespeichert werden, kann der Steady State durch zwei statistische Gegebenheiten charakterisiert werden, die für alle gemessenen Performance-Metriken gelten:

- Es gab nicht mehr als 20% Variation der Extremwerte (Min, Max) von der Durchschnittsperformance seit fünf aufeinanderfolgenden Messpunkten
- Die maximale Differenz von Minimum-Messpunkt und Maximum-Messpunkt zu der linearen Kurve, die diese fünf Messpunkte am besten annähert, darf 10% nicht überschreiten

Die Testspezifikation beschreibt dafür ein einheitliches Verfahren, wie der Speicher für jede zu testende Metrik in den Steady State gelangt. Der Grund dafür ist, dass nur Ergebnisse, die in diesem Zustand aufgezeichnet werden valide vergleichbare Ergebnisse darstellen.

Grundlagen zum Testaufbau

Im Folgenden sollen wichtige Grundlagen zum grundsätzlichen Testaufbau erläutert werden. Auf Abbildung 3.3 sind die grundsätzlichen vier Testphasen zu sehen. Kurz zusammengefasst

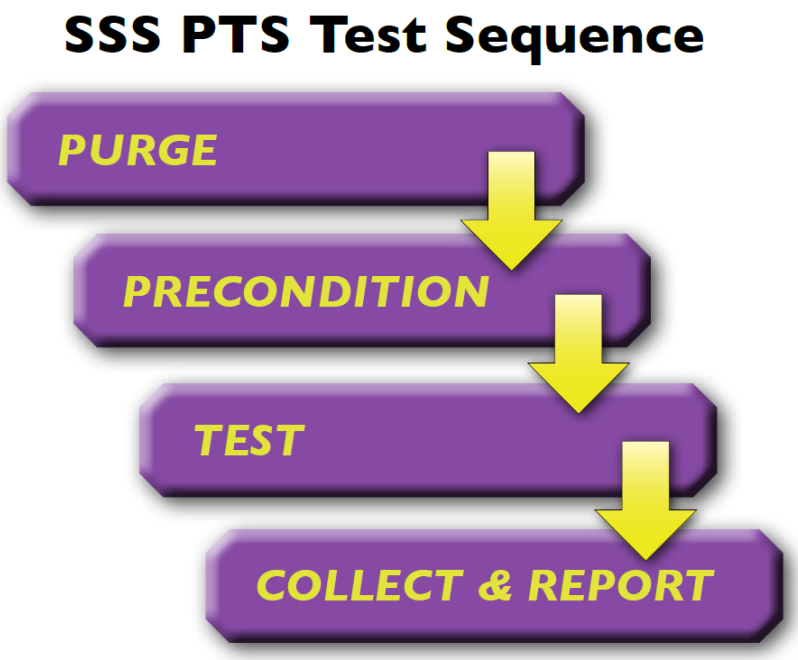


Abbildung 3.3.: Grundsätzlicher Testablauf [EAW]

kann gesagt werden, dass in Phase eins ein einheitlicher Startpunkt für den Test hergestellt wird, dann in Phase zwei der Steady State erreicht wird und in Phase drei getestet wird. In Phase vier werden schließlich die gesammelten Ergebnisse in einheitlichen Formaten präsentiert [Sto15].

Damit das DUT vergleichbar getestet werden kann, soll es am definierten Teststartpunkt jeden Tests in den FOB-Zustand gebracht werden, um sich dann von diesem Zustand an den Steady State heranzutasten. Um in den FOB-Zustand zu kommen, wird immer am Anfang des Tests ein sogenannter „Purge“ durchgeführt. Ein Purge wird als Prozess definiert, der alle Daten im gesamten Nutzerbereich löscht. Nach einem Purge erzielen Schreiboperationen ihren besten Performance-Wert, da auf leere Blöcke geschrieben wird und keinerlei Löschaktionen mehr für die Bereinigung notwendig sind, bis der Speicher wieder gefüllt ist.

Um in den Steady State zu kommen, soll eine Vorkonditionierung für den eigentlichen Test durchgeführt werden. Diese unterteilt sich wiederum in zwei Teilprozesse:

- Workload unabhägige Vorkonditionierung: Umfasst einen vorgeschriebenen Workload, der vom Test unabhängig ist, um die Annäherung an den Steady State zu vereinfachen (beschleunigen).

- Workload abhängige Vorkonditionierung: Umfasst das Durchführen der eigentlichen Test-Workload, um das DUT in den Steady State in Beziehung auf die zu testende Metrik zu bringen.

Das DUT wird also erst von der Test-Metrik unabhängig möglichst schnell und nah an den Steady State angenähert. Anschließend wird der Test-Workload durchgeführt und sobald der Steady State tatsächlich festgestellt wurde, wird der Test beendet und ein Durchschnittswert

aus den Testrunden berechnet, die den Steady State erzielten. Zeitlich gesehen sind die Workload abhängige Vorkonditionierung und der Test an sich eine Testphase.

Sind alle Datenpunkte gesammelt, werden die Ergebnisse auf exakt vordefinierten Plots dargestellt. Der Inhalt der Plots ist für jeden Test individuell spezifiziert, wobei drei Arten von Plots unterschieden werden:

- Steady State Annäherungs-Plot: zeigt visuell, wie sich die zu testende Metrik bei der Annäherung an den Steady State verhält
- Steady State Verifikations-Plot: zeigt, dass sich das DUT im Steady State befindet, wobei die zu testende Variable im Messfenster untersucht wird
- Messungs-Plot: eine Gruppe von Plots/Berichten, die die zu untersuchende Variable im Messfenster präzisieren

Testumgebung

[Sto15] ist plattformagnostisch. Das bedeutet, es werden verwendete Hardware, Betriebssystem oder Software-Tools nicht vorgeschrieben. Für die Hardwareplattform wird unverbindlich eine Referenzplattform vorgestellt, auf welcher eine Linux-Distribution installiert wurde. Für die verwendete Software werden verschieden Anforderungen spezifiziert:

- Fähigkeit, als Workload-Generator und Datenrekorder zu fungieren
- Fähigkeit, verschiedene wahlfreie und sequentielle Block-Level-I/O-Operationen anzufordern
- Fähigkeit, Zugriffe auf Logical Block Adresss (LBAs) auf einen bestimmten Bereich oder auf mehrere bestimmte Bereiche innerhalb des verfügbaren Benutzerbereichs zu beschränken
- Der aktive Bereich soll auf eine bestimmte Anzahl an LBAs beschränkt werden können
- Fähigkeit eine Vielzahl an gleichgroßen LBA-Segmenten zufällig über den aktiven Bereich zu verteilen
- Fähigkeit, das Verhältnis von generierten Lese- zu Schreibaktionen festzusetzen
- Fähgikeit, das Verhältnis von wahlfreier zu sequentieller IO festzusetzen
- Die Größe der I/O-Anfragen muss gesetzt werden können.
- Fähigkeit, multiple ausstehende I/O-Anfragen zu generieren und aufrecht zu erhalten. Alle Testsequenz-Schritte sollen direkt nacheinander ausgeführt werden. Dies ermöglicht, dass den Speichern nicht die Zeit gegeben wird, sich zwischen Prozessschritten zu erholen, es sei denn dies ist ein definiertes Testziel.
- Es muss möglich sein Output zu generieren, von dem IOPS, MB/s, maximale Latenz und die durchschnittliche Antwortzeit in einer bestimmten Messperiode abgeleitet werden können.

Des Weiteren werden Anforderungen an den Zufallsgenerator der Software gestellt. Die Software muss zufällige LBAs für wahlfreie I/O-Operationen ausgeben. Diese muss die Möglichkeit haben einen Seed zu setzten, einen Output von mindestens 48 Bit haben und eine gleichmäßige zufällige Verteilung, unabhängig von der Output-Größe, liefern.

Für die Testtools werden drei unverbindliche Vorschläge gegeben [EAW]:

- Calypso (kostenpflichtige Hardware und Software)
- IOmeter (Open Source Software)
- Vdbench (Open Source Software)

Calypso ist ein kommerzielles Test-Tool bestehend aus einer Hardware mit zugehöriger Software zum Testen von SSDs. Es beinhaltet automatisiertes Testen und Erstellen von Berichten nach Vorschrift der PTS. Für die Zwecke dieser Arbeit ist es allerdings nicht interessant, da es nicht Open Source ist und somit zukünftige Forschung nicht darauf aufbauen kann. Viel wichtiger noch ist, dass es eine Hardware beinhaltet, die nur eine Schnittstelle zu SSD hat und somit für den Test anderer Flash-Speicher, wie UFS oder eMMC nicht geeignet ist. Würde die Arbeit der Frage nachgehen, ob nur SSDs für den Automobilbereich geeignet sind, so wäre dies ein interessantes Tool. Das Tool ist deswegen so ausgereift im Hinblick auf die PTS, da das Unternehmen Calypso Mitglied in der SNIA ist.

IOmeter ist ein Open Source Mess- und Charakterisierungstool für das I/O-Subsystem. Es wurde ursprünglich von Intel entwickelt und dann als Open Source Project freigegeben. Das Tool ist gleichzeitig ein Generator für Workloads (es erzeugt I/O-Operationen im System) und ein Performance-Messinstrument (es inspiziert die Performance und den Einfluss, die seine erzeugten I/O-Operationen auf das System auswirken). Es kann dabei synthetische Workloads generieren und ist dabei sehr flexibel.

Vdbench ist auch ein Workload-Generator. Das Tool wird für die Verifizierung von Datenintegrität und für die Performance-Messung von direkt mit dem System verbundenem oder über ein Netzwerk verbundenem Speicher verwendet. Auch dieses Tool ist Open Source.

Tests

Nun soll erläutert werden welche Tests spezifiziert sind. Genauere Beschreibung auf Pseudo-Code-Ebene sowie eine komplette Beschreibung aller Ergebnisberichte sollen im Kapitel 4 folgen. Im Prinzip gibt es für jede Testmetrik (IOPS, Durchsatz, Latenz) einen Test sowie einen zusätzlichen Schreibsättigungstest.

Beim IOPS-Test werden Blockgrößen (Block Sizes (BSs)) und Read-Write (RW)-Verhältnisse (Lesen und Schreiben) im Steady State variiert. Der Test erstellt somit eine 56-Element-Matrix von 8 BS x 7 RW-Verhältnissen in einer Tabelle. Dies erlaubt somit dem Nutzer ein schnelles Auswählen der BS/RW-Verhältnisse, die dem Workload entsprechen, der ihn interessiert. Es gibt drei Kombinationen, die besonders interessant sind:

- 4KB 100% R und 100% W
- 4KB/8KB 65:35 RW

Laut [Kim] machen 4KB 100% R und 100% W einen großen Anteil von vielen Workloads bei Performance-Tests aus. Die Blockgröße 4KB ist dabei ein typischer Wert, der für sogenannte „Corner Case-Tests“ mit kleinen Blockgrößen verwendet wird. Bei Corner Case Tests

Test Run Date:	03/02/2013 03:30 PM	Report Run Date:	05/13/2013 05:48 PM

IOPS Test (REQUIRED) - Report Page

SNIA SSS TWG	Solid State Storage Performance Test Spec (PTS)	IOPS - Block Size x RW Mix Matrix	Rev.	PTS-C 1.1
			Page	4 of 6

Vendor:	ABC Co.	SSD Model:	ABC Co. MLC-A 250	TEST SPONSOR	CALYPSO Systems

Test Platform		Device Under Test		Set Up Parameters		Test Parameters	
Ref Test Platform	Calypso RTP 2.0	Mfgr	ABC Co.	Data Pattern	RND	Data Pattern	RND
Motherboard	Intel S520HC	Model No.	MLC-A	AR	100%	AR & Amount	100%
CPU	Intel XEON 5580W	S/N	123 456	AR Segments	N/A	Test Stimulus 1	IOPS Loop
Memory	8 GB PC1600 DDR2	Firmware ver	FFFF	Pre Condtion 1	SEQ 128K W	RW Mix	Outer Loop
Operating System	CentOS 6.3	Capacity	250 GB	TOIO - TC/QD	TC 2/ QD 16	Block Sizes	Inner Loop
Test SW	CTS 6.5 1.13.8	Interface	SATA 6Gb/s	Duration	Twice User Capacity	TOIO - TC/QD	TC 4/QD 16
Test SW Info	1.10.7/1.9.16	NAND Type	MLC	Pre Condtion 2	IOPS Loop	Steady State	1 - 5
Test ID No.	R30-5146	PCIe NVM	N/A	TOIO - TC/QD	TC 4/ QD 16	Test Stimulus 2	N/A
HBA	LSI 9212-4e4i	Purge Method	Security Erase	SS Rounds	1 - 5	TOIO - TC/QD	N/A
PCIe	Gen 2 x 8	Write Cache	WCE	Note	-	Steady State	N/A

Client IOPS - ALL RW Mix & BS – Tabular Data

Block Size (KiB)	Read / Write Mix %						
	0/100	5/95	35/65	50/50	65/35	95/5	100/0
0.5	49,053.4	32,137.8	21,205.1	21,452.2	22,868.5	45,001.1	112,880.9
4	62,079.1	27,433.1	18,450.9	18,515.8	19,760.5	37,734.1	70,630.4
8	32,683.6	16,954.9	11,394.2	11,547.9	11,968.9	22,078.8	45,403.6
16	16,306.5	10,776.8	7,430.6	7,536.9	7,756.2	12,587.9	26,747.9
32	8,137.5	6,903.3	4,821.4	4,894.0	5,156.5	7,500.5	15,215.7
64	4,070.8	4,097.6	2,980.1	3,044.3	3,218.5	4,650.9	8,169.2
128	2,034.1	2,113.2	1,830.5	1,912.9	2,034.1	2,827.4	4,224.2
1024	253.4	263.6	293.6	317.5	352.9	474.9	540.6

Abbildung 3.4.: Ergebnisbericht eines IOPS-Tests [Kim]

werden verschiedene Variablen auf extreme „Eckwerte“ konfiguriert, um das Verhalten bei gewissen Grenzkonfigurationen der Parameter zu untersuchen. 4KB/8KB 65:35 RW ist deswegen so interessant, weil dieses Mischverhältnis in vielen Betriebssystemen und Applikationen zu finden ist. Das Mischverhältnis ist deshalb passend, um für verschiedene Blockgrößen die IOPS zu vergleichen.

In anderen Worten kann gesagt werden, dass mit diesen drei Werten schnell die IOPS-Performance eines Speichers auf einen Blick gesehen werden kann. Auf Abbildung 3.4 ist der Ergebnisbericht eines IOPS-Tests zu sehen, bei dem die drei wichtigen Werte hervorgehoben wurden. Hier entspricht eine Spalte verschiedenen Blockgrößen und eine Zeile verschiedenen RW-Verhältnissen.

Ein weiterer Interpretationsansatz ist es, eine einzige Spalte (z.B. 100% W) oder eine einzige Reihe (z.B. 4KB) zu betrachten. Hierbei können mögliche Optimierungen des Speichers auf eine bestimmte Blockgröße erkannt werden. Bei den Spalten ist besonders die 100% W-Spalte interessant, da hier Optimierungen auf bestimmte Blockgrößen besonders gut zu erkennen sind.

Beim Durchsatztest werden große Blockgrößen für sequentielles 100% R und W verwendet. Die Blockgröße wird dabei auf 1024 KB spezifiziert. Als Ergebnis gibt es also nur zwei Maxmimaldurchsatzwerte für jeweils Schreiben und Lesen und keine Tabelle.

Der Latenztest misst Antwortzeiten von einzelnen IOs. Als Metrik dienen sowohl die Durchschnitts- als auch die Maximalantwortzeit für drei verschiedene Blockgrößen und drei verschiedene RW-Mischverhältnisse. Auf Abbildung 3.5 ist der Ergebnisbericht eines Latenztests zu sehen.

Der letzte Test ist der sogenannte Schreibsättigungstest, bei dem eine der drei Performance-

Test Run Date:	05/19/2013 02:24 PM	Report Run Date:	05/19/2013 04:52 PM

LATENCY Test (REQUIRED) - Report Page

SNIA SSS TWG	Solid State Storage Performance Test Spec (PTS)	LATENCY - Response Time OIO=1	Rev.	PTS-E 1.1
			Page	4 of 6

Vendor:	ABC Co.	SSD Model:	ABC Co. MLC-A 256	TEST SPONSOR	CALYPSO Systems

Test Platform		Device Under Test		Set Up Parameters		Test Parameters	
Ref Test Platform	Calypso RTP 2.0	Mfgr	ABC Co.	Data Pattern	RND	Data Pattern	RND
Motherboard	Intel 5520HC	Model No.	MLC-A	AR	100%	AR & Amount	100%
CPU	Intel XEON 5580W	S/N	123456	AR Segments	N/A	Test Stimulus 1	LAT Loop
Memory	8 GB PC1600 DDR2	Firmware ver	ABCDEF	Pre Condtion 1	SEQ 128K W	RW Mix	Outer Loop
Operating System	CentOS 6.3	Capacity	256 GB	TOIO - TC/QD	TC 1/ QD 1	Block Sizes	Inner Loop
Test SW	CTS 6.5 1.13.8	Interface	SATA 6Gb/s	Duration	Twice User Capacity	TOIO - TC/QD	TC 1/QD 1
Test SW Info	1.10.9/1.9.16	NAND Type	MLC	Pre Condtion 2	LAT Loop	Steady State	1-5
Test ID No.	R30-5314	PCIe NVM	N/A	TOIO - TC/QD	TC 1/ QD 1	Histogram	N/A
HBA	LSI 9212-4e4i	Purge Method	Security Erase	SS Rounds	1-5	TOIO - TC/QD	N/A
PCIe	Gen 2 x 8	Write Cache	WCE	Note	-	Note	-

Average and Maximum Response Time - ALL RW Mix & BS – Tabular Data

Average Latency (ms)			
	Read / Write Mix %		
Block Size (KiB)	0/100	65/35	100/0
0.5	0.20	0.24	0.13
4	0.19	0.24	0.14
8	0.29	0.42	0.19

Maximum Latency (ms)			
	Read / Write Mix %		
Block Size (KiB)	0/100	65/35	100/0
0.5	38.92	9.31	0.79
4	19.10	9.37	0.79
8	34.43	9.38	6.25

Abbildung 3.5.: Ergebnissbericht eines Latenztests [Kim]

Metriken gewählt werden kann, aber standardmäßig IOPS gemessen werden. Anders als bei den anderen Test soll hier jedoch keine Vorkonditionierung stattfinden und es ist auch kein Ziel den Steady State festzustellen. Stattdessen soll die Performance-Entwicklung über eine Periode mit dem Purge als Startpunkt untersucht werden. Es kann hier z.B. herausgefunden werden wie konstant das DUT über die Zeit die Performance hält, oder ob sich darin bestimmte Phasen von hoher oder niedriger Performance erkennen lassen. Es wird ein Workload nach dem Schema „zufällig 4KB 100% W“ verwendet. Interessant ist es hier, dass der Test über eine bestimmte Zeitperiode oder eine fest definierte Anzahl an geschriebener Datenmenge ausgeführt wird.

3.2. Ergänzende Literatur

Um die Performance-Test-Methodik der SNIA im Hinblick auf den Automobilbereich zu ergänzen, soll im folgenden noch zusätzliche ergänzende Literatur analysiert werden.

3.2.1. Write Amplification Factor

Der wichtige Begriff *Write Amplification Factor (WAF)* wurde bis jetzt nur indirekt beim Thema FTL angesprochen. In [LKB+17] wird dieser Begriff erst definiert, dann wird dieser praktisch bei SSDs gemessen und anschließend wird noch eine Lösung erarbeitet, wie dieser reduziert werden kann. Wichtig für diese Arbeit ist die Definition des WAF sowie der Test, der diesen ermittelt, um diesen Faktor in der eigenen Arbeit messen zu können. Der WAF definiert sich als die Menge der Daten, die tatsächlich auf den Flash geschrieben werden im Verhältnis zu der Menge an Daten die eigentlich durch I/O-Requests angefragt wurden. Die FTL erzeugt durch ihre Algorithmen (Garbage Collection, Wear Leveling) zusätzliche Schreibaktionen, die in bestimmten Fällen die Haltbarkeit und Performance des Speichers negativ beeinflussen. Ein Beispiel dafür wäre, wenn der Garbage Collector nicht volle Blöcke löscht, weil die FTL auf Performance ausgelegt ist und zu vorzeitig für freien Speicherplatz sorgt. Deshalb ist es wichtig diesen Faktor zu kennen und somit die Effizienz der Speicher-Firmware in Erfahrung zu bringen. Diese Metrik ist also keine strikte Performance-Metrik, sondern eher ein Indikator der generellen Effizienz eines Speichers. Wichtig ist auch, dass dieser Faktor in Speicher-Lastenheften im Automobilbereich vorgegeben wird. Ein Speicher mit einem zu hohen WAF erfüllt nicht die Anforderungen. In [LKB+17] wurde dieser Faktor bei SSDs experimentell ermittelt. Dafür wurde *SSDsim*, ein SSD-Simulator, verwendet um Workloads zu generieren und anschließend die tatsächlich erfolgten Schreibaktionen gemessen und ins Verhältnis gesetzt. Dieses Experiment verwendet also auch Block oder Device Level I/O, wie es auch in der PTS vorgeschrieben ist. Die verwendeten Workloads können in zwei Kategorien aufgeteilt werden. Die erste Einteilung stellt synthetische Workloads dar, d.h. sequentielle und wahlfreie I/O-Operationen. Dabei wurden fünf Millionen Schreiboperationen mit 4KB Datenblöcken zu insgesamt 20 GB geschrieben. Die zweite Kategorie wird durch zwei reale Workloads repräsentiert. Hier wurde zum einen *JEDEC*, ein Workload zum Testen der Haltbarkeit von der Joint Electron Device Engineering Council (JEDEC) konzipiert, verwendet. Zum anderen wurde *Online Transaction Processing (OLTP)*, ein Workload, der Finanztransaktionen simulieren soll, verwendet. Bei den synthetischen Workloads wurde bei den wahlfreien Zugriffen ein WAF von 2.84 und bei den sequentiellen Zugriffen ein WAF von 1.00 gemessen. Daraus schließt sich, dass bei wahlfreien Zugriffen eine angeforderte Schreiboperation rund drei Schreibaktionen auf dem Flash erzeugt während sequentiell

keine Erhöhung der Schreibanzahl erkennbar ist. Dies ist dadurch zu erklären, dass beim wahlfreien Schreiben Blöcke mit vielen invaliden Pages entstehen. Die validen Pages in diesen Blöcken werden dann in leere neue Blocks kopiert während der alte Block gelöscht wird. Dieser Kopiervorgang erzeugt die zusätzlichen Schreibaktionen. Beim sequentiellen Schreiben enthält der zu überschreibende Block nur invalide Pages und erzeugt somit keine zusätzlichen Scheibaktionen nach dem Löschen. Die realen Workloads haben dabei ein sehr ähnliches Ergebnis zu den 100% wahlfreien Zugriffen gezeigt, woraus abgeleitet werden kann, dass der WAF in realen Workloads denen einer vollständig wahlfreien Workload entspricht. Aufgrund der Wichtigkeit des WAFs soll dieser gemessen werden. Genauer wird darauf in Kapitel 4 eingegangen.

3.2.2. Temperatur

[Che08] wurde von der NASA durchgeführt und fokussiert sich ausschließlich auf die Haltbarkeit der Speicher, indem Zeit durch seine Erhitzung simuliert wird. Der Einflussfaktor Temperatur soll dabei für die eigene Arbeit übernommen werden. Bei der Anwendung in einer Weltraum-Umgebung steht das Material im Vordergrund, da der Speicher nicht ausgetauscht oder repariert werden kann. Zusätzlich ist dieser auch noch hoher Strahlung ausgesetzt. Aus diesem Grund wurde auch eine experimentelle Studie in dieser Arbeit erwähnt, die sich auf die Strahlenbelastung von Flash-Speichern konzentriert. Der Testaufbau ist eine Simulation von Zeit durch Hitze. Es wird konkret ein Speicher sehr hoher Temperatur ausgesetzt, um die Data Retention von mehreren Jahren zu simulieren. Es wurde hier untersucht, welcher physikalischer Effekt die Hauptursache für den Datenverlust ist. Da bei den Tests kein ECC verwendet wurde, wurde geschlussfolgert, dass ein ECC essentiell für das Erreichen einer hohen Data Retention ist. In dieser Arbeit wurde die Temperatur nur im Hinblick auf die Haltbarkeit untersucht. Im Automobilbereich werden Speicher auch für verschiedene Umgebungstemperaturen spezifiziert. Interessant wäre es nun auch zu wissen, wie sich die Temperaturen am Rande des spezifizierten Bereichs auf die Performance der Speicher auswirkt. Wie die Temperatur in die Testmethodik integriert werden kann, deckt Kapitel 4 auf.

3.2.3. Stromverbrauch

In [OSSP08] wird der Stromverbrauch von USB-Speichern gemessen. Der Energieverbrauch spielt gerade bei eingebetteten Systemen und somit auch im automotiven Einsatzgebiet eine entscheidende Rolle. Für den Testaufbau wurden verschiedene USB-Speichersticks auf verschiedenen Host-Geräten (Testboards) getestet. Die Messwerte wurden dabei von einem Multimeter und einem Oszilloskop abgelesen. Als Ergebnis stellt [OSSP08] fest, dass der Energieaufwand von Schreib- und Leseoperationen den Energieaufwand im Idle-Zustand (Leerlauf) nicht signifikant überschreiten. Auf Abbildung 3.6 wird dieses Ergebnis dargestellt. Als zweites Hauptergebnis wird festgestellt, dass die Größe des Flash-Speichers nur sehr geringen Einfluss auf den Energieverbrauch hat. Wie der Stromverbrauch in die Testmethode integriert werden kann, wird in Kapitel 4 erklärt.

3.2.4. P/E-Cycling und Betriebssystem

Bei [Kha10] handelt es sich um eine umfangreiche Studie zu themenverwandten Arbeiten. Aus dieser Studie können die zwei Performance-Faktoren P/E-Cycling und Betriebssystem

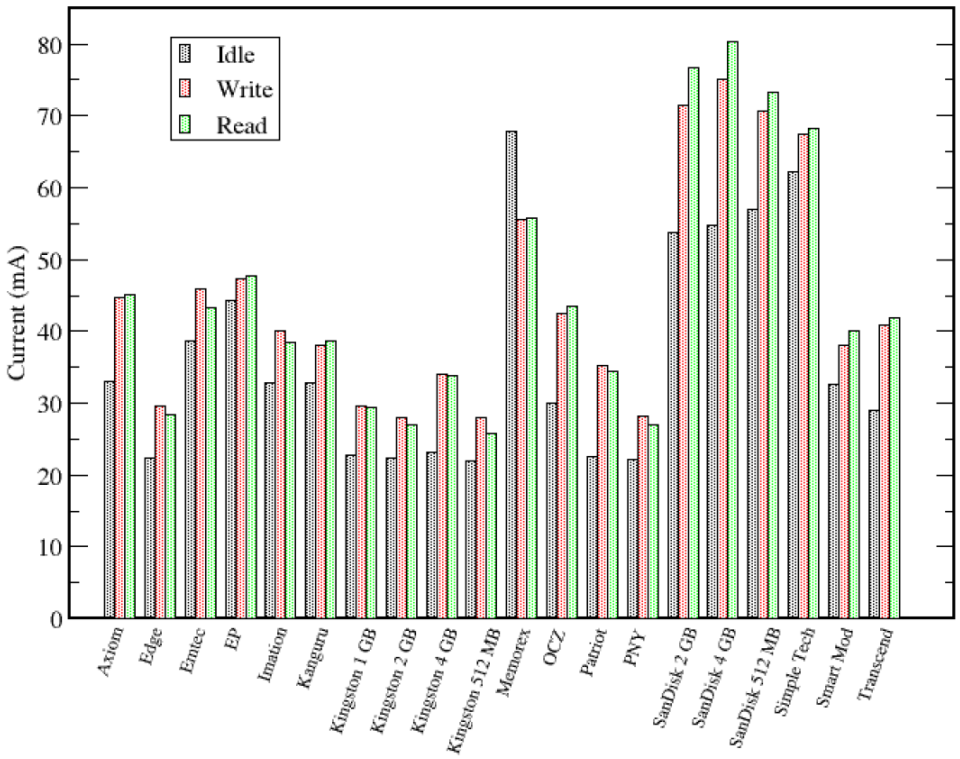


Abbildung 3.6.: Energieverbrauch von verschiedenen USB-Speichersticks bei Schreib- und Leseoperationen, sowie im Leerlauf

entnommen werden. Außerdem können die hier gefundenen Ergebnisse später zur Interpretation der eigenen Testergebnisse verwendet werden.

Der erste wichtige Punkt der vorliegenden Arbeit ist die Charakterisierung der Workloads bei Performancetests. Generell können die I/O-Requests laut [Kha10] durch folgende Kriterien charakterisiert werden:

- Request-Zeit: Zeitpunkt der Anfrage
- Request-Größe: z.B. Größe eines zu schreibenden Datenpakets
- Request-Lokalisation: der logische Ort der Anfrage oder auch LBA
- Request-Modus: lesen oder schreiben

Die zeitliche Komponente kann nun in folgende drei Kategorien unterteilt werden:

- *aufeinanderfolgende* Anfragen, wobei eine Anfrage beginnt, sobald die vorhergehende abgeschlossen wurde
- es kann eine *Pause* zwischen Anfragen eingeführt werden
- ein *Burst*-Pattern: hier werden Pausen zwischen mehreren nacheinander folgenden Anfragen gemacht

Für die örtliche (engl. spatial) Komponente werden vier verschiedene Patterns vorgeschlagen:

- *sequentiell*
- *zufällig*
- *geordnet*, wobei ein lineares Inkrement (oder Dekrement) auf jede LBA angewendet wird
- *partitioniert*: hier wird der Zielspeicherplatz in verschiedene Partitionen geteilt, die wie in einer Art Round Robin behandelt werden; innerhalb jeder Partition sind die Anfragen sequentiell

Außerdem wird auf den logischen Level (Betriebssystem) eingegangen und zwischen verschiedenen Zugriffsmodi unterschieden:

- Beim *Buffered Mode* nutzt das System Cache-Buffers (Lesen oder Schreiben)
- Beim *Direct Mode* wird das System dazu forciert, die Daten direkt vom Speichermedium in den User-Buffer zu kopieren, ohne einen File-Cache-Buffer zu nutzen
- Beim *Synchronous Mode* antwortet das Speichermedium erst dann, wenn die angefragte Operation tatsächlich durchgeführt wurde

Auch die Ergebnisse der analysierten Studien werden in jener Arbeit präsentiert. Eine wichtige Erkenntnis ist, dass der Zustand der zu untersuchenden Speicher gleich sein muss, um valide Ergebnisse zu erzielen. Die Speicher müssen also im selben P/E-Cycle-Bereich liegen. Die referenzierte Studie [J. 09] betont die Wichtigkeit der Rolle des Betriebssystems, der Speicherschnittstelle (z.B. USB) und der FTL beim Thema I/O-Performance. Die Variation des Zugriffsmodus beim Schreibvorgang beispielsweise reduziert die Performance drastisch. Dabei bewegt sich der maximale Durchsatz von ca. 6 MB/sec bei einem Buffered Write hin zu einem Durchsatz von weniger als 0.1 MB/sec, bei einem synchronen Schreibvorgang. Des Weiteren verringert sich die Performance bei fehlausgerichteten (engl. misaligned) Schreiboperationen (die Startadresse der Schreibaktion ist kein Vielfaches der Blockgröße) verglichen mit richtig ausgerichteten Schreiboperationen. Der vorher erwähnte *Burst* (mehrere Anfragen hintereinander gefolgt von einer Pause) von wahlfreien Schreiboperationen verlangsamt darauffolgende sequentielle Schreiboperationen. Allerdings wurde dieser Effekt nicht auf darauffolgende Leseoperationen festgestellt.

In [L.] wird beschrieben, dass wahlfreie I/O-Anfragen signifikant die Performance bremsen, im Vergleich zu sequentiellen Anfragen. Außerdem entsteht kein Performance-Boost durch ein paralleles abschicken von I/O-Anfragen, sondern teilweise sogar das Gegenteil.

Zusammengefasst kann aus dieser Arbeit entnommen werden, dass die Perfomance des Speichers vom Zugriffsmodus des Betriebssystems abhängt. Des Weiteren muss beim Vergleich der Performance verschiedener Speicher die Abnutzung durch P/E-Cycling identisch sein, um vergleichbare Ergebnisse zu erzielen. Wie diese beiden Faktoren in die Testmethode integriert werden können, wird in Kapitel 4 erläutert. Außerdem ergeben sich aus dieser Studie einige interessante Ansatzpunkte für das Interpretieren von Testergebnissen.

3.3. Weitere themenverwandte Arbeiten

In diesem Teil der Arbeit sollen themenverwandte Arbeiten gezeigt werden, die zwar das selbe Themengebiet bearbeiten, jedoch keine zusätzlichen Erkenntnisse liefern, die speziell für die eigene Arbeit genutzt werden können. Dadurch soll ein vollständigerer Eindruck vom Forschungungsfeld der Flash-Performance gewonnen, eventuell das eigene Konzept bestätigt und Schwächen der vorgestellten Arbeiten aufgezeigt werden.

Performance- und Zuverlässigkeitsstudie zu SSDs

In [Gua13] wird der aktuelle Stand zu Flash Performance und Haltbarkeit erläutert und zusätzlich Forschung betrieben, wie bestehende Probleme behoben werden können. Hierbei wurden drei theoretische Konzepte erstellt. Die Arbeit geht dabei ausschließlich auf SSDs ein.

Der erste entwickelte Ansatz soll helfen verschiedene Workloads besser auszunutzen, indem diese durch verschiedene ECCs angepasst werden. Das heißt konkret, dass ein NAND-Speicher eine Page in verschiedene Segmente partitioniert. Jedes Segment wird dabei durch einen schwächeren ECC geschützt. Beim Lesen einer Seite führt dies dazu, dass sich der Flash-zu-Controller Datentransfer und die Dekodierung des ECC großteils überlappen, da jedes kleine Segment unabhängig dekodiert werden kann. Dies führt zu einer enorm reduzierten Leselatenz.

Der zweite Ansatz soll die Zugriffslatenz für SSDs reduzieren, indem eine P/E-Verzögerung verwendet wird. Laut diesem Verzögerungsschema, sollen laufende P/E-Operationen

unterbrochen werden, um ausstehende Leseoperationen durchzuführen. Danach werden die P/E-Operationen wieder aufgenommen und durchgeführt. Das Konzept wird noch dadurch ergänzt, dass es Schreibaktionen möglich ist Löschaktionen vorzugreifen, um die Schreiblatenz zu verringern.

Beim dritten Konzept soll die Lebenszeit der SSD, durch das Ausnutzen von „Content Locality“, erhöht werden. Wenn eine Page überschrieben wird, so ähneln sich der neue und der alte Inhalt in der Regel häufig. Diesen Umstand macht sich das Prinzip zu nutze, indem beim Überschreiben einer Page nicht der neue Inhalt gespeichert wird, sondern nur der Unterschied (Delta) zu dem alten Inhalt. Es entsteht also eine Art Kompression, die zu einer Reduktion der Schreiboperationen führt.

Empirische Performance-Evaluation von rohen NAND-Flash-Speichern

In [Des10] werden Performance-Tests von rohen Flashchips beschrieben. Dafür wurde ein eigenes Evaluationsboard konstruiert, wobei auch der Flash-Controller selbst programmiert wurde. Das bedeutet, dass die FTL komplett eigens geschrieben wurde und die Ergebnisse maßgeblich von dieser Tatsache abhängen. Diese wurde sehr einfach und ohne Haltbarkeits- oder Performancemaßnahmen wie Garbage Collection oder Wear Leveling implementiert. Das Ziel des Experiments war es, die Haltbarkeit des rohen Flash-Speichers, bei der konstanten Belastung durch hohe P/E-Cycles, zu erforschen. Auch die Performanceänderung im Bezug auf den Wear-Zustand sollte erforscht werden. Letzteres Kriterium erscheint sehr sinnvoll und soll auch in den eigenen Tests dieser Arbeit beachtet werden. Die Methodik in [Des10] erscheint jedoch nicht ausgereift genug, um valide Ergebnisse zu erzielen. Die P/E-Cycles wurden auf folgende Weise erzielt: Es wurde immer eine Page innerhalb eines Blocks auf nur Nullen programmiert (also ein einfacher Bitflip) und darauf folgend wurde der gesamte Block gelöscht. Es wurde also bei der Programmierung nur eine Page gestresst und beim Löschvorgang muss folglich auch nur eine Page gelöscht werden. Da dieser Löschvorgang über mehrere Pulse auf die Zellen ausgeführt wird und die nicht programmierten Zellen sofort den richtigen Zustand haben, ist diese Art von Zellenbeanspruchung nicht realitätsnah. Folglich ergeben sich in dieser Arbeit auch bis zu 100 mal höhere Werte für die Ausdauer der Speicher. Die Performance im Bezug auf die Abnutzung war im Vergleich zu den Herstellerangaben anfangs niedriger und sank mit fortschreitender Belastung.

Flash-Performance bei Datenbankanwendungen

In [OBO+13] werden Performanceanalysen, speziell für die Nutzung von eingebetteten Datenbanken (SQLite) auf Flash-Speichern, durchgeführt. Dabei ist das Zusammenspiel von Datenbank und Flash File System (FFS) besonders im Fokus. Auf die verschiedenen FFSs wird in Kapitel 4 noch näher eingegangen. Es wird eine Methodologie konzipiert, die den Einfluss von Datenbankanfragen auf die Performance und Lebensdauer von Flash-Speichern, ermitteln soll. Gleichzeitig werden dabei zwei verschieden FFSs verglichen. Abbildung 3.7 zeigt die konzipierte Evaluationsmethodologie. Auf Hardwareseite wurden zwei verschieden Testboards verwendet, um zu zeigen, dass sich die Ergebnisse auf verschiedenen Plattformen reproduzieren lassen. Auf Softwareebene wurde ein C-Tool geschrieben, das die SQLite-Application Programming Interface (API) nutzt, um Datenbankanfragen zu generieren. Parallel wird ein Monitoring-Tool (Flashmon) für die Messung der erzeugten Zugriffszeiten verwendet. Dieses Tool ermöglicht einen präzisen Einblick zu bekommen, wie viele I/O-Requests

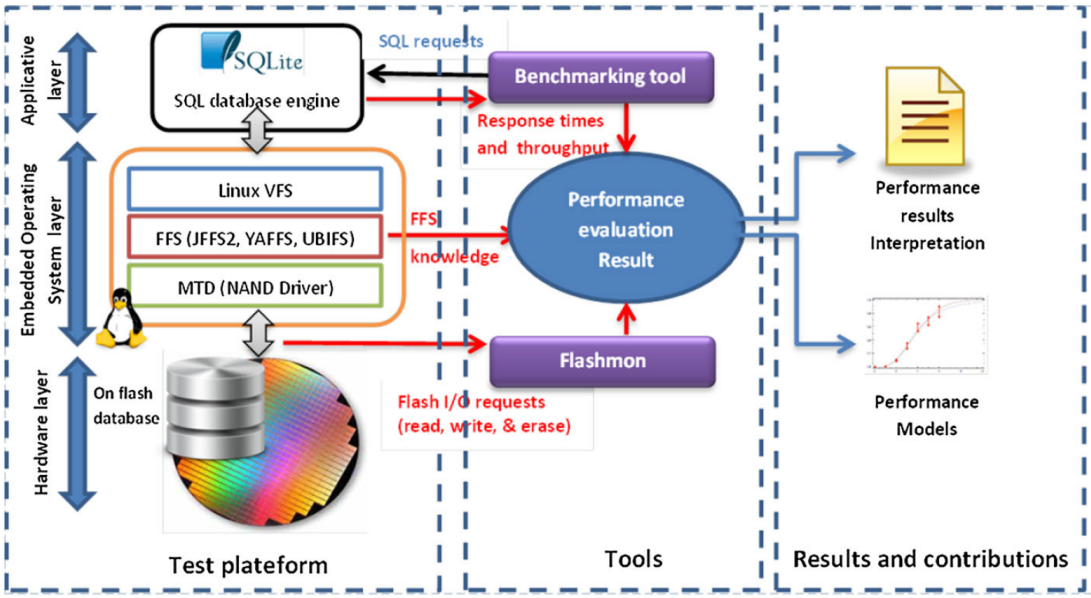


Abbildung 3.7.: Methodologie zur Performance-Evaluation von eingebetteten Datenbanken auf Flash-Speichern

für eine bestimmte Anfrage tatsächlich produziert werden. Bewertungskriterien sind also zum einen Zugriffszeiten und zum anderen die Anzahl an produzierten I/O-Operationen. Es ergibt sich folglich ein Kostenmodell, das theoretische Kosten einer Datenbankanfrage mit den effektiven (tatsächlichen) Kosten in Beziehung bringt. Dieses Kostenmodell ist im Prinzip eine auf Datenbankanfragen spezialisierte Version des WAF, bei dem vom Host gesendete Schreiboperationen (theoretische Kosten) mit den tatsächlich auf dem NAND geschriebenen Operationen (tatsächliche Kosten) ins Verhältnis gesetzt werden.

Als Ergebnis wurden mehrere Tendenzen aufgezeigt. Verschiedene Datenbankoperationen (*insert, select, join und update*) erzeugen jeweils verschiedene Performance-Muster, wobei sich nur *select* und *join* in ihrem Performance-Muster ähneln. Des Weiteren hängt die Performance von der Anzahl und der Größe der Datenbankoperation ab. Auch das FFS hat einen erkennbaren Einfluss auf die Performance, indem verschiedene Anzahlen an I/O-Operationen und Ausführungszeiten für die selben Datenbankoperationen entstehen. Zusätzlich beeinflusst die Datenbankstruktur, die vom Datenbank-Management-System (DBMS) erzeugt wird, die Performance, da dessen interne Fragmentierung der Daten verschieden gut mit der Page-Größe des Flash-Speichers zusammen harmoniert. Auf Abbildung 3.8 sind beispielhaft Performanceergebnisse für die *insert*-Operation auf den beiden verwendeten Hardwareplattformen dargestellt. Hier werden die beiden FFSs JFFS2 und UBIFS im Bezug auf I/O-Durchsatz verglichen, wobei die Anfragenanzahl, die Datenbankeintraggröße und die Plattform (Grafiken links und rechts) variiert werden. Geschlussfolgert wird dabei, dass der I/O-Durchsatz für beide Filesysteme bei Datenbankanfragen im Vergleich zu einfachen Schreibaktionen relativ gering ist.

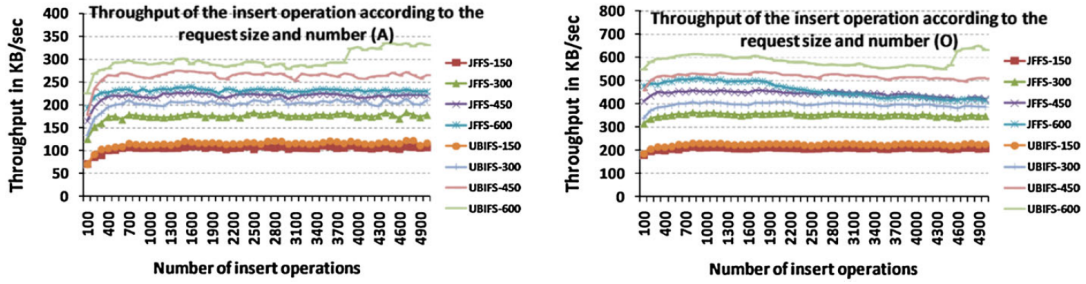


Abbildung 3.8.: I/O-Durchsatz (KB/s) bei *insert*-Operationen im Zusammenhang mit Datenbankeintraggröße und Operationenanzahl auf zwei verschiedenen Hardwareplattformen (Grafiken links und rechts)

3.4. Zusammenfassung

Als Grundlage für die Testmethode dieser Arbeit wird die Testspezifikation der SNIA verwendet. Sie legt die aufzuzeichnenden Performance-Metriken fest (IOPS, Durchsatz, Latenz) und die dafür durchzuführenden Tests fest. In der Beschreibung der Tests inklusive Pseudo-Code sind bereits folgende Einflussfaktoren einbezogen:

- Zugriffsmuster
- Datenmuster
- Outstanding I/O-Operations
- Workload
- Aktiver Bereich

Außerdem werden die Angaben zur Testumgebung und den zu erstellenden Berichten in die eigen Testmethode einfließen. In der ergänzenden Literatur wurden die Themen WAF, Temperatur, Stromverbrauch, P/E-Cycling und Zugriffsmodus des Betriebssystems angesprochen. In Kapitel 4 werden dann alle Einflussfaktoren zu einer eigenen Methode inkorporiert.

4. Messgrößen und Messmethode

In diesem Kapitel wird das vollständige Testkonzept erarbeitet. Dazu werden zuerst die Komponenten Diagnose und Konfiguration für die beiden JEDEC-Flash-Standards eMMC und UFS analysiert. Darauf folgend soll die Basis der Testmethodik, die Testspezifikation der SNIA (siehe 3.1) mit ihren grundlegenden Komponenten wiederholt werden. Anschließend wird erläutert, welche Anpassungen an das automotive Umfeld diese Methodik ergänzen sollen. Schließlich soll das vollständige Testkonzept, bestehend aus Messgrößen und Messmethode, zusammengefasst dargestellt werden.

4.1. Diagnose und Konfiguration

4.1.1. Informationskategorisierung

Der Begriff der Diagnose kann im Zusammenhang mit dieser Arbeit mehrdeutig verwendet werden. Um die einzelnen Bedeutungen richtig erläutern zu können, muss zuerst geklärt werden, welche Art von Diagnoseinformationen Flash-Speicher (eMMC und UFS) zur Verfügung stellen. Auf Abbildung 4.1 wird ein Überblick über die verschiedenen Informationsarten gegeben. Dabei spielen zwei Kriterien eine wichtige Rolle:

- Dynamik
- Zugriffsebene

Mit Dynamik ist gemeint, ob die ausgelesene Information unveränderlich (statisch) oder während der Lebensdauer des Speichers veränderlich (dynamisch) ist. Bei der Zugriffsebene wird unterschieden, ob eine Information nur gelesen oder geschrieben werden kann, oder ob sogar beides möglich ist.

Anhand dieser Kriterien lassen sich drei wichtige Informationsarten festlegen, welche sich wiederum durch semantische Kategorien weiter unterteilen lassen:

- Statische lesbare Informationen
- Dynamische lesbare Informationen
- Dynamische les- und schreibbare Informationen

Die wichtigste Kategorie für diese Arbeit bilden die statischen lesbaren Informationen. Das erste Einsatzgebiet dieser Eigenschaften ist die Identifikation des Speichers (DUT). Hier können beispielsweise Hersteller und Seriennummer ausgelesen werden, um sogar Speicher der selben Bauart untereinander unterscheiden zu können. Dies ist wichtig, falls die Validität eines Tests an mehreren Speichern der gleichen Bauart sichergestellt werden soll. Das zweite semantische Einsatzgebiet dieser Informationen ist das ermitteln von Messgrößen via Diagnose. Dabei kann entweder die Unterstützung einer bestimmten Funktion überprüft

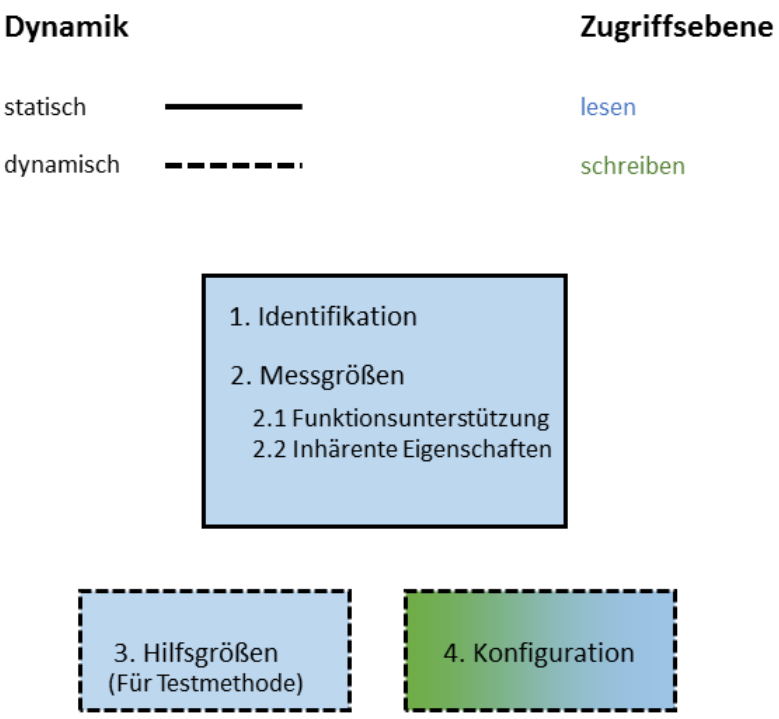


Abbildung 4.1.: Kategorisierung von Diagnoseinformationen

(z.B. Unterstützung eines speziellen Sicherheitsfeatures), oder eine dem Speicher inhärente Eigenschaft überprüft werden (z.B. Gesamtkapazität in Byte).

Die dynamischen lesbaren Informationen dienen nur einem einzigen Zweck. Sie werden als Hilfsgrößen verwendet, um bei den Performancetests bestimmte Metriken zu ermitteln (berechnen). Ein Beispiel wäre die Berechnung des WAF siehe 4.3.

Die letzte Kategorie an Informationen, die dynamischen les- und schreibbaren Informationen, dienen dem Zweck der Speicherkonfiguration. Das bedeutet gewisse Eigenschaften (z.B. der Speicher-Cache) werden vor dem Durchlaufen der Performancetests so konfiguriert, dass alle für die Testmethode wichtigen Einstellungen richtig gesetzt sind und vor allem bei allen DUTs gleich sind. Die Konfiguration kann allerdings zusätzlich auch über bestimmte Aktionen durchgeführt werden. Beispielsweise kann ein Purge durchgeführt werden oder eine Art Partitionierung vorgenommen werden. Damit diese Aktionen korrekt durchgeführt werden können, müssen teilweise auch wieder Informationen der anderen Kategorien ausgelesen werden. Ein Beispiel wäre hier die Partitionierung des Speichers auf den vollen Nutzerbereich. Dafür müsste zuvor die Größe des Nutzerbereichs ausgelesen werden. Die Konfiguration führt folglich zu sinnvollen und vergleichbaren Testergebnissen.

4.1.2. eMMC

Im Prinzip können nach der Kategorisierung beliebig viele Eigenschaften jeder Kategorie ausgewählt und verwendet werden. Da jedoch nicht alle Eigenschaften eine sinnvolle Information für diese Arbeit liefern und es den Rahmen dieser Arbeit sprengt alle Eigenschaften der beiden Speichervarianten zu erläutern, wird für beide Speichervarianten gezeigt, wel-

che grundsätzlichen Quellen die Informationen zur Verfügung stellen. Anschließend wird in Sektion 4.4 ein sinnvolles Subset an Eigenschaften ausgewählt, das auch später bei der Implementierung des Prototypen (siehe 5) verwendet werden kann.

Der eMMC stellt als Informationsquelle sieben Register zur Verfügung[JED15]:

- Operation Conditions Register (OCR): enthält Spannungsprofil des Speichers
- Device Identification Register (CID): enthält Informationen zur Speicheridentifizierung
- Device Specific Data Register (CSD): enthält diverse Speichereigenschaften
- Extended CSD Register (EXTCSD): Erweiterung des CSD-Registers
- Relative Device Adress Register (RCA): enthält vom Host erhaltene Speicheradresse
- Driver Stage Register (DSR): enthält elektrische Eigenschaften der einzelnen Busleitungen
- Queue Status Register (QSR): enthält den Status der internen Warteschlange (Queue)

Nur drei dieser Register sind für diese Arbeit wirklich von Interesse: CID,CSD und EXTCSD. Wirft man noch einmal einen Blick auf die Abbildung 4.1, können nun die semantischen Kategorien den Registern zugeordnet werden. Das CID liefert ausschließlich Informationen zur Identifikation. Sowohl CSD als auch EXTCSD decken dabei alle verbleibenden Kategorien ab, d.h. Messgrößen, Hilfsgrößen und Konfigurationseigenschaften. Beide Register besitzen einen Anteil an nur lesbaren Informationen, sowie einen schreibbaren Teil, der zur Speicherkonfiguration verwendet wird. Einige Hilfsgrößen können jedoch nicht ohne weiteres ausgelesen, bzw. interpretiert werden. Im EXTCSD gibt es einen Bereich der für den Hersteller reserviert ist, um dort seine eigens formatierten Informationen zur Speicherabnutzung und Lebensdauer abzulegen. Das bedeutet welche Informationen (konkret Felder) in diesem Bereich zur Verfügung gestellt werden, ist komplett dem Hersteller überlassen. Deshalb ist es notwendig vom Hersteller sogenannte „Application Notes“ zu bekommen. In diesen wird beschrieben welche Felder in dem reservierten Bereich vorhanden sind und vor allem, wie die jeweiligen Werte zu interpretieren sind.

4.1.3. UFS

Beim UFS gestalten sich die Informationsquellen etwas vielfältiger im Vergleich zum eMMC. Das Pendant zu den eMMC-Registern sind hier die UFS-nativen Deskriptoren, Attribute und Flags. Jedoch kommen noch native SCSI-Befehle hinzu (z.B. READ BUFFER)[JED].

Der Unterschied zwischen Deskriptoren ist, abgesehen vom Inhalt der Felder, die Größe dieser Daten-Container:

- Deskriptor: großer Block an Parametern; z.B. 512 Byte
- Attribut: einzelner Parameter; 1-32 bit
- Flag: einzelner Parameter; 1 bit

Bei der genaueren Analyse hat sich gezeigt, dass Attribute und Flags Parameter enthalten, die eine Host-Applikation für die Kommunikation und Steuerung des Speichers benutzt. Das bedeutet diese beiden Informationsquellen passen nicht in eine der, in dieser Arbeit definierten, Informationskategorien. Folglich werden von den UFS-nativen Quellen lediglich die Deskriptoren für die Diagnose verwendet.

Es gibt insgesamt neun Arten von Deskriptoren:

- Device: Hauptdeskriptor; diverse Eigenschaften des Gesamtspeichers
- Unit: Pro LU ein Deskriptor mit LU-spezifischen Eigenschaften
- Interconnect: Versionen der untersten Verbindungsebene (MIPI M-PHY, UniPro)
- String: Diverse Deskriptoren mit jeweils einer Information (z.B. Hersteller)
- Geometry: Geometrische (Hardware-)Eigenschaften
- Power: Informationen zur elektrischen Leistung und Energiestatus
- Health: Informationen zum Abnutzungszustand des Speichers
- Configuration: Konfiguration von Device und LUs

Von dieser Vielzahl an Deskriptoren werden wiederum nur vier für diese Arbeit genutzt. Die String-Deskriptoren werden ausschließlich für die Identifikation des Speichers ausgelesen. Sowohl Device- als auch Geometry-Deskriptoren werden für Mess- und Hilfsgrößen verwendet. Der Configuration-Deskriptor wird schließlich für die Konfiguration des Gesamtspeichers und der einzelnen LUs verwendet.

Das Thema Hilfsgrößen muss jedoch auch für den UFS noch genauer betrachtet werden. Anders als beim eMMC gibt es für die herstellerspezifischen Eigenschaften keinen fest definierten Platz (Deskriptor, etc.). Stattdessen kann der Hersteller eine beliebige Methode zur Ermittlung dieser Informationen verwenden. Das heißt konkret, dass Hersteller A einen zusätzlichen Device Health Deskriptor verwenden kann, während Hersteller B eine der SCSI-nativen Methoden zur Informationsermittlung nutzt (z.B. READ BUFFER). Dies zeigt, dass die JEDEC bei dem neueren Standard UFS mehr Freiraum für den Hersteller geschaffen hat, wobei das Konzept eines einheitlichen Standards dabei geschwächt wird. Gerade im Hinblick auf Software, die Speicher verschiedener Hersteller nutzen muss, ergeben sich dadurch zusätzliche Mehraufwände, da jeder Hersteller einzeln implementiert werden muss. Eine Folge ist möglicherweise, dass sich Treiber beispielsweise nur noch auf den minimalen Konsens beschränken und die wertvollen Zusatzfunktionen der einzelnen Hersteller, die einen Mehrwert für Applikationen bieten können, nicht implementieren werden. Folglich können Hilfsgrößen durch sämtliche Methoden zur Verfügung gestellt werden, wobei sich bei der Recherche verschiedener Speicherhersteller ergeben hat, dass die wichtigsten Quellen die genannten Deskriptoren sowie die beiden SCSI-Befehle „READ BUFFER“ und „INQUIRY“ sind. Bei der Konfiguration kommt noch ein wichtiger Befehl hinzu, nämlich der SCSI-Befehl „FORMAT UNIT“. Mit diesem wird bei Verwendung der richtigen Parameter der Speicher komplett gelöscht und somit ein Purge durchgeführt.

Kurz zusammengefasst zeigt sich, dass beide Speicher die selbe Art der Information anbieten. Dabei fällt auf, dass UFS bei herstellerspezifischen Informationen loser definiert wurde als eMMC und es somit bei der Implementierung zu Mehraufwand, aufgrund hoher Variation

	eMMC	UFS
Identifikation	CID	String-Deskriptoren
Funktionsunterstützung	CSD, EXTCSD	Device-, Geometry- Deskr.
Inhärente Eigenschaften	CSD, EXTCSD	Device-, Geometry- Deskr.
Hilfsgrößen	CSD, EXTCSD	Deskriptoren, SCSI-Befehle
Konfiguration	CSD, EXTCSD	Config-Deskriptor, SCSI-Format

Abbildung 4.2.: Kategorien von Diagnoseinformationen mit ihren jeweiligen Quellen für UFS und eMMC

unter den Herstellern kommen kann. Tabelle 4.2 zeigt jeweils für UFS und eMMC für jede Informationskategorie die dazugehörigen Informationsquellen.

4.2. Messgrößen und Messmethode laut SNIA

In diesem Abschnitt soll wiederholt werden, welche Komponenten der PTS der SNIA für die Arbeit übernommen werden können[Sto15].

4.2.1. Performance Metriken

Aus der PTS gehen drei verschiedene Performance Metriken hervor:

- IOPS
- Durchsatz
- Latenz

Warum diese wichtig sind, kann in Sektion 3.1 nachgelesen werden. Bei den IOPS wird gemessen, wie viele Ein- und Ausgabeoperationen pro Sekunde übertragen werden. Dabei wird nicht die Datenrate überprüft, sondern nur eine Anzahl an Operationen. Die Größe der Datenpakete muss beachtet werden, jedoch gibt die Metrik an sich keine Angabe zur Datenrate. Der Durchsatz zeigt im Gegensatz zu den IOPS die tatsächliche Datenrate (z.B. in MB/s). Die Latenz gibt die Zeit wieder, die eine einzelne I/O-Operation vom Anfordern des Hosts an den Speicher bis zur Rückantwort des Speichers, benötigt. Im Prinzip kann die Gesamtlatenz in einen Anforderungs-, einen Verarbeitungs- und einen Antwortanteil unterteilt werden. Bei der Methodik der PTS soll dabei die Gesamtlatenz gemessen werden.

4.2.2. Workload Parameter

Die PTS spezifiziert einige Einflussgrößen auf die Performance, die als Parameter für die zu erzeugende Workload konfiguriert werden.

Als erstes wird festgelegt, dass grundsätzlich IO auf Block- oder Device-Level erzeugt wird. Das bedeutet, es wird kein Filesystem auf dem Speicher verwendet. I/O-Operationen werden somit direkt an die logischen Blockadressen des Speichers adressiert. Damit wird verhindert, dass der I/O-Stack des Betriebssystems einen zu großen Einfluss auf die Testergebnisse hat, da einige Komponenten des Stacks damit umgangen werden.

Das Zugriffsmuster ist auch Teil der PTS. Hierbei werden sequentielle und wahlfreie Zugriffe, sowie verschiedene Blockgrößen und Schreib-Lese-Mischverhältnisse unterschieden.

Auch das Datenmuster spielt eine wichtige Rolle. Dieses wird für alle Tests auf zufällige Daten bestimmt. Feste Datenmuster könnten unerwünschte Effekte in den Speichern hervorrufen, die einen objektiven Vergleich erschweren würden.

Die OIOs müssen auch für jeden Test festgelegt werden. Dabei werden die Threadanzahl und die Tiefe der Warteschlange (Queue Depth) festgelegt.

Der letzte Parameter ist die Active Range. Diese soll laut PTS auf 100% gesetzt werden, um das Problem des Überprovisionierung zu verhindern. Dabei wird ein nicht genutzter Bereich des Speichers als Pufferspeicher genutzt. Somit würden bei einer Active Range unter 100% wieder unerwünschte Effekte im Speicher eintreten, die die objektive Betrachtung der Ergebnisse erschweren würde.

4.2.3. Testablauf

Der grundsätzliche Testablauf pro Einzeltest (siehe Abbildung 3.3) ist in vier Phasen aufgeteilt:

- Purge
- Preconditioning
- Teststimulus
- Sammeln und Aufbereiten von Ergebnisdaten

Grund dieses Aufbaus ist es eine einheitliche Methode zu definieren, die sicherstellt, dass die Ergebniswerte im Steady State gemessen werden. Dazu wird der Speicher zuerst im *Purge* komplett geleert. Praktisch bedeutet dies, dass alle Flash-Einzelzellen auf ihren initialen (Spannungs-)Wert zurückgesetzt werden.

Danach folgt ein Test-unabhängiges *Preconditioning*, bei dem der Speicher in großen Datenblöcken (1024 KB) vollgeschrieben wird. Dies soll auf einen Zustand vorbereiten, bei dem die Firmware des Speichers nun Blöcke löschen muss, bevor wieder erneut auf den Speicher geschrieben werden kann und somit realistischere Performancewerte gemessen werden können.

Im *Testsegment* wird der tatsächliche Teststimulus (Testschleife) so lange laufen gelassen, bis sich der Speicher nachweislich im Steady State befindet oder 25 Runden vergangen sind, um dann in diesem Zustand die Messwerte abzugreifen. Innerhalb einer Testrunde wird für jeden Lese-Schreib-Mix (z.B. 100% Lesen und 0% Schreiben) über alle Blockgrößen, die für diesen Test spezifiziert sind, iteriert und jeweils für jede Paarung ein 60 Sekunden Teststimulus ausgeführt. Zur Überprüfung des Steady State werden eine oder mehrere dieser Kombinationen, auch Testvariablen genannt, über die letzten fünf Runden auf die in 3.1.3 beschriebenen Steady State (SS)-Kriterien überprüft. Sobald alle Testvariablen im SS sind oder 25 Runden vergangen sind, wird der Test abgeschlossen.

Im Anschluss werden die gemessenen Daten in den definierten Plots aufbereitet, um die Ergebnisse interpretieren zu können.

4.2.4. Tests

Die PTS definiert vier Performance Tests:

- IOPS
- Durchsatz
- Latenz
- Write Saturation Test (WSAT)

Die ersten drei Tests (IOPS, Durchsatz, Latenz) folgen alle dem vorher erläuterten grundsätzlichen Testablauf. Sie unterscheiden sich durch die Wahl der Testparameter und der gemessenen Performance Metriken. Außerdem wird beim Latenztest nach der Testschleife noch ein zusätzlicher 20 Minuten langer Teststimulus gesetzt. Über diesen Zeitraum werden konstant die Latenzzeiten abgegriffen und es soll als Ergebnis ein Latenzhistogramm erstellt werden. An diesem kann erkannt werden, welche Latenzzeit einem Wert von „Five Nines“ also zu deutsch fünf Neunen entspricht. Dieser Wert gibt an, welche Latenzzeit von 99,999% der I/O-Operationen unterschritten wird. Es ist also eine Absicherungsmetrik, falls in einer Applikation ein gewisser Wert nicht überschritten werden darf (z.B. Real-Time-Applikationen).

Der WSAT ist etwas anders strukturiert. Initial wird ein Purge durchgeführt. Danach wird durchgehend für sechs Stunden mit 4KB großen Blöcken auf den Speicher geschrieben. Dadurch soll sich zeigen, wie sich die Performance im maximalen Belastungsfall über die Zeit hin verändert. Bei der Veränderung über die Zeit können mehrere Faktoren eine Rolle spielen, wovon der wichtigste der Übergang in den SS darstellt.

4.2.5. Testergebnisse

Sind alle Ergebnisse gemessen, so müssen diese in den spezifizierten Plots dargestellt werden. Dabei werden drei Arten von Plots unterschieden:

- SS-Annährungsplot
- SS-Verifikationsplot
- Messplots

Der SS-Annährungsplot zeigt, wie sich die Testvariablen über alle gemessenen Runden hinweg verändern, also wie diese sich an den SS annähern. Hiermit kann gezeigt werden, wie stabil die Performancemetrik über die Runden hinweg war. Besonders, wenn der SS nicht erreicht wurde ist dieser Plot interessant, weil er zeigt bei welcher Testvariable Abweichungen in den SS-Kriterien bestehen.

Der tabellarische Verifikationsplot gehört zu dem Annäherungsplot und zeigt anhand konkreter Werte, wie der SS erreicht wurde. Er enthält z.B. den Durchschnittswert der Testmetrik sowie die erlaubte und tatsächliche Abweichung vom Durchschnittswert. Es wird also anhand dieses Plots sichergestellt, dass der Annährungsplot valide Ergebnisse zeigt.

Die Messplots sollen die gemessene Testvariable in den Runden des SS zeigen. Dabei werden die Werte in Tabellen mit konkreten Werten sowie in Plots mit 2D-Graphen und Balkendiagrammen gezeigt.

4.2.6. Testumgebung

Zur Testumgebung wird in der PTS nicht definiert, welche Hardwareplattform verwendet werden soll. Es wird lediglich eine auf Linux basierende Referenzplattform präsentiert. Zur Softwareplattform allerdings werden klare Anforderungen definiert, sowie Referenztools vorgeschlagen. In Sektion 3.1.3 werden die Anforderungen klar beschrieben. Die Recherche während dieser Arbeit ergab, dass ein Open Source Software-Tool für Linux alle Anforderungen abdeckt. Dieses Benchmark Tool heißt *FIO* (Flexible I/O Tester) und wurde von Jens Axboe geschrieben, der unter anderem Hauptentwickler der Linux Block Layer und der Block-Geräte unter Linux ist[1]. Es kann also für diese Arbeit als Workload Generator verwendet werden, weil es sowohl die Workload-spezifischen Anforderungen, als auch die Anforderungen an den Zufallsgenerator (für zufällige Datenmuster und zufällige Adressen) erfüllt.

4.3. Anpassungen an das automotive Umfeld

Da die Methodik der PTS noch ergänzt werden muss, um eine Eignungsfeststellung für die Automobilbranche festzustellen, werden nun im Folgenden die zusätzlichen Anpassungen erläutert.

4.3.1. Write Amplification Factor

Aus Kapitel 3.2 gingen einige Erkenntnisse über den WAF hervor. Es handelt sich dabei um keine direkte Performance-Metrik, sondern vielmehr um eine Effizienz-Metrik. Die zusätzlichen Aktionen können sowohl für eine verbesserte Performance (Garbage Collection) als auch für eine erhöhte Lebensdauer (Wear Leveling) verwendet werden. Der WAF gibt also einen Einblick, wie „aktiv“ die FTL generell ist. Ist diese *zu* aktiv (zu ineffizient), so wird der Speicher unumgänglich zu stark abgenutzt. Gemessen wird das Verhältnis von Host-Schreibaktionen zu tatsächlich auf dem physikalischen NAND-Array durchgeführten Schreibaktionen an. Der WAF stellt ein klares Anforderungskriterium in Speicherlastenheften dar und sollte somit auf jeden Fall bestimmt werden, da gerade im Automobilbereich Langlebigkeit (ca. 15 Jahre) enorm wichtig ist. Wird der Wert bestimmt und die maximalen P/E-Zyklen eines Speichers in Erfahrung gebracht (muss der Hersteller angeben), so kann grob abgeschätzt werden, ob die erwünschte Lebensdauer des Speichers erzielt werden kann.

In Kapitel 3.2 wurde erläutert, dass 100% wahlfreie Zugriffe den selben WAF erzielen, wie spezielle reale Testpatterns (z.B JEDEC). Deswegen bietet es sich an, den Teststimulus des IOPS-Tests für diesen Test zu verwenden. Beeinflusst wird der WAF jedoch nur von Schreibaktionen, weshalb die Lese-Schreib-Verhältnisse, die Leseaktionen enthalten, weggelassen werden können. Damit der Test im SS des Speichers beginnt, soll er nach dem IOPS-Test ausgeführt werden, ohne eigenes Preconditioning. Der WAF-Test wird also an den IOPS-Test gekoppelt.

[1] Informationen zu FIO auf der offiziellen Github-Webseite https://github.com/axboe/fio/blob/master/README Stand 11.07.2018

4.3.2. Stromverbrauch

Der Stromverbrauch von Speichern in eingebetteten Systemen kann aus mehreren Gründen eine sehr wichtige Rolle spielen. Die elektrische Leistung im Leerlauf ist wichtig, da der Speicher sozusagen in Leerlaufzeiten Strom „sparen“ kann. Eine besonders niedrige Leistung ist hier von Vorteil. Bei einem Mobiltelefon beispielsweise ist dies essentiell. Wird hier zu viel Strom in der Zeit, in der das Telefon nicht genutzt wird, verbraucht, so wird der Akku sehr schnell leer. Das Automobil stellt grundsätzlich eine vielseitigere Umgebung für eingebettete Speicher dar, da es zu mehreren Versorgungsszenarien kommt. Während der Fahrt eines Autos mit Verbrennungsmotor wird die Batterie von einer Lichtmaschine gespeist, einer Art Stromgenerator, der durch die mechanische Leistung des Motors Energie für das Bordstromnetz erzeugt. Es würde sich also die Batterie nicht aufgrund dieser sehr kleinen Verbraucher (Motorleistung verglichen mit elektrischer Leistung eines eingebetteten Speichers) entleeren. Steht jedoch das Auto und ist in einem Park-Modus, heißt das, dass die Zündung an ist und ein großer Teil der Elektronik noch mit Strom versorgt werden muss. In diesem Fall kann auch hier ein zu hoher Stromverbrauch die Batterie leeren. Eine weitere wichtige Rolle spielt der erhöhte Benzinverbrauch des Autos bei erhöhter elektrischer Leistung der Verbraucher (z.B. Speicher). Wird zum Beispiel die Klimaanlage im Auto angeschaltet, so steigt der Benzinverbrauch. Die Lichtmaschine benötigt zusätzliche Leistung und diese wird durch eine mechanische Mehrleistung des Motors kompensiert. Es folgt also ein höherer Benzinverbrauch. Da im Verlauf dieser Arbeit festgestellt wurde, dass in Zukunft immer mehr an Speicherkapazität im Automobil benötigt wird, muss die elektrische Leistung dieser vielen kleinen Komponenten beachtet werden. Ein weiterer Grund, die elektrische Leistung möglichst gering zu halten, ist die Tatsache, dass nur ein gewisses Kontingent an Leistung für einen bestimmten Bereich im Auto zur Verfügung steht. Die Lichtmaschine kann also nur eine limitierte Menge an elektrischer Leistung zur Verfügung stellen. Wird diese Leistung von vielen Speichern aufgebraucht, so ergeben sich Engpässe für andere Komponenten. Dies muss dann mit einer größeren und teureren Batterie kompensiert werden. Bei einem Elektro- oder Hybridfahrzeug verkürzt ein zu hoher Stromverbrauch zusätzlich die Reichweite des Fahrzeugs. Außerdem korreliert eine höhere verbrauchte elektrische Leistung mit einer erhöhten Temperatur des Halbleiterelements. Es werden also sowohl der Speicher, als auch die mit ihm verbundenen Leitungen mehr beansprucht. Praktisch kann die elektrische Leistung über die Spannungsquellen der Speicher bestimmt werden. Um einen aussagekräftigen Überblick über die elektrische Leistung des Speichers zu bekommen, sollen vier verschiedene Szenarien in einem Test bewertet werden:

- Leerlauf
- Lesen
- Schreiben
- Lesen und Schreiben gemischt

Lesen und Schreiben sollte gesondert betrachtet werden, um Aufschluss zu bekommen, welches Leistungsprofil beispielsweise eine stark schreib- oder leselastige Applikation haben wird. Zusätzlich soll noch eine Art Extremfall geprüft werden. Hier soll wieder der IOPS-Teststimulus verwendet werden, da die verschiedenen Kombinationen an Blockgrößen und Lese-Schreib-Verhältnissen alle Beanspruchungen im Hinblick auf die Workload beinhalten.

UFS	eMMC
User Capacity	User Capacity
Max Number Logical Units	Cache Size
Max Number Secure Write Protected Areas	Max Size Replay Protected Memory Block (RPMB)
Max Size of System Code Area	
Max Size of Non Persistent Area	

Abbildung 4.3.: Katalog Funktionsunterstützung

4.3.3. Temperatur

Im Automobilbereich werden Speicher für bestimmte Temperaturbereiche spezifiziert (z.B. -40 bis 80 Grad). Hierbei handelt es sich um eine klare Qualitäts- und Zuverlässigkeitsanforderung. Es kann allerdings auch untersucht werden, ob sich die Temperatur auch auf die Funktion, sprich auf seine Performance auswirkt. Deshalb sollen die Testreihen für verschiedene Temperaturen durchgeführt werden. Das bedeutet konkret, dass die Tests bei der niedrigsten spezifizierten, der höchsten und bei Zimmertemperatur durchgeführt werden sollen, um hier mögliche Performanceschwankungen zu erkennen. Das heißt es muss festgestellt werden, ob durch den Temperatureinfluss die konkrete Performancemetrik unter den vorgeschriebenen Wert fällt und somit in einem gewissen Bereich dem Lastenheft nicht gerecht wird.

4.4. Zusammenfassung

In dieser Sektion soll die gesamte Messmethodik mit ihren Komponenten zusammengefasst und dargestellt werden.

4.4.1. Messgrößen

Wie in der Einleitung schon angedeutet, soll es einen Katalog an möglichen Messwerten geben, der von der Testplattform abgeprüft werden kann. Dabei soll ein Messwert einer Anforderung eines Speicherlastenheftes entsprechen. Die Diagnosemesswerte sind, wie vorher schon beschrieben, in eine Sektion Funktionsunterstützung und eine Sektion inhärente Informationen aufgeteilt. Es würde den Rahmen der Arbeit sprengen, würde man alle Informationen der JEDEC-Spezifikation, die in die jeweilige Kategorie passen, aufzählen. Deshalb wurde ein sinnvolles Subset an Informationen für den jeweiligen Katalog erstellt.

Auf Abbildung 4.3 ist der Katalog für die Funktionsunterstützung zu sehen. Für den UFS werden unter anderem eine Sicherheitsfunktion (Secure Logical Unit), sowie die Möglichgkeit ein Field Firmware Update durchzuführen geprüft. Beim eMMC wird beispielsweise geprüft, welche Sicherheitsfunktionen zum Löschen zur Verfügung stehen, sowie die Möglichkeit einen Schreibschutz anzuwenden.

Auf Abbildung 4.4 ist der Katalog für die Inhärenten Informationen abgebildet. Beim UFS kann beispielsweise überprüft werden, wie viele LUs maximal konfiguriert werden können oder welche Kapazität an verfügbarem Nutzerbereich der Speicher zur Verfügung stellt. Bei einem eMMC kann ausgelesen werden, wie groß der interne Cache ist oder auch wie groß der verfügbare Nutzerbereich ist.

UFS	eMMC
User Capacity	User Capacity
Max Number Logical Units	Cache Size
Max Number Secure Write Protected Areas	Max Size Replay Protected Memory Block (RPMB)
Max Size of System Code Area	
Max Size of Non Persistent Area	

Abbildung 4.4.: Katalog Inhärente Informationen

TEST	MESSWERTE
IOPS	Für jede Paarung aus Blocksize und RW-Mix ein Zahlenwert (IOPS)
Durchsatz	Für jede Paarung aus Blocksize und RW-Mix ein Zahlenwert (MB/s)
Latenz	Für jede Paarung aus Blocksize und RW-Mix ein Zahlenwert (Durchschnitts-Latenz in msec)
Elektrische Leistung	Pro Beanspruchung (Leerlauf, Lesen, Schreiben, Gemischt) ein Zahlenwert (Elektrische Leistung in mW)
Write Amplification Factor	Ein Zahlenwert für den Write Amplification Factor
Write Saturation (WSAT)	Kein konkreter Messwert

Abbildung 4.5.: Katalog Performance-Messwerte für jeden Test

Abbildung 4.5 fasst stellt dar, welche Werte für jeden Test abgegriffen werden können. Für die PTS-Standardtests IOPS, Latenz und Durchsatz kann jeweils für jede Paarung aus Blockgröße und Lese-Schreib-Verhältnis ein Zahlenwert in der jeweiligen Metrik überprüft werden. Es kann beispielsweise beim Durchsatztest der Wert in MB/s bei einer Blockgröße von 4KB und einem Lese-Schreibverhältnis von 65/35 gemessen werden. Beim Stromverbrauchstest kann für die verschiedenen Teststimuli jeweils die elektrische Leistung in mW gemessen werden. Der WAF wird als Einheitsloser Faktor gemessen. Für den WSAT wird kein konkreter Zahlenwert überprüft. Hier muss der Ergebnisplot analysiert werden, um ein Gefühl für das Performance-*Verhalten* zu bekommen.

4.4.2. Messmethode

In der Messmethode soll nun der gesamte Ablauf noch einmal erläutert werden, wie auf Abbildung 4.6 zu sehen ist. Die noch nicht geklärten Komponenten werden dann in den folgenden Unterkapiteln erläutert. Die Methode beginnt mit einem Input durch den Nutzer. Daraufhin werden die essentiellen Eigenschaften für die Konfiguration ausgelesen und das DUT konfiguriert. Dann folgt die Diagnose mit den zwei Unterteilungen Funktionsunterstützung und Inhärente Informationen. Danach wird die Testreihe, mit allen Teiltests ausgeführt. Als Ergebnis wird dann ein entsprechender Output generiert.

4.4.3. Tests

Zur Wiederholung werden hier noch einmal alle Einzeltests mit ihrer jeweiligen Herkunft aufgeführt:

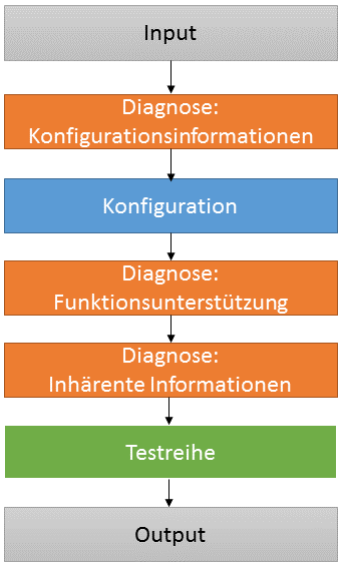


Abbildung 4.6.: Ablauf der Testmethodik

- IOPS (PTS)
- Durchsatz (PTS)
- Latenz (PTS)
- Stromverbrauch (Ergänzung)
- WAF (Ergänzung)
- WSAT (PTS)

Input

Als Input für die Testmethode gilt zum einen das DUT und zum anderen eine Testbeschreibung (Inputdatei). Diese wird in Kapitel 5 konkret implementiert. Die Komponenten dieser Testbeschreibung sind folgende:

- Speichertyp
- Konfigurationsparameter
- Zu überprüfende Funktionen
- Zu überprüfende Inhärente Informationen (mit Erwartungswerten)
- Abzuprüfende Tests mit Testvariablen und Erwartungswerten

	Ergebnisbericht	Setup-Bericht
Gesamte Testreihe	X	X

Abbildung 4.7.: Generierte Berichte für gesamte Testreihe

	Annäherungsplot	2D-Messübersicht	Messdatentabelle	Messplots	WSAT-Plot	Power-Profil-Plot
IOPS	X	X	X	X		
Durchsatz	X	X	X	X		
Latenz	X	X	X	X		
WAF						
WSAT					X	
Strom						X

Abbildung 4.8.: Generierte Plots pro Test

Output

Die Testmethode soll zwei verschiedene Arten an Output generieren. Zum einen den allgemeinen Output, der nur einmal in einer gesamten Testreihendurchführung generiert wird. Zum anderen soll jeder Test individuell Output generieren. Die Abbildungen 4.7 und 4.8 zeigen alle konkreten Outputs (Berichte und Plots), die während einer Testreihe entstehen. Gekapselt sind diese Ergebnisse in einem übergreifenden Testordner, der für eine gesamte Testreihe erzeugt wird. Dieser besitzt dann für jeden Einzeltest einen Unterordner, der wiederum jeweils einen Ordner mit Log-Dateien und einen Ordner mit Plots enthält.

Der allgemeine Output umfasst zwei Berichte: den Ergebnisbericht und den Voraussetzungsbericht. Der Ergebnisbericht soll eine Abbild des Speicherlastenheftes bzw. der Inputdatei mit den zu überprüfenden Eigenschaften sein. Das heißt konkret, dass hier alle Prüfkriterien mit (sofern vorhanden) Erwartungswerten und tatsächlichen Messwerten aufgelistet werden. Zusätzlich soll für jedes Prüfkriterium auch noch gezeigt werden, ob dieses bestanden oder nicht bestanden wurde. Anhand diesen Berichts kann vollständig abgelesen werden, ob die funktionalen Anforderungen eines Speicherlastenheftes erfüllt werden.

Im Voraussetzungsbericht sollen folgende Eigenschaften dokumentiert werden:

- Testplattform (Hardware und Software)
- DUT mit verschiedenen Identifikationsparametern
- Test Setup: pro Test verschiedene Testparameter zum Workload

Dieser Bericht dokumentiert alle wichtigen Vorraussetzungen und Parameter der individuellen Testdurchführung. Somit ist nach dem Test klar, wie die Ergebnisse für welchen Speicher zustande kamen und können somit auch besser mit anderen DUTs verglichen werden.

Der testspezifische Output ist für IOPS-, Latenz- und Durchsatztest relativ ähnlich (siehe Abbildungen 4.7 und 4.8). Zu beachten ist hier, dass die drei PTS-Standard-Tests IOPS, Durchsatz, Laten und WAF die in der PTS spezifizierten Plots generieren müssen. Der in

dieser Arbeit konzipierte WAF-Test generiert keine zusätzlichen Plots. Der Stromverbrauchstest, der auch eigens in dieser Arbeit konzipiert wurde, soll nur ein Balkendiagramm mit den verschiedenen elektrischen Leistungen bei den definierten Teststimuli generieren.

Grundsätzlich sollen auch für alle Tests Log-Dateien gespeichert werden, damit im Nachhinein überprüft werden kann, ob es Probleme bei den Tests gab.

5. Implementierung eines Prototypen

Dieses Kapitel zeigt auf, wie die Testmethode konkret als Software realisiert wurde. Dazu wird zu Beginn ein Input-Datenformat erstellt, gefolgt von einem Befüllen einer Input-Datei mit beispielhaften Daten eines Speicherlastenheftes von BMW. Der letzte Schritt zeigt, wie die verschiedenen Testkomponenten der Testmethode implementiert wurden.

5.1. Erstellung eines Input-Datenformats

Die Input-Datei dient zum einen dazu, den gesamten Ablauf der Testreihe steuern und zum anderen als Abbildung eines Speicherlastenheftes. Das heißt es sollen alle Anforderungswerte als Input in die Datei eingetragen werden können, um daraufhin abgeprüft werden zu können. Im Anschluss können die Ergebnisse dann im Output überprüft werden. Als Basis für das Input-Datenformat wurde für diese Arbeit Extensible Markup Language (XML) ausgewählt. Dieses Format wird in der Softwareentwicklung als Datenformat für verschiedenste Zwecke genutzt, wie beispielsweise für die Datenübertragen bei Webapplikationen oder als Basis von Datenbanken. Aufgrund der großen XML-Community gibt es fundierte Bibliotheken für den Umgang mit XML in diversen Programmiersprachen. Außerdem ist die Darstellung mit den sogenannten „Tags“ (Elemente) selbst-beschreibend und von Mensch und Maschine lesbar. Aus diesen Gründen und auch weil das Format nun in der aktuellen Version schon seit 2008 besteht, wurde sich für XML als Basis-Datenformat entschieden.

Abbildung 5.1 zeigt einen generischen Input für den Prototypen. Das Wurzelelement (äußerstes Element) spezifiziert den Speichertypen (UFS oder eMMC). Die drei Kindelemente spezifizieren die drei Hauptkomponenten der Testmethode:

- Konfiguration
- Diagnose
- Tests

Generell ist der Input so spezifiziert, dass alle Komponenten optional sind. Das heißt, es kann beispielsweise nur die Diagnose durchgeführt werden, falls nur die Werte dieser Komponente für den Benutzer interessant sind. Dazu wird einfach auf das Konfigurations- und das Test-Element verzichtet.

Das Konfigurationselement hat beliebig viele Kindelemente. Diese stehen jeweils für eine Konfigurationseigenschaft. Deren Inhalt spezifiziert jeweils den zu konfigurierenden Wert.

Das Diagnoseelement wird in die zwei bekannten Kategorien Funktionsunterstützung und Inhärente Eigenschaften unterteilt. Diese beiden werden wieder als Kindelemente zum Diagnoseelement eingetragen. Das Funktionsunterstützungs-Element besitzt wiederum beliebig viele Kindelemente. Diese spezifizieren abzuprüfende Funktionen. Besonders behandelt werden Funktionsgruppen. Diese Elemente beinhalten mehrere Gruppenelemente. Die Inhärenten Eigenschaften haben als Inhalt jeweils für jede spezifizierte Eigenschaft einen Erwartungswert.

```
<DeviceType>
    <Config>
        <ConfigField1>Configuration Value</ConfigField1>
        <ConfigFieldN>Configuration Value</ConfigFieldN>
    </Config>
    <Diagnosis>
        <FeatureSupport>
            <Feature1></Feature1>
            <FeatureN></FeatureN>
            <FeatureGroup1>
                <GroupFeature1></GroupFeature1>
                <GroupFeatureN></GroupFeatureN>
            </FeatureGroup1>
            <FeatureGroupN>
                <GroupFeature1></GroupFeature1>
                <GroupFeatureN></GroupFeatureN>
            </FeatureGroupN>
        </FeatureSupport>
        <InherentCharacteristics>
            <Characteristic1>Expected Value</Characteristic1>
            <CharacteristicN>Expected Value</CharacteristicN>
        </InherentCharacteristics>
    </Diagnosis>
    <Tests>
        <IOPS>
            <Read_Write_Mix1>
                <BlockSize1>Expected Value</BlockSize1>
                <BlockSizeN>Expected Value</BlockSizeN>
            </Read_Write_Mix1>
            <Read_Write_MixN>
                <BlockSize1>Expected Value</BlockSize1>
                <BlockSizeN>Expected Value</BlockSizeN>
            </Read_Write_MixN>
        </IOPS>
        <WAF>Expected Value</WAF>
        <PowerProfile>
            <Idle>Expected Value</Idle>
            <Read>Expected Value</Read>
            <Write>Expected Value</Write>
            <Mixed>Expected Value</Mixed>
        </PowerProfile>
        <Throughput>
            <Read_Write_Mix1>
                <BlockSize1>Expected Value</BlockSize1>
                <BlockSizeN>Expected Value</BlockSizeN>
            </Read_Write_Mix1>
            <Read_Write_MixN>
                <BlockSize1>Expected Value</BlockSize1>
                <BlockSizeN>Expected Value</BlockSizeN>
            </Read_Write_MixN>
        </Throughput>
        <Latency>
            <Read_Write_Mix1>
                <BlockSize1>Expected Value</BlockSize1>
                <BlockSizeN>Expected Value</BlockSizeN>
            </Read_Write_Mix1>
            <Read_Write_MixN>
                <BlockSize1>Expected Value</BlockSize1>
                <BlockSizeN>Expected Value</BlockSizeN>
            </Read_Write_MixN>
        <WSAT></WSAT>
    </Tests>
</DeviceType>

```

Abbildung 5.1.: Generischer Input

Das Testelement kapselt alle auszuführenden Tests. Wichtig ist hier, dass das Format sensitiv im Bezug auf die Reihenfolge der Tests ist. Das bedeutet der Test, der als erstes in der Input-Datei eingetragen ist, wird auch tatsächlich als erstes ausgeführt. Es gilt auch wieder, wenn ein Test nicht in die Input-Datei eingetragen wurde, so wird dieser auch nicht ausgeführt.

Die Kindelemente des Testelements spezifizieren die Einzeltests. Dabei werden IOPS-, Latenz- und Durchsatztest auf die gleiche Weise beschrieben. Diese Elemente beinhalten einen oder mehrere Lese-Schreib-Verhältnisse, welche wiederum die abzuprüfenden Blockgrößen mit ihren Erwartungswerten enthalten. Es werden also für jedes Lese-Schreib-Verhältnis alle dafür spezifizierten Blockgrößen mit dem Erwartungswert für das jeweilige Verhältnis abgeprüft.

Das WAF-Element enthält keinen Inhalt, jedoch einen Erwartungswert.

Der Stromtest hat für jeden Teststimulus (Idle, Lesen, Schreiben, Gemischt) ein Kindelement mit dem jeweiligen Erwartungswert als Inhalt.

Das WSAT-Element enthält keinen Inhalt und keine Kindelemente. Nur das Vorhandensein des Elements entscheidet ob der Test ausgeführt wird. Mehr Input kann für diesen Test nicht eingegeben werden.

5.2. Spezifikation von Sollzuständen für BMW

In dieser Sektion soll ein beispielhafter Input für den Prototypen (UFS) gezeigt werden. Dafür wird das Anforderungsprofil eines Infotainmentsystems (von BMW), auch Head Unit genannt, verwendet. Abbildung 5.2 zeigt den Anforderungsabschnitt der Input-Datei mit den spezifizierten Werten.

Die Anforderungen sollen nun erläutert werden. Die persistenten Daten der Head Unit können in drei Bereiche untergliedert werden:

- Ein Bootbereich
- Applikationsbereich
- Nutzer- und Kartendaten

Der Bootbereich soll als SLC konfiguriert werden, da dies generell in einer erhöhten Performance (Latenz, IOPS, Durchsatz) im Vergleich zu einer MLC-Konfiguration resultiert. Es ist besonders wichtig die Boot-Zeit zu minimieren, um ein bestmögliche User-Experience zu ermöglichen. Die Möglichkeit eine LU mit SLC zu konfigurieren wird mit dem „Enhanced Memory Type 1“ abgeprüft. Der Applikationsbereich muss auch so konfiguriert werden, dass hoch-performante Zugriffe ermöglicht werden. Da dieser Bereich relativ groß im Vergleich zum Boot-Bereich ist, ist eine Konfiguration als SLC zu „teuer“. Eine andere Lösung ist es den Bereich als *System Code* zu konfigurieren. Dies ist ein UFS-Memory Type, der für ausführbaren Code vorgesehen ist (z.B. Binärdateien). Im Bezug auf die Performance sind die Anforderungen beim Nutzer- und Kartenbereich relativ gering. Hier hat die Gesamtkapazität des Bereichs die höchste Priorität. Deswegen soll der Bereich auf den „Normal Type“ konfiguriert werden. Dies resultiert in einer MLC- oder TLC-Konfiguration. Es ist zu beachten, dass die *Memory Types* nicht exakt von der JEDEC spezifiziert sind, d.h. beispielsweise dass nicht jeder Hersteller für eine SLC-Konfiguration den Typ „Enhanced_Type1“ verwenden muss. Es hat sich jedoch bei den praktischen Tests im Rahmen dieser Arbeit gezeigt,

```
<Diagnose>
    <FeatureSupport>
        <FFU></FFU>
        <MemoryTypes>
            <NormalType></NormalType>
            <SystemCode></SystemCode>
            <Enhanced_Type1></Enhanced_Type1>
        </MemoryTypes>
    </FeatureSupport>
    <InherentCharacteristics>
        <UserCap>30112318411</UserCap>
        <MaxLU>3</MaxLU>
        <SysCodeMaxSize>5866633900</SysCodeMaxSize>
    </InherentCharacteristics>
</Diagnose>
<Tests>
    <IOPS>
        <_100_0>
            <_4>10000</_4>
        </_100_0>
        <_65_35>
            <_4>1200</_4>
        </_65_35>
        <_0_100>
            <_4>3200</_4>
        </_0_100>
    </IOPS>
    <WAF>20</WAF>
    <PowerProfile>
        <Idle>2</Idle>
        <Read>900</Read>
        <Write>900</Write>
        <Mixed>900</Mixed>
    </PowerProfile>
    <Throughput>
        <_0_100>
            <_1024>50</_1024>
        </_0_100>
        <_100_0>
            <_1024>200</_1024>
        </_100_0>
    </Throughput>
</Tests>
```

Abbildung 5.2.: Sollzustände für eine Head Unit

Speicherbereich	*Speichertyp*	*Kapazität (Bytes)*
User- und Kartendaten	MLC oder TLC	23.696.084.511
Applikation	System Code (MLC oder TLC)	6.266.233.900
Boot	SLC	50.000.000
Gesamt		**30.112.318.411**

Abbildung 5.3.: Speicherbereiche der Head Unit mit Speichertyp und Kapazität

dass die Typen von allen getesteten Herstellern, wie eben beschrieben aufgeschlüsselt werden. Theoretisch könnten die Typen jedoch von jedem Hersteller unterschiedlich interpretiert werden. Auf Abbildung 5.3 ist die genaue Aufschlüsslung der Speicherbereiche mit ihren jeweiligen Konfigurationen und Kapazitäten zu sehen. Die Gesamtkapazität kann mit dem Prototypen abgeprüft werden (Feld: *UserCap*). Außerdem ergibt sich durch die verschiedene Konfiguration ein Mindestanzahl an LUs (Feld: *MaxLU*) von drei. Es müssen zusätzlich die drei Speichertypen *NormalType, SystemCode und Enhanced_Type1* unterstützt werden.

Eine essentielle Funktion muss noch unterstützt werden. Es handelt sich um das „Field Firmware Update“ (Feld: *FFU*), also ein Firmware-Update im bereits gelöteten Zustand des Speichers. Dies kann notwendig werden, sollte es zu Ausfällen mehrerer Speicher kommen und ein Update der Firmware könnte das Problem lösen.

Die Performance-Werte auf Abbildung 5.2 entsprechen Mindestanforderungen, die in der Entwicklungsphase der Beispiel-Head-Unit von den Applikations-Ingenieuren spezifiziert wurde (alle anderen wurden somit weggelassen). Für zukünftige Systeme können natürlich noch weitere zusätzliche Szenarien (z.B. Blockgrößen oder Tests) angelegt werden. Der WAF wurde relativ hoch angesetzt, da die Test-Methode dieser Arbeit ein Extrem-Szenario prüft und somit ein deutlich höherer Wert zu erwarten ist. Für die elektrische Leistung wurden im Beispiel nur zwei Zustände definiert: Leerlauf und Betrieb. Das bedeutet für den Test, dass das Ergebnis für den Idle-Stimulus für den Leerlaufzustand gilt und die restlichen Ergebnisse den Grenzwert für den Betriebszustand nicht überschreiten dürfen. Die Testergebnisse für dieses Fallbeispiel werden in Kapitel 7 gezeigt und diskutiert.

5.3. Implementierung von Messreihen

5.3.1. Entwicklungsumgebung

Hardwareumgebung

Abbildung 5.4 zeigt den gesamten Testaufbau inklusive Hardware zur Temperaturmessung. Der grundsätzliche Aufbau besteht aus fünf Komponenten:

- Test-Board
- Steuerungs-Board
- Laptop
- Thermometer (Überwachung der Temperatur am Speicher-Chip)

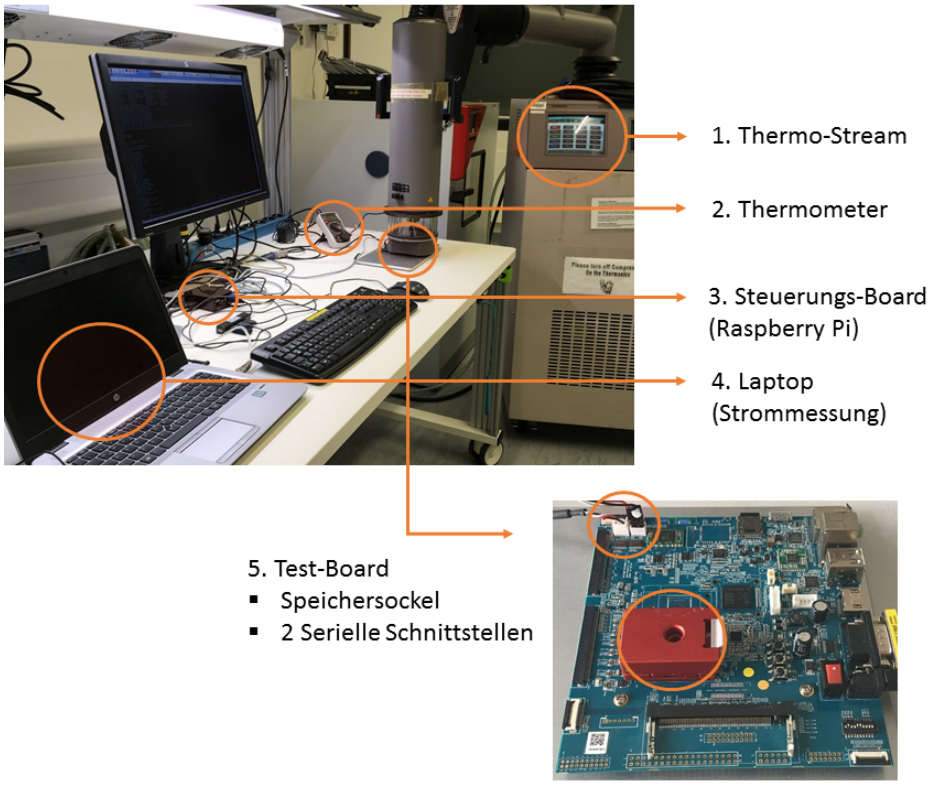


Abbildung 5.4.: Hardware Setup mit Temperaturmessung

Eigenschaften	*Beschreibung*
CPU	Exynos 7420 ARM Cortex A57 Quad, A53 Quad @1.8GHz
Arbeitsspeicher	LPDDR4 3GB 1600MHz POP Typ
Betriebssystem	Ubuntu 15.04
Speicherschnittstelle UFS	UFS 2.0
Speicherschnittstelle eMMC	eMMC 5.1
Speichersockel	Für UFS und eMMC
Zwei serielle Schnittstellen	Steuerung des Boards (Linux Konsole)

Abbildung 5.5.: Essentielle Daten zum Test-Board (HC7420-UFSeMM)

- Thermo-Stream (Erzeugung einer spezifizierten Temperatur)

Das Test-Board enthält die Software des Prototypen und ist in der Lage in einem speziellen Sockel eingebettete Speicher-Chips ohne einen Lötvorgang (UFS und eMMC) zu erkennen und zu mounten. Das Board erzeugt also den Teststimulus auf dem DUT und ist somit das wichtigste Gerät im Testaufbau. Es handelt sich dabei um das Performance-Test-Board *HC7420-UFSeMM* der Firma HowChip (Südkorea) [1]. Abbildung 5.5 zeigt die wichtigsten Eckdaten zu diesem Board. Da es sich um ein dediziertes Performance-Test-Board handelt, liefern die Hardwarekomponenten ausreichend Leistung (CPU, Arbeitsspeicher). Als Betriebssystem wird eine Linux-Distribution verwendet (Ubuntu 15.04), allerdings ohne HDMI-Unterstützung, das auf einer SD-Darte gespeichert ist. Das heißt es muss über zwei serielle Ports mit dem Board kommuniziert werden. Ein Port wird dabei zur Steuerung des Boards über Remote-Konsole verwendet, während der andere für eine Stromüberwachung genutzt werden kann. Dafür wurde vom Hersteller eine eigene Überwachungssoftware für Windows mitgeliefert.

Zur Steuerung wurde im Rahmen dieser Arbeit einer der bekanntesten Einplatinencomputer, der *Raspberry Pi 2 Model B* verwendet. Genaue Eckdaten sind bei diesem Steuerungs-Board zu vernachlässigen. Wichtig ist, dass eine Linux-Distribution (Raspbian) als Betriebssystem verwendet wird und somit eine Kommunikation über Remote-Konsole (serielle Schnittstelle) mit dem Test-Board möglich wird. Die Performance des Steuerungs-Boards spielt deswegen keine Rolle, weil lediglich das Testprogramm auf dem Testboard ausgeführt wird. Es fällt also keinerlei Rechnenaufwand auf dem Steuerungs-Board an.

Da die Stromüberwachungssoftware ein Windowsprogramm mit Graphical User Interface (GUI) ist, wurde die Strommessung für den Prototypen nicht vollständig automatisiert. Dazu ist der Laptop (Windows 7) notwendig, um parallel zum Teststimulus Stromdaten aufzuzeichnen. Das Überwachungsprogramm kann über einen Adapter (USB auf serielle Schnittstelle) mit dem Strom-Port des Test-Boards kommunizieren.

Für die Messreihen unter Temperatureinfluss wird ein Setup mit zwei Komponenten verwendet. Eine Komponente zur Temperaturerzeugung und eine zur Temperaturüberwachung. Ein sogenannter „Thermo-Stream“ erzeugt eine konstante Temperatur am Sockel und somit am DUT. Die Erhitzung oder Abkühlung erfolgt dabei über einen Luftstrom mit einer

[1] Informationen zum Board auf der offiziellen HowChip-Webseite: http://howchip.com/products/exsom7420/HC7420_UFSeMM.php; Stand 20.07.2018

UFS-Diagnosetool Hersteller 1
Sg3_utils; Version 1.39
eMMC-Diagnosetool Hersteller 1
eMMC-Diagnosetool Hersteller 2
Fio; Version 3.7

Abbildung 5.6.: Auf dem Test-Board verwendete Linux CLI-Tools

Eigenschaft	*Beschreibung*
Python	Version 2.7.9
Pandas	Python-Paket; Version 0.23
Matplotlib	Python-Paket; Version 1.4.2
PyPDF2	Python-Paket; Version 1.26

Abbildung 5.7.: Auf dem Test-Board verwendete Python-Version und -Pakete

konstanten Temperatur. Um eine Art Temperaturkammer zu erzeugen, wird durch den Einsatz von Schaumstoff der Zwischenraum zwischen Test-Board und Arm des Thermo-Streams abgedichtet. Es findet also kein oder ein sehr geringer Wärmeaustausch zwischen dem Inneren (Chip) und dem äußeren System (Labor) statt. Dadurch verändert sich die Temperatur des Sockels und speziell auch des DUTs, da ein Loch im Sockel den Luftstrom direkt auf dieses ermöglicht. Der Thermo-Stream überwacht zwar die Lufttemperatur direkt an dessen Ausgang, jedoch muss dies nicht die tatsächliche Temperatur am Speicher wiedergeben. Deswegen soll ein Thermometer die Temperatur direkt am Gehäuse des Speicherchips überwachen. Dazu wurde der Temperaturfühler durch ein Loch im Sockel direkt auf den Speicherchip geklebt. Erst, wenn dieses Thermometer die spezifizierte Temperatur anzeigt, wird die Testreihe gestartet.

Softwareumgebung

Dieses Unterkapitel spezifiziert die verwendete Software für die zwei Boards (Test und Steuerung).

Ubuntu 15.04 (Vivid) mit Kernel-Version 3.10.61 ist das Betriebssystem auf dem Test-Board. Auf Abbildung 5.6 sind alle auf dem Test-Board verwendeten CLI-Tools aufgelistet. Vier Tools werden für die Diagnose und Konfiguration von eMMC und UFS verwendet. Davon sind wiederum drei Tools direkt von Speicher-Herstellern zur Verfügung gestellt worden. *Sg3_utils* wird als Diagnose-/Konfigurationstool für UFS verwendet, da es SCSI-Befehle an Speicher versenden sowie empfangen kann. Das einzige verbleibende Tool ist *FIO*, das als Workloadgenerator für den Prototypen verwendet wird.

Abbildung 5.7 listet die Python-Version sowie alle auf dem Test-Board verwendeten Python-Pakete auf, die nicht bereits standardmäßig vorinstalliert sind. *Pandas* wird für die dynamische Datenverarbeitung des Prototypen verwendet. *Matplotlib* wird für das erzeugen von Tabellen und Plots genutzt. Die entstanden PDF-Dateien können zur Übersichtlichkeit schließlich mit dem Paket *PyPDF2* zu mehrseitigen PDF-Dateien zusammengefasst werden.

Auf dem Steuerungs-Board wird das Betriebssystem Raspbian 8 (Jessie) mit Linux Kernel 4.1.13 verwendet. Das einzige für diese Arbeit ausschlaggebende CLI-Tool, das auf diesem Board verwendet wird, ist *Minicom* in der Version 2.7. Dieses ermöglicht eine Kommunikation über Remote-Konsole zwischen Test- und Steuerungs-Board.

Die Auflistung der Tools in diesem Unterkapitel dient nur als Übersicht. Genauere Informationen, wie diese CLI-Tools und Python-Pakete für die Implementierung genutzt werden, folgen in Sektion 5.3.3.

5.3.2. Softwarearchitektur

In diesem Unterkapitel soll ein Verständnis für die implementierte Software-Architektur geschaffen werden. Der Prototyp wurde hauptsächlich in Python mit einem objektorientierten Ansatz entwickelt. Der zweite Programmieransatz für wenige Einzelfälle ist der Aufruf von Shell-Skripten der Linux-Shell. Als Ergebnis ergibt sich ein Python-Konsolen-Programm, das mit verschiedenen Argumenten ausgeführt werden kann. Die Abbildung 5.8 zeigt ein vereinfachtes UML-Klassendiagramm. Es ist in sofern vereinfacht, als dass nur die wichtigsten Funktionen der Klassen und keine Felder (bis auf DataFrame-Klasse) gezeigt werden. Durch das Eliminieren unwichtiger Eigenschaften wird eine übersichtlichere Darstellung ermöglicht. Diese vereinfachte Darstellung soll ausreichen, um ein Grundverständnis zur Architektur zu erlangen.

Als Hauptklasse dient die Klasse „Main“. Sie nimmt in Form von CLI-Argumenten Benutzereingaben entgegen. Diese Klasse besitzt wiederum eine Instanz der Klasse „TestManager“. Der TestManager besitzt Funktionen zur Steuerung der gesamten Funktionalität des Prototypen. Main nimmt die Benutzereingabe entgegen und ruft daraufhin die passende Funktion des TestManagers auf, wie beispielsweise die Konfiguration eines Speichers oder die Diagnose. Um all diese Funktionen ausführen zu können, besitzt der TestManager wiederum Instanzen aller übrigen Klassen, wovon die meisten keine Vererbung nutzen. Dabei handelt es sich um folgende Klassen:

- Configurator: Konfiguration des DUTs
- FileManager: Umgang mit Dateien (Lesen, Schreiben, Erstellen, usw.)
- FioUtility: Erzeugung von Workloads mit dem Tool *FIO*
- InputParser: Parsen der XML-Input-Datei und Bereitstellen der Informationen
- Plotter: Erstellung jeglicher Art von Plot (Tabelle, Graph, etc.)
- Reporter: Erstellung der Berichte
- PowerValidator: Erzeugung von Ergebnissen zur Strommessung

Eine dieser Klassen muss noch genauer erläutert werden, nämlich der *PowerValidator*. Diese Klasse ist notwendig, da die Strommessergebnisse extern (Laptop) entstehen und deshalb erst nach der Beendigung der Testreihe hinzugefügt werden können. Deshalb muss zur Erzeugung der Stromergebnisse (Plots, Teile der Reports) dieses Objekt nachträglich aufgerufen werden.

Drei Klassen nutzen Vererbung:

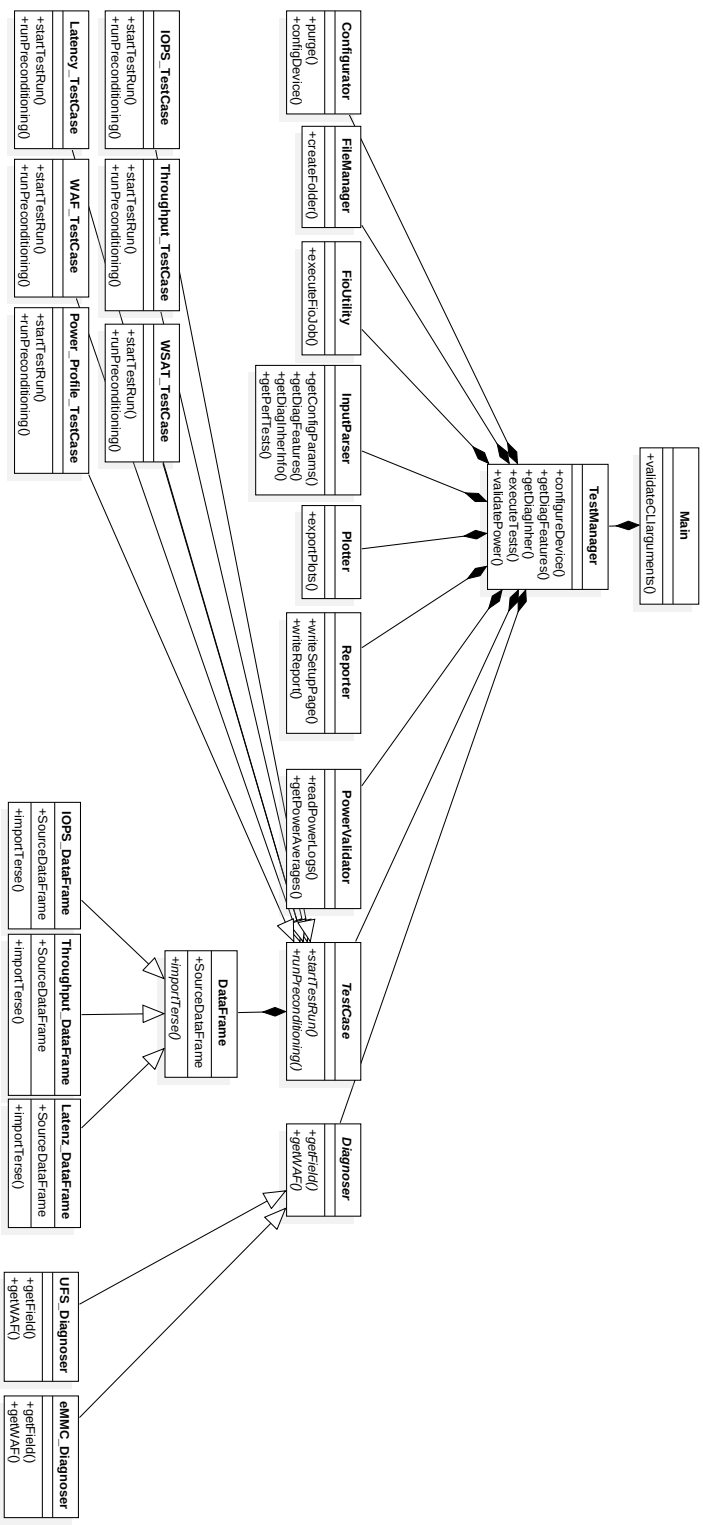


Abbildung 5.8.: Software-Architektur (vereinfachtes UML-Klassendiagramm)

```
usage: main.py [-h] [-t] [-f <filename>] [-v] [-c] [-x]

optional arguments:
  -h, --help            show this help message and exit
  -t, --test            execute Testcases
  -f <filename>, --file <filename>
                        the Input File with Config, Test and Diagnosis
                        Information
  -v, --validatePower   validate the Power Logs located in PowerTraces
  -c, --clean           Clean the Results Directory. All result Data is lost!
  -x, -export           export a zipped Archive of the last Test to the
                        Download Folder of the Raspberry
```

Abbildung 5.9.: Mögliche CLI-Argumente des Prototypen

- Diagnoser: Unterklassen für UFS und eMMC
- DataFrame: Unterklassen für die TestCases IOPS, Latenz und Durchsatz
- TestCase: Unterklassen für die verschiedenen TestCases (IOPS, Latenz, etc.)

Die Diagnoser-Klasse nutzt Vererbung, da die Informationen von Grund auf verschieden für beide Speichertypen ausgelesen werden, jedoch abstrakt gesehen beide die gleichen Informationskategorien liefern müssen. Es wird also die Informationsbeschaffung Schnittstellenebene abstrahiert. Die DataFrame-Klasse beinhaltet als Feld einen sogenannten „DataFrame“. Dieser fungiert als zentrale Datenstruktur. Der DataFrame lebt nur zur Laufzeit des Programms. Da die verschiedenen TestCases unterschiedliche Arten an Daten importieren, soll dieser Import abstrahiert werden. Außerdem können zusätzliche Funktionen in den Unterklassen anfallen, da die Ausführung der verschiedenen TestCases auch unterschiedlich ist und somit sogar unterschiedliche Datenformate produziert.

Die TestCase-Klasse nutzt Vererbung, da die verschiedenen TestCases verschiedene Parameter (Workload), sowie teilweise unterschiedliche Programmabläufe (vergleiche WSAT und Stromtest) besitzen. Deshalb wurde der Testablauf, sowie die Parameter abstrahiert. Der TestManager kann somit einfach einen TestCase „ausführen“.

5.3.3. Implementierung der Testkomponenten

Dieses Unterkapitel beschreibt die Implementierung der wichtigsten Komponenten des Prototypen. Zum einen soll die Funktion der Software erläutert und zum anderen in besonders interessanten Fällen die konkrete Implementierung gezeigt werden.

Steuerung und Ablauf des Prototypen

Abbildung 5.9 zeigt einen Screenshot der Anleitung des Testtools mit allen möglichen Optionen (Argumenten). Das Parsen der Argumente wird durch das Python-Modul „ArgumentParser“ realisiert. Das Argument „--help“ resultiert in dem gezeigten Screenshot. Die wichtigste Option ist „--test“ in Verbindung mit „--file“ gefolgt von dem Dateinamen der Input-Datei.

Ein beispielhafter Aufruf würde somit lauten: *python main.py --test --file <File>*. Die Option „--validatePower“ wieder in Verbindung mit der Option „--file“ gefolgt von dem Dateinamen der Input-Datei führt dazu, dass die beim Stromtest entstandenen Log-Dateien ausgewertet und in Ergebnisse umgewandelt werden. Dazu müssen die Log-Dateien vom

Entstehungsort (Laptop) auf das Test-Board kopiert werden. Bei der Ausführung werden dann Plots erzeugt und die Teile, die zum Stromtest gehören, an die Reports angehängt. Schließlich gibt es noch zwei Optionen für Hilfsfunktionen. Die Option „--clean“ führt zum Löschen des aktuellen Testordners, um eine Überfüllung der SD-Karte (Betriebssystem) des Test-Boards zu verhindern. Die Export-Option führt letztlich zu einer Komprimierung und Übertragung des gesamten Testordners an das Steuerungs-Board.

Für den Aufruf von anderen Programmen (nicht Python) wurde das Modul „Subprocess“ verwendet. Die Funktion *Call* ermöglicht es beispielsweise installierte Linux-CLI-Tools oder ein lokales Shell-Skript auszuführen.

Abbildung 5.10 zeigt ein Ablauf- oder Flussdiagramm des Prototypen. Hier soll noch einmal vereinfacht der Kontrollfluss des Programms dargestellt werden. Eine Durchführung einer Messreihe beginnt mit der Eingabe der Input-Datei. Daraufhin wird das DUT konfiguriert. Bei diesem Prozess werden die beiden Bericht-Dateien angelegt. Daraufhin folgt die zweiteilige Diagnose (Funktionsunterstützung und Inhärente Eigenschaften). Die Ergebnisse dieses Prozesses werden im Ergebnisbericht angehängt. Nun folgt eine Schleife, bei der alle Tests durchgeführt werden. Bei der Durchführung eines Tests werden Log-Dateien und Plots erzeugt. Die konkreten Ergebniswerte werden daraufhin in den Ergebnisbericht eingetragen. Nach dem Test wird geprüft, ob bereits alle Tests durchgeführt wurden. Ist dies nicht der Fall, so folgt der nächste Test. Sind alle Tests durchgeführt worden, wird die nächste Phase eingeleitet. Beim Stromtest entstehen die Log-Dateien auf dem Laptop, also extern. Diese müssen nun auf das Board übertragen werden und werden dann als Input für den „ValidatePower-Prozess“verwendet. Dieser interpretiert die Strom-Daten und erzeugt einen Power-Plot daraus. Schließlich werden noch die Ergebnisse dem entsprechenden Bericht hinzugefügt und die Messreihe wird beendet.

Input

Für das Interpretieren (Parsen) der Input-Datei wurde das Python-Modul *Xml* verwendet. Mit diesem kann ein *ElementTree* erstellt werden, anhand dessen die Informationen der Elemente ausgelesen werden können

Konfiguration

Generell wird im Rahmen dieser Arbeit Konfiguration in zwei Teile aufgeteilt. Zum einen wird *einmalig* das DUT vor der Testreihe konfiguriert. Dieser Vorgang wurde im Verlauf dieser Arbeit mit der Konfigurationsphase beschrieben. Das heißt bestimmte Speichereigenschaften werden auf einen gewollten Wert gesetzt, bzw. bestimmte Funktionen werden an- oder ausgeschaltet. Zum anderen stellt der Purge einen Konfigurationsakt dar. Dieser ist jedoch im Programmablauf Teil der Testdurchführung und wird somit *mehrmals* durchgeführt. Beides kann verschieden Implementiert werden und sollen nun jeweils für UFS und eMMC getrennt beschrieben werden.

Für UFS werden beide Prozesse folgendermaßen realisiert:

- Konfiguration (einmalig): Konfigurationsdeskriptor
- Purge: Durchführen des Befehls SCSI-Format

Der Konfigurationsdeskriptor wird ist zweigeteilt. Der *Device*-Bereich lässt es zu, Eigenschaften für den Gesamtspeicher festzulegen (z.B. Bootfunktion des Speichers an/aus). Auf

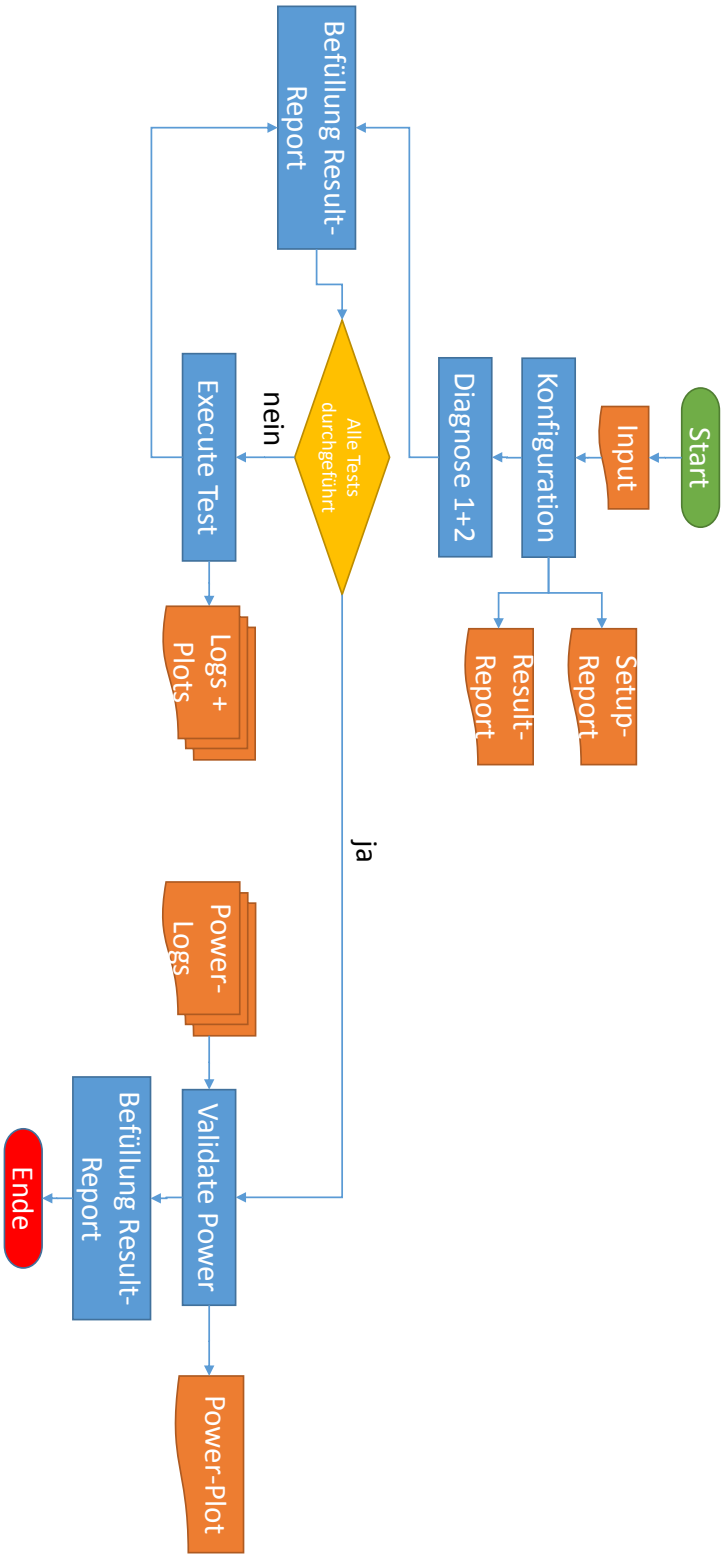


Abbildung 5.10.: Ablaufdiagramm des Prototypen

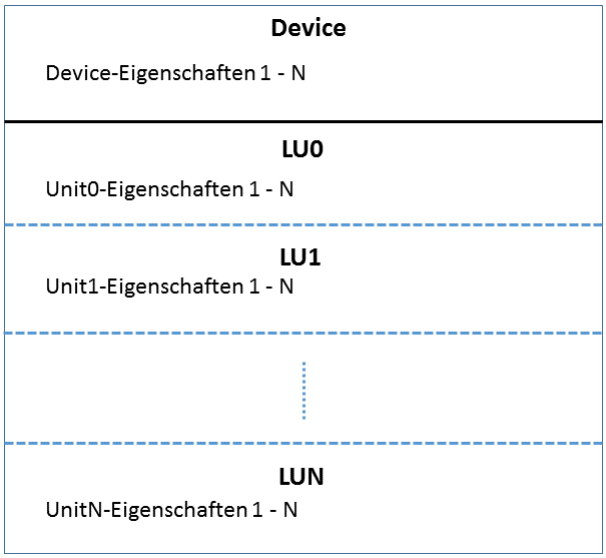


Abbildung 5.11.: Konfigurationsdeskriptor (UFS)

Bit-Ebene bedeutet das, dass die konfigurierbaren Werte des *Device Descriptors* gesetzt werden. Der *Unit*-Bereich konfiguriert Eigenschaften, die nur für eine LU gelten (z.B. Speicherkapazität der LU). Jede LU des Speichers, ob aktiv oder inaktiv, muss in diesem Deskriptor beschrieben werden. Besitzt der Speicher beispielsweise acht LUs, so müssen auch alle acht beschrieben mit all ihren Eigenschaften beschrieben werden. Soll die LU inaktiv sein, so werden die entsprechenden Felder auf null gesetzt. Abbildung 5.11 verdeutlicht den Aufbau des Konfigurationsdeskriptors noch einmal. Um diesen Deskriptor zu schreiben wurde ein Tool genutzt, das direkt von einem Speicherhersteller zur Verfügung gestellt wurde.

Für den Purge, der im UFS-Standard *Wipe* genannt wird, wird lediglich der SCSI-Format-Befehl ausgeführt. Dieser löscht alle Datenbits im Nutzerbereich des Speichers und setzt diese wieder auf den Initialwert (1 oder 0), sodass der Speicherbereich wieder dem unbenutzten Zustand entspricht. Der Befehl wurde durch das Tool *sg_format* aus der *sg3_utils*-Sammlung generiert.

Beim eMMC werden sowohl die einmalige Konfiguration als auch der Purge über das Schreiben von Eigenschaften in den Registern CSD und EXTCSD realisiert. Hierfür wurde wiederum ein Tool verwendet, das von einem Speicherhersteller zur Verfügung gestellt wurde. Der Purge wird im eMMC-Standard *Sanitize* genannt.

Diagnose

Mit einem Tool, das von einem Hersteller zur Verfügung gestellt worden ist, können alle Deskriptoren ausgelesen werden. Die ausgelesenen Hex-Werte werden dementsprechend vom Prototypen interpretiert. Das bedeutet beispielsweise, dass der Wert mit einem bestimmten Faktor (512 Byte) multipliziert werden muss, damit das Ergebnis genutzt werden kann. Für die Hilfsgrößen muss für UFS-Speicherhersteller 1 das besagte Tool verwendet werden. Das generiert dafür den SCSI-Read-Buffer-Befehl mit den richtigen Parametern (z.B. Offset). Somit gibt der Befehl die herstellerspezifischen Hilfsgrößen zurück.

Für den Speicherhersteller 2 muss das*sg_inquiry* Tool aus der *sg3_utils*-Sammlung. Prak-

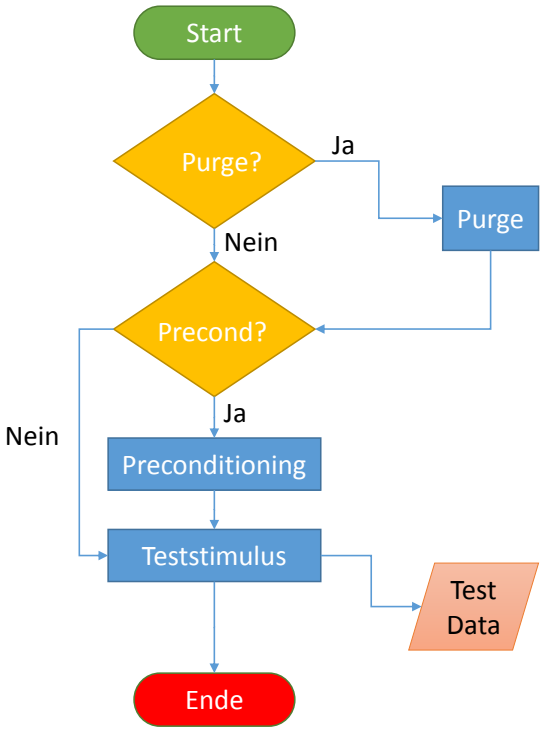


Abbildung 5.12.: Ablaufdiagramm eines Einzeltests

tisch sendet dies einen SCSI-Inquiry-Befehl an den Speicher. Werden dem Befehl die richtigen Parameter (z.B. Passwörter) mitgeben, so erhält man die herstellerspezifischen Hilfsgrößen.

Für Hilfsgrößen wird für Hersteller 1 das UFS-Hersteller-Tool verwendet. Hat extra Funktion für vendor-Felder. Dann genauso wie vorher mit pipe.

Die wichtigen Hilfsgrößen sind für UFS und eMMC gleich. Für die Berechnung des WAF müssen die tatsächlich auf dem NAND geschriebenen Schreibaktionen ausgelesen werden können damit sie mit den vom Host abgesendeten Schreiboperationen ins Verhältnis gesetzt werden können. Dies kann über den Durchschnittswert der Blocklöschaktionen angenähert werden. Wenn jeder Block auf dem Speicher einmal gelöscht wurde, so wurde er auch (annähernd)komplett geschrieben. Damit kann diese Datenmenge mit der Host-Datenmenge verglichen werden. Sowohl eMMC als auch UFS verschiedener Hersteller bieten diese Hilfsgröße in ihrem spezifischen Bereich an.

Beim eMMC müssen die Register CSD und EXTCSD ausgelesen werden. Dafür und für das Auslesen der Hilfsgrößen wurde für beide Hersteller 1 und 2 jeweils ihr eigenes Tool verwendet.

Einzeltest

Abbildung 5.12 zeigt ein generisches Ablaufdiagramm eines Einzeltests. Bezogen auf den Ablauf unterscheiden sich die Tests durch zwei Merkmale:

Testtyp	*Teststimuli*
IOPS	Testloop
Durchsatz	Testloop
Latenz	Testloop + Einzelstimuli (20 Minuten)
WAF	Testloop
Strom	Testloop
WSAT	Einzelstimuli (6 Stunden)

Abbildung 5.13.: Einzeltests mit ihren jeweiligen Teststimuli

- Purge notwendig: IOPS, Durchsatz, Latenz, WSAT
- Preconditioning notwendig: IOPS, Durchsatz, Latenz

Die zwei Tests WAF und Strom werden an den IOPS-Test gekoppelt, da dieser das Preconditioning sozusagen für die beiden Tests übernimmt. Deswegen soll vor diesen beiden Tests kein Purge und kein Preconditioning durchgeführt werden. Der WSAT stellt noch einen anderen Spezialfall dar, da hier das Verhalten der verschiedenen Performance-Zustände untersucht werden soll. Um auch den FOB und den Übergangszustand aufzuzeichnen, muss auf das Preconditioning verzichtet werden.

Der Workload für die Einzeltest wird von dem Tool *FIO* übernommen. Dieses Tool erzeugt den Teststimulus und zeichnet gleichzeitig in Log-Dateien die erzielten Ergebnisse für verschiedene Metriken (z.B. IOPS, Latenz) auf.

Grundsätzlich sind alle Testfälle durch zwei Teststimuli charakterisiert:

- Testloop mit verschiedenen Lese-Schreib-Verhältnissen und Blockgrößen
- Einzelstimulus über einen bestimmten Zeitraum

Eine Testloop kreiert dauerhafte IOs für verschiedene Lese-Schreib-Verhältnissen und Blockgrößen, wobei jeweils eine Paarung aus beiden Parametern eine Runde a 60 Sekunden ausgeführt wird. Beim Einzelstimulus wird eine einzige Parameterwahl getroffen und für eine bestimmte Zeit durchgeführt. Abbildung 5.13 zeigt ein Übersicht für alle Einzeltests mit ihren jeweiligen Teststimuli.

Datenverarbeitung

Für die Datenverarbeitung wurde das Python-Paket *Pandas* verwendet. Die Applikationsdaten werden zu keinem Zeitpunkt in einer Datenbank gespeichert, sondern bleiben nur zur Laufzeit in der dynamischen Datenstruktur „Data Frame“. Das bedeutet wiederum, dass alle Ergebnisse nach der durchgeführten Messreihe in Form der Berichte, Plots und Log-Files gespeichert werden. Die Rohdaten des Data Frames werden nach Applikationsende verworfen. Wie ein Data Frame konkret aufgebaut ist, kann auf Abbildung 5.14 erkannt werden. Das Beispiel zeigt einen IOPS-Data Frame. Jede Zeile entspricht dabei einem 60-sekündigen Teststimulus. Aufgezeichnet wird die Testrunde, das Lese-Schreib-Verhältnis, die Blockgröße und das Ergebnis der getesteten Metrik (hier IOPS). Dieser Dataframe enthält am Ende des Tests alle Ergebnisse um daraus den Output zu generieren.

	round_nr	rw_mix	bs	iops
0	1	100_0	4	1824.0
1	1	100_0	64	498.0
2	1	100_0	1024	85.0
3	1	65_35	4	148.0
4	1	65_35	64	84.0
5	1	65_35	1024	4.0
6	1	0_100	4	1470.0
7	1	0_100	64	132.0
8	1	0_100	1024	2.0
9	2	100_0	4	1745.0
10	2	100_0	64	461.0
11	2	100_0	1024	81.0
12	2	65_35	4	508.0
13	2	65_35	64	86.0
14	2	65_35	1024	4.0
15	2	0_100	4	524.0
16	2	0_100	64	36.0
17	2	0_100	1024	1.0
18	3	100_0	4	1625.0

Abbildung 5.14.: Beispiel eines IOPS-Data Frames

Plots und Berichte

Für das Erstellen der Plots wurde das Python-Paket *Matplotlib* verwendet. Ausgegeben werden PDF-Dateien für jeden einzelnen Plot. Sind alle Tests erfolgreich durchgeführt worden, so werden alle Plots, die zu einem bestimmten Test gehören noch einmal zu einem PDF zusammengefasst. Dafür wurde das Python-Paket *PyPDF2* verwendet. Die Berichtdateien sind einfache Textdateien, die über Python Standardfunktionen erstellt und nach und befüllt werden.

6. Messdurchführung

Dieses Kapitel soll nun die konkrete Durchführung einer Messreihe zeigen. Auch die entstandenen Ergebnisse werden gezeigt. Die beispielhafte Messreihe wird für einen UFS (Hersteller 1) mit einer Kapazität von 32 GB durchgeführt. Um einmal den kompletten Umfang an Ergebnissen zu sehen, soll die Messreihe den gesamten UFS-Diagnosekatalog abprüfen (siehe 4.4) und alle Tests mit allen Testgrößen durchführen. Es werden hier keine Erwartungswerte (falls Erwartungswert notwendig 0) spezifiziert, sondern nur Ergebnisse gesammelt. Dies entspricht einer Benchmark-Testreihe und kann für Speicher verschiedener Hersteller und Speicherkapazitäten durchgeführt werden. Die Interpretation der Ergebnisse mehrerer Testreihen folgt dann im Kapitel 7. Abbildung 6.1 zeigt die entsprechende Benchmark-Input-Datei.

Die die notwendigen Schritte um einen Test durchzuführen, wurden bereits in Kapitel 5 erläutert. Zur Wiederholung werden sie hier noch einmal aufgezählt:

- Input-Datei erstellen
- Testreihe mit Befehl starten
- Bei Stromtest parallel Strom-Logs mit HowChip-Software erstellen
- Nach Durchführung aller Tests die Strom-Logs auswerten

Werden diese Schritte durchgeführt ergibt sich ein Testordner, dessen Struktur auf Abbildung 6.2 zu sehen ist. Die erzeugten Plots und Berichte, die bei der durchgeführten Messreihe am besagten 32 GB UFS entstanden, befinden sich im Anhang 6.2.

```
<UFS>
    <Config>
        <DataReliability>True</DataReliability>
    </Config>
    <Diagnose>
        <FeatureSupport>
            <SecLU></SecLU>
            <FFU></FFU>
            <PSA></PSA>
            <DevLifeSpan></DevLifeSpan>
            <OOData></OOData>
            <MemoryTypes>
                <NormalType></NormalType>
                <SystemCode></SystemCode>
                <Non_Persistent></Non_Persistent>
                <Enhanced_Type1></Enhanced_Type1>
                <Enhanced_Type2></Enhanced_Type2>
                <RPMB></RPMB>
            </MemoryTypes>
        </FeatureSupport>
        <InherentCharacteristics>
            <UserCap>0</UserCap>
            <MaxLU>0</MaxLU>
            <NumSWPA>0</NumSWPA>
            <SysCodeMaxSize>0</SysCodeMaxSize>
            <NonPersistentMaxSize>0</NonPersistentMaxSize>
        </InherentCharacteristics>
    </Diagnose>
    <Performance>
        <IOPS>
            <_100_0>
                <_4>0</_4>
                <_64>0</_64>
                <_1024>0</_1024>
            </_100_0>
            <_65_35>
                <_4>0</_4>
                <_64>0</_64>
                <_1024>0</_1024>
            </_65_35>
            <_0_100>
                <_4>0</_4>
                <_64>0</_64>
                <_1024>0</_1024>
            </_0_100>
        </IOPS>
        <WAF>0</WAF>
        <PowerProfile>
            <Idle>0</Idle>
            <Read>0</Read>
            <Write>0</Write>
            <Mixed>0</Mixed>
        </PowerProfile>
        <Throughput>
            <_0_100>
                <_1024>0</_1024>
            </_0_100>
            <_100_0>
                <_1024>0</_1024>
            </_100_0>
        </Throughput>
        <Latency>
            <_100_0>
                <_4>0</_4>
                <_8>0</_8>
                <_16>0</_16>
            </_100_0>
            <_65_35>
                <_4>0</_4>
                <_8>0</_8>
                <_16>0</_16>
            </_65_35>
```

```
73              <_0_100>
74                  <_4>0</_4>
75                  <_8>0</_8>
76                  <_16>0</_16>
77              </_0_100>
78          </Latency>
79          <WSAT></WSAT>
80      </Performance>
81  </UFS>
82
```

Abbildung 6.1.: Input für einen UFS-Benchmark

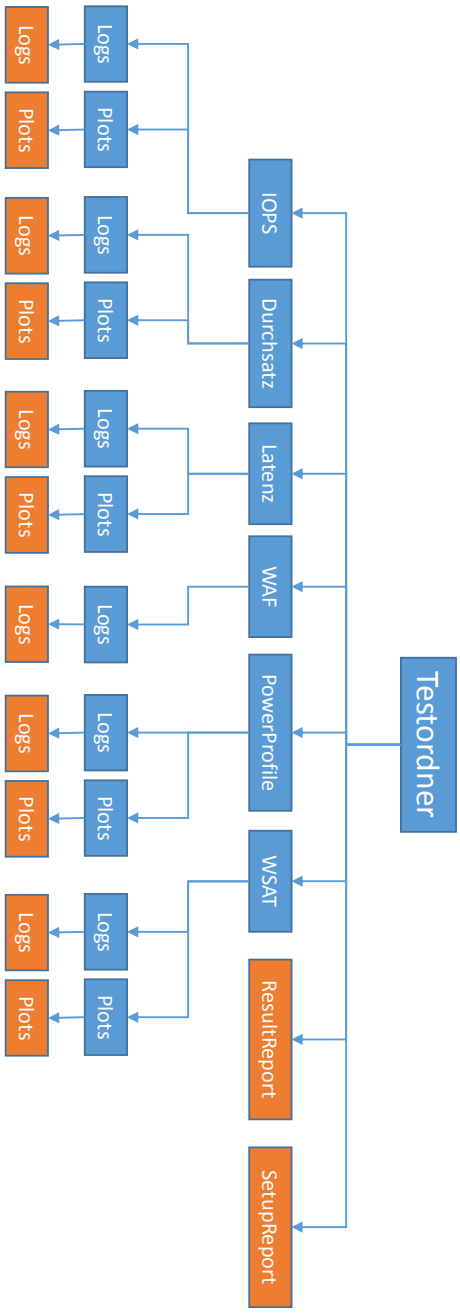


Abbildung 6.2.: Verzeichnisstruktur des Ergebnisordners (blau: Verzeichnis, orange: Datei)

7. Ergebnisinterpretation und Ausblick

In diesem Kapitel sollen die Ergebnisse der durchgeführten Testreihen diskutiert werden. Erst werden die Ergebnisse des BMW-Fallbeispiels betrachtet. Anschließend werden die Ergebnisse der Benchmark-Testreihen für verschiedene Speicher besprochen. Dazu sollen alle Testtypen separat im Fokus stehen. Schließlich soll noch eine Zusammenfassung und Bewertung der Erkenntnisse, sowie ein Ausblick auf zukünftige Weiterentwicklungen der Testmethode folgen.

7.1. BMW Fallbeispiel

In der Sektion 5.2 wurde eine BMW-Head-Unit als Fallbeispiel verwendet, um die Anforderungen einer realen Applikation zu prüfen. Der resultierende Input wurde verwendet und eine Testreihe an einem 32 GB UFS des Herstellers 1 getestet.

Abbildung 7.1 zeigt den Ergebnisbericht dieser Testreihe. Alle überprüften Funktionen werden vom DUT bereitgestellt. Auch die gemessenen inhärenten Eigenschaften übertreffen die geforderten Werte. Für alle Performance-Anforderungen ergibt sich das gleiche Bild. Sowohl die IOPS-, als auch Durchsatz-Erwartungswerte werden erreicht. Lediglich der WAF wird gravierend überschritten (rund 80%). Die Anforderungen an die elektrische Leistung werden deutlich übertroffen.

Nun soll versucht werden zu interpretieren, warum diese Ergebnisse zu Stande kamen. Die spezifizierten Anforderungen, stammten, sofern Vorhanden aus dem Anforderungskatalog einer Head-Unit, die bereits in Produktion ist. Als persistenter Speicher wird hier ein 32 GB eMMC verwendet. Die Funktionalität eines UFS-Speichers ist vielseitiger im Vergleich zum eMMC. Deshalb wurde die Funktionsunterstützung ohne Ausnahme bestanden. Zur Performance ist zu sagen, dass die für einen eMMC spezifizierten Werte von einem aktuelleren Speicherstandard (UFS) erfüllt werden sollten. Sonst wäre der neuere Standard ein Rückschritt in Sachen Performance. Der WAF wurde deshalb so gravierend überschritten, da der Test stark als Extremfall konzipiert wurde. Eine dauerhafte Beanspruchung mit Schreibaktionen verschiedener Blockgrößen stellt eine sehr hohe Beanspruchung für NAND-basierte Speicher dar, insbesondere für die Firmware. Diese muss für gewisse Fälle optimiert werden. UFS-Speicher scheinen nicht für diesen Anwendungsfall ausgelegt zu sein (siehe Sektion 7.2). Der Test muss also eine anwendungsnährere Workload erzeugen, um einen WAF zu überprüfen, der einem realistischen Szenario entspricht. So wie der Test aufgebaut war, bildet er einen Extremfall ab und hat somit eher einen experimentellen Charakter, als eine tatsächliche Anforderungsüberprüfung. Die Anforderungen an die elektrische Leistung wurden deshalb stark unterschritten, da der UFS-Standard einen starken Fokus auf die niedrige elektrische Leistung legt. Die treibende Kraft in der Entwicklung von NAND-Speichern ist die Mobiltelefon-Industrie. Hier ist der niedrige Stromverbrauch eine essentielle Anforderung, da ein zu hoher Verbrauch den Akku zu schnell entleert und somit das Produkt stark abwertet. Folglich unterschreitet der neuere Standard UFS gerade im Leerlauf-Zustand die für eMMC spezifizierten Anforderungen.

```
1
2   RESULT REPORT
3
4
5   ----------Diagnosis Section----------
6   Feature Support
7
8
9   FFU: Passed
10  NormalType: Passed
11  SystemCode: Passed
12  Enhanced_Type1: Passed
13
14
15  Inherent Information
16
17  UserCap: Expected Value: 30112318411    Diagnosed Value: 32006733824  ---> Passed
18  MaxLU: Expected Value: 3    Diagnosed Value: 8  ---> Passed
19  SysCodeMaxSize: Expected Value: 5866633900  Diagnosed Value: 32006733824  ---> Passed
20
21
22  ----------Performance Section----------
23
24  IOPS:
25
26  RW-Mix: 100_0 BlockSize: 10000  Expected Value: 0 Measured Value: 11803.0 ---> Passed
27  RW-Mix: 0_100 BlockSize: 1000  Expected Value: 0 Measured Value: 1583.0 ---> Passed
28  RW-Mix: 65_35 BlockSize: 3500  Expected Value: 0 Measured Value: 4455.33333333 --->
    Passed
29
30  Write Amplification Factor:
31
32  Expected WAF: 10
33  Measured WAF: 18.114987715   ---> Failed
34
35
36  Throughput (MB/s):
37
38  RW-Mix: 0_100 BlockSize: 1024  Expected Value: 50 Measured Value: 84.4382208 --->
    Passed
39  RW-Mix: 100_0 BlockSize: 1024  Expected Value: 200 Measured Value: 551.0150144 --->
    Passed
40
41
42  ----------Power Section----------
43
44  Average Consumed Power (mW):
45
46  Read: Expected Value: 900   Measured Value: 248.80542989850954   ---> Passed
47  Write: Expected Value: 900  Measured Value: 312.9047465141357    ---> Passed
48  Idle: Expected Value: 2 Measured Value: 0.43285668383462195  ---> Passed
49  Mixed: Expected Value: 900  Measured Value: 348.1271586959069    ---> Passed
```

Abbildung 7.1.: Ergebnisbericht der Testreihe zum BMW-Fallbeispiel (für UFS)

Als Resultat wäre der getestete Speicher für diesen Anwendungsfall des automotiven Umfelds funktional geeignet, mit Ausnahme des WAF. Dieser ist jedoch, wie bereits erklärt, nicht repräsentativ. Wäre eine andere Anforderung gescheitert, so wäre Eignung nicht festgestellt. Ein jedes Kriterium eines Anforderungskatalogs ist essentiell und das Nicht-Erfüllen würde ergeben, dass der Speicher nicht geeignet ist. Eine Gewichtung der einzelnen Anforderungen ist somit nicht sinnvoll.

7.2. Benchmark

Vier verschiedene Speicher wurden für den Benchmark getestet:

- UFS 2.1 von Hersteller 1, 32 GB, 2D-Technologie, 2 NAND-Dies, MLC-Konfiguration
- UFS 2.1,von Hersteller 2, 32 GB, 3D-Technologie, 1 NAND-Die, TLC-Konfiguration
- eMMC 5.0, von Hersteller 1, 8 GB, 2D-Technologie, 1 NAND-Die, MLC-Konfiguration
- eMMC 5.0, von Hersteller 2, 64 GB, 2D-Technologie, 8 NAND-Dies, MLC-Konfiguration

Aufgrund der verfügbaren Muster, konnten zwei vergleichbare UFS-Speicher getestet werden. Die eMMC-Speicher waren jedoch von der Kapazität zu unterschiedlich, als dass die Performance-Ergebnisse vergleichbar waren. Daraus folgt, dass für einige Tests nur die UFS-Speicher verglichen werden, jedoch wo es sinnvoll ist ein Vergleich der eMMCs hinzukommt.

Es soll nun erst für die Diagnose und anschließend für jeden Einzeltest erläutert werden, welche Auffälligkeiten beobachtet wurden. Sofern möglich sollen dann Gründe für diese Auffälligkeiten gefunden wurden.

7.2.1. Diagnose

Die beiden UFS-Speicher bieten unterschiedliche Funktionen an. Hersteller 1 bietet sowohl die non-persistente Speicherkonfiguration, als auch die Konfiguration für System Code. Hersteller 2 bietet beide Konfigurationen nicht an. Bei den inhärenten Eigenschaften fällt auf, dass Hersteller 1 nur 8 LUs zur Verfügung stellt, während Hersteller 2 sogar 32 LUs bietet.

Bei den beiden eMMCs war die Funktionsunterstützung identisch. Lediglich die inhärenten Informationen unterscheiden sich, wie beispielsweise die Cache-Größe. Da die Speicher jedoch unterschiedliche Kapazitäten besitzen, können diese Ergebnisse nicht gewertet werden.

Als Ergebnis lässt sich hier zusammenfassen, dass sich die Speicher durchaus im Diagnosebereich unterscheiden können und somit dieser Testteil auch sinnvoll ist. Es wären jedoch noch mehr verschiedene Hersteller notwendig, um sich ein wirklich aussagekräftiges Bild über die Unterschiede machen zu können.

7.2.2. IOPS

Abbildungen 7.2 und 7.3 zeigen die beiden IOPS-Annährungsplots. Die tatsächlichen Werte sollen bei der Betrachtung dieser beiden Plots nicht Gegenstand der Diskussion sein. Es geht hier vielmehr um die Stabilität des Speichers. Der Speicher des Herstellers 1 ist sehr stabil (SS-Kriterien) in seiner Performance (Lesen, Schreiben, Gemischt). Aus diesem Grund wird der SS auch schon in Runde sechs erreicht. Beim Speicher des Herstellers 2 fällt jedoch

auf, dass die Messwerte sehr stark, insbesondere für Schreiben und gemischten Workload. Aus diesem Grund wird der SS nicht erreicht und nach Runde 25 abgebrochen. Außerdem kommt es bei beiden Speicher beim Schreiben zu einer positiven Steigung der Werte und bei gemischten Workload zu einer fallenden Steigung in den ersten Runden. Dass sich die Firmware auf bestimmte Workloads anpasst, wurde in Kapitel 4 beschrieben und ist Grund für die testabhängige Präkonditionierung. Dieses Verhalten war also zu erwarten.

Der entscheidende Faktor für die unterschiedliche Stabilität könnte die kurze Pause zwischen den 60-sekündigen Teststimuli (2 Sekunden) sein. Damit man versteht, was sich in dieser Zeit abspielt, muss der sogenannte *Pseudo-SLC-Cache* verstanden werden. Im Gespräch mit den verschiedenen Speicherherstellern wurde diese Arbeitsweise der Firmware erläutert. Um die sehr hohe Datenrate des UFS für den Benutzer zu garantieren, werden bei einem Schreibvorgang die Daten als SLC-Daten in eine MLC- oder TLC-Zelle geschrieben. Das heißt es werden nur noch zwei Spannungslevel und somit nur noch ein Bit Information unterschieden. Damit wird der Schreibvorgang gerade im Vergleich zum TLC-Schreibvorgang deutlich beschleunigt. Daraus ergibt sich jedoch, dass der Speicher beginnt die Daten als TLC oder MLC „umzuschreiben“, sobald er nicht mehr aktiv beschrieben wird. Es werden also SLC-Blöcke gelöscht und die Daten in einen neuen Block mit höherer Bitdichte geschrieben.

Die große Schwankung könnte ein Resultat der nicht vollständigen Leerung des *Pseudo-SLC-Caches* sein. Das heißt in den zwei Sekunden Pause zwischen den Teststimuli wird nur ein Teil umgeschrieben und ein Teil bleibt im Cache. Nun steht in der nächsten Testrunde weniger Cache zur Verfügung und es kommt zu einer verminderten Leistung. Diese Vermutung stützt, dass zwei weitere Faktoren dafür sprechen:

- Nur 1 NAND-Die im Vergleich zu Hersteller 2
- TLC- im Vergleich zu MLC-Konfiguration

Ein physikalischer NAND-Die mit TLC-Konfiguration bedeuten nur die Hälfte an physikalischen NAND-Zellen im Vergleich zu Hersteller 2 mit zwei NAND-Dies und MLC-Konfiguration. Daraus folgt auch eine stark reduzierte Pufferkapazität des SLC-Caches. Konkret heißt das bei 32 GB Nutzerbereich, dass der Speicher von Hersteller 1 ungefähr ein maximal ein Drittel (ca. 10.5 GB) und der Speicher von Hersteller 2 maximal die Hälfte (ca. 16 GB) an SLC-Cache zur Verfügung haben. Diese Verminderte Puffer-Kapazität bedeutet, dass früher umgeschrieben werden muss und somit der Leistungseinbruch früher stattfindet. Verstärkt wird dieser Effekt zusätzlich durch den Vorgang des Umschreibens. Da das Schreiben von TLC-Daten länger braucht als das Schreiben von MLC-Daten (siehe Kapitel 2) wird die Entleerungsaktion des SLC-Caches zwischen den Teststimuli noch zeitintensiver für den Speicher von Hersteller 1. Hinzukommt, dass in dieser Zwischenzeit auch der Garbage Collector Zeit für das Löschen von nicht gemappten Blöcken nutzen kann. Dieses Phänomen kann sich folglich auch auf andere Tests auswirken, die die selbe Art der Belastung beinhalten (Durchsatz, Latenz).

Bei den konkreten IOPS-Werten im Vergleich hat sich ergeben, dass der Speicher von Hersteller 1 deutlich besser im Lesen abschneidet (siehe Abbildung 7.4). Bei Blockgrößen von 64 und 1024 K, also eher großen Blockgrößen (hoher Durchsatz), ist der gemessene Wert sogar doppelt so hoch, wie beim Speicher von Hersteller 2. Im gemischten Bereich (65% Lesen, 35% Schreiben) ist bei UFS 1 ein ähnlicher Trend zu erkennen. Auch hier wird UFS 1 mit zunehmender Blockgröße besser performant (bis zu 40% bei 1024K Blockgröße). Beim

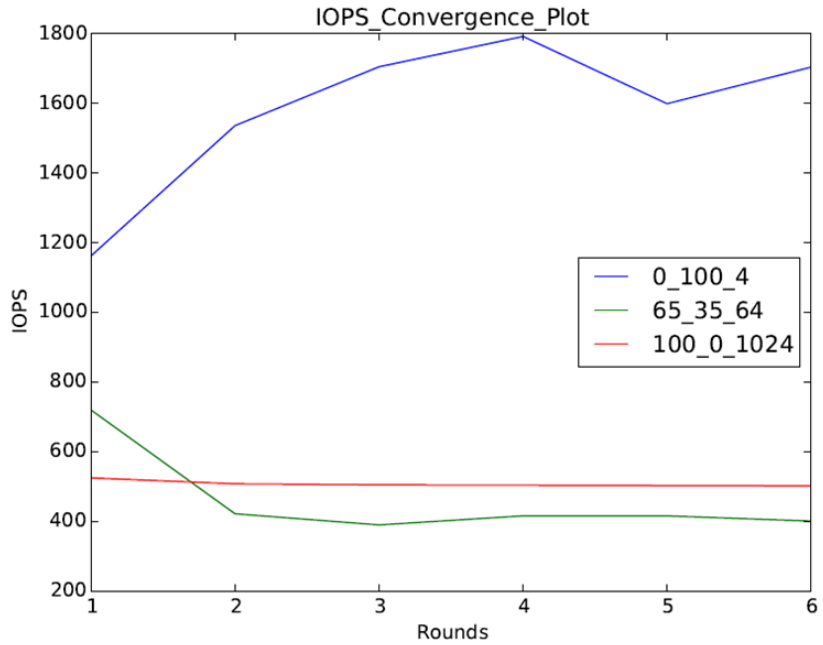


Abbildung 7.2.: IOPS Annährungsplot für UFS (Hersteller 1)

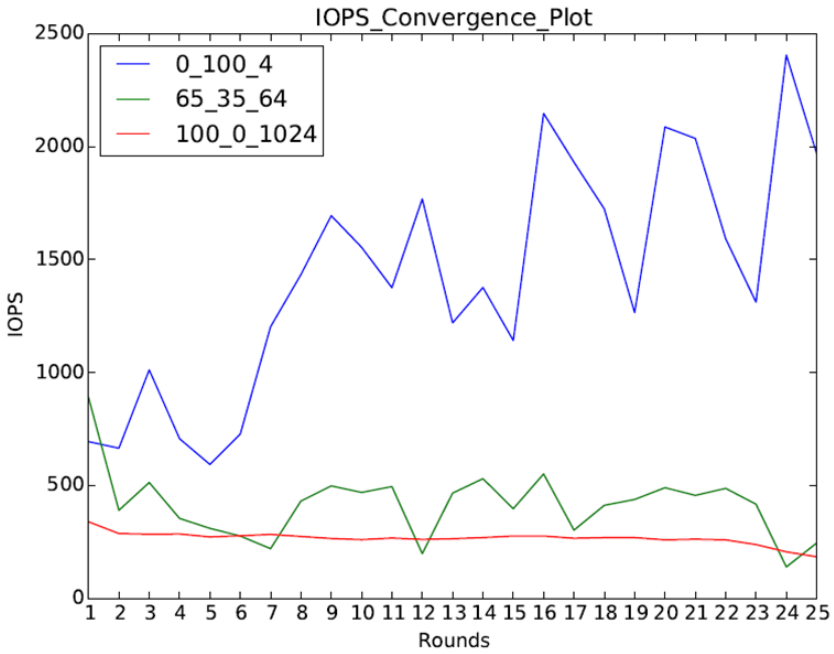


Abbildung 7.3.: IOPS Annährungsplot für UFS (Hersteller 2)

IOPS(in) - All RW Mix & BS - Tabular Data – Hersteller 1

	100_0	65_35	0_100
4K	11803.0	4455.333	1583.0
64K	5612.0	460.833	119.833
1024K	507.833	36.0	13.167

IOPS(in) - All RW Mix & BS - Tabular Data – Hersteller 2

	100_0	65_35	0_100
4K	9046.24	3963.32	1425.88
64K	2273.12	414.92	140.4
1024K	266.04	21.72	20.04

Abbildung 7.4.: Testergebnisse für den IOPS-Test für UFS von Hersteller 1 und 2

Throughput (in MB/sec) - All RW Mix & BS - Hersteller 1

	100_0	0_100
1024K	551.015	84.438

Throughput (in MB/sec) - All RW Mix & BS - Hersteller 2

	100_0	0_100
1024K	385.957	41.515

Abbildung 7.5.: Testergebnisse für den Durchsatz-Test für UFS von Hersteller 1 und 2

Schreiben ist UFS 2 sogar mit zunehmender Blockgröße höher performant (bis zu 35% bei 1024 K Blockgröße).

Es zeigt sich also deutlich eine unterschiedliche Optimierung für beide Speicher. UFS 1 besonders beim Lesen performant, während UFS 2 höher beim Schreiben abschneidet.

7.2.3. Durchsatz

Die beschriebene Instabilität in der Performance konnte beim Durchsatz-Test nicht beobachtet werden. Es gibt nur einen Unterschied im Teststimulus, nämlich das Zugriffsmuster der I/O-Anfragen. Beim Durchsatz wird, im Gegensatz zu allen anderen Tests, ein sequentieller statt einem wahlfreien Zugriff verwendet. Dies ist sinnvoll, da der Durchsatz besonders wichtig beim Transport großer Datenmengen weniger Quellen ist. Es macht Sinn diese Daten in einem zusammenhängenden Speicherbereich zu schreiben, da die Wahrscheinlichkeit hoch ist, dass diese wieder zusammen ausgelesen werden. Der sequentielle Zugriff ermöglicht es, Daten in separaten Blöcken zu speichern, anstatt die Daten über mehrere Blöcke zu verteilen. Somit ist beim Löschen eines Datenblocks auch nur ein Logischer Block betroffen. Es kann also sehr effizient gelöscht und geschrieben werden und die 2 Sekunden zwischen den Teststimuli reichen scheinbar aus, um vollständig die „Aufräumarbeiten“ auszuführen.

Zu den gemessenen Werten ist zu sagen, dass der Speicher von Hersteller 1 um ca. 30% höheren Lesedurchsatz und den doppelten Schreibdurchsatz schafft(siehe Abbildung 7.5). Verantwortlich dafür könnte die Die-Anzahl der Speicher sein. Hersteller 1 besitzt zwei physikalische Dies, während Hersteller 2 einen physikalischen Die besitzt. Nun muss Abbildung 2.28 aus dem Grundlagenkapitel aufgegriffen werden. Jeder zusätzliche NAND-Die bedeutet eine zusätzliche *parallele* NAND-Schnittstelle (sofern die Dies nicht gestapelt werden).

Latency(avg_lat in msec) - All RW Mix & BS - Hersteller 1

	100_0	65_35	0_100
4K	0.401	0.428	0.936
8K	0.393	0.404	1.331
16K	0.441	0.51	2.469

Latency(avg_lat in msec) - All RW Mix & BS - Hersteller 2

	100_0	65_35	0_100
4K	0.493	0.34	0.985
8K	0.529	0.448	2.364
16K	0.572	0.764	4.495

Abbildung 7.6.: Testergebnisse für den Latenz-Test für UFS von Hersteller 1 und 2

Das hat zur Folge dass die Daten bei mehreren Dies parallel weggeschrieben werden können. Dies würde den doppelten Schreibdurchsatz erklären. Im Prinzip müsste auch der Lesedurchsatz doppelt so hoch sein. Dieser kann jedoch nicht größer als der maximale Durchsatz der Außenschnittstelle sein. Der Test wurde bei HS-Gear 2 mit zwei Lanes durchgeführt. Die maximale Geschwindigkeit der UFS-Schnittstelle mit dieser Konfiguration (nur Bits ohne Protokoll) beträgt ca. 580 MB/S (Berechnung siehe Sektion 2.6.1). Der gemessene Wert bei Hersteller 1 kommt also relativ nahe an diesen Wert. Wird der Mehraufwand durch das Übertragungsprotokoll mitbedacht, so scheint es, als ob die maximale Lesegeschwindigkeit erreicht wird. Das Bottleneck ist hier also die Außenschnittstelle, wohingegen beim Schreiben die NAND-Schnittstelle den Engpass darstellt.

7.2.4. Latenz

Die Stabilität der Performance der beiden UFS-Speicher zeigt das gleiche Verhalten, das beim IOPS-Test beobachtet wurde. UFS 1 bleibt stabil während UFS 2 starken Schwankungen unterliegt.

Anhand der Messergebnisse (siehe Abbildung 7.6) der durchschnittlichen Latenz lässt sich das Speicherverhalten noch genauer beschreiben. Die Latenzen beim Lesevorgang sind für beide Speicher im gleichen Zeitbereich und von 8 bis 16K Blockgrößen steigern sich die Werte nur knapp. Interessant ist die Latenzentwicklung über die Blockgrößen beim Schreibvorgang. Bei 8K Blockgröße erzielen beide Speicher ungefähr den selben Wert (ca. 0.9ms). Wird nun aber die Blockgröße verdoppelt, so steigert sich die Latenzzeit bei UFS 1 (2 physikalische Dies) um ca. 30%, während sich die Latenzzeit von UFS 2 (1 physikalischer Die) mehr als verdoppelt. Bei einer Blockgröße von 16K findet wieder bei beiden Speichern eine nahezu Verdopplung der Latenzzeit statt. Interessant ist hier auch zu erwähnen, dass 16K die physikalische Page-Größe der getesteten UFS-Speicher ist. Ein Erklärungsansatz wäre, dass der UFS 1 bei einer 8K Anfrage das Datenpaket in zwei 4K-Teile aufteilt und zwei Pages auf zwei verschiedenen NAND-Dies nur jeweils ein Viertel vollschreibt. Hier würde es sich also um einen sogenannten „Partial Write“ handeln (siehe Kapitel 2). Bei der nächsten 8K-Anfrage würden dann jeweils die zweiten Viertel der zwei Pages beschrieben. Somit verlängert sich die Zeit nur durch den Auteilungs- und Mappingsmehraufwand, die Schreiblatenz bleibt jedoch gleich. Es findet also ein paralleler Schreibvorgang auf zwei Dies statt. UFS 2 kann diese Aufsplittung nicht vornehmen, da er nur einen Die zur Verfügung hat. Somit verdoppeln sich die Latenzzeiten bei zweifacher Blockgröße. Bei den gemischten Anfragen überwiegen die

Leseanfragen (65 zu 35%), deshalb kann dieser Trend nur abgeschwächt beobachtet werden.

Der 59s-Test ergab, dass die „garantierte“ Latenzzeit für Schreibanfragen mit 4K Blöcken bei UFS 1 um ca. 20ms oder 25% geringer ist. Abbildungen 7.7 und 7.8 zeigen jeweils das Latenzzeit-Histogramm für die beiden Speicher. Bei UFS 1 ist ein relativ konstanter Abfall der I/O-Anzahl bei steigender Latenzzeit zu beobachten. Das bedeutet, je länger die Latenzzeit ist, desto seltener treten diese auf. Bei UFS 2 bilden sich im Bereich um 40 und um 80 ms Anhäufungen. Dies ist auch der Grund, dass sich somit der zu 99,999% garantierte Wert zu den längeren Latenzzeiten verschiebt. Genaue Gründe für diese Häufungen sind ohne Kenntnis der Firmware schwer zu finden.

7.2.5. WSAT

Der WSAT-Test zeigt das Firmware-Verhalten eines Speichers über einen längeren Zeitraum (sechs Stunden) unter Dauerbelastung (100% Schreiben). Abbildung 7.9 zeigt den WSAT-Plot für alle vier getesteten Speicher. Es können zwei wichtige Erkenntnisse aus dieser Abbildung abgeleitet werden. Zum einen kann eine Art „Sägezahnmuster“ bei eMMC 1 erkannt werden. Es könnte sich um einen Garbage Collection-Algorithmus der Firmware handeln, der nach einem bestimmten Muster Löschaktionen durchführt. Ein tatsächlicher Erklärungsversuch dieses Verhaltens kann auf Grund des Informationsmangels im Zuge dieser Arbeit nicht durchgeführt werden. Zum anderen scheint es ein Firmware-Verhalten zu geben, das sowohl UFS 1 und 2 als auch eMMC 1 teilen. Die Leistung ist in vier Plateaus gestaffelt.EMMC 2 besitzt nur zwei Leistungsplateaus. Der Grund dafür ist der schon erwähnte *Pseudo-SLC-Cache*. Nur eMMC 2 benutzt diese Funktion nicht. Die Daten werden direkt im MLC-Modus geschrieben. Ist der Speicher voll, so muss für das Schreiben eines neuen Blocks auch vorher ein Block gelöscht werden. Deshalb ergibt sich auch der dramatische Leistungseinbruch.

Was allerdings bei den restlichen drei Speichern passiert wird durch Abbildung 7.10 verdeutlicht. Hier werden die WSAT-Plateaus mit ihren jeweiligen geschriebenen Datenmengen für den UFS 1 gezeigt. Die Gründe für die Plateau-Bildung wurden von Hersteller 1 erläutert. Die ersten 14,88 GiB werden im Pseudo-SLC-Modus geschrieben und dienen im Prinzip als Puffer. Somit ist der gesamte Speicher vollgeschrieben. Würde jetzt der Teststimulus eingestellt, so würde der Speicher alle SLC-Blöcke in MLC-Blöcke umschreiben und der halbe Speicher wäre wieder leer. Wird der Stimulus jedoch weitergeführt, so muss nun der Speicher SLC-Cache-Blöcke in MLC umschreiben und die eintreffenden neuen Daten als MLC-Daten wegschreiben. Das 4,69 GiB Plateau könnte auf das Beschreiben von den sogenannten „Spare Blocks“ also Reserveblöcke hindeuten. Dies ist allerdings nur eine These. In dem gesamten 14,79 GiB großen Abschnitt wird nach jeder Schreibaktion *Garbage Collection* (siehe Kapitel 2) durchgeführt, wobei die Aktivität kontinuierlich erhöht wird. Es wird also kontinuierlich mehr gelöscht, wenn der Garbage Collector aktiv wird. Sobald jedoch der gesamte SLC-Cache umgeschrieben wurde (14,88 GiB) und der entstandene MLC-Bereich voll ist (14,88 GiB umgeschriebene Cache-Daten + 14,79 GiB neue Anfragen = 29,67 GiB Nutzerbereich), fällt die Performance dramatisch auf das Plateau vier. Hier muss vor jedem Schreibbefehl eine Löschaktion durchgeführt werden, sofern kein Block frei ist. Es dominiert also die (im Vergleich zur Schreibzeit) sehr lange Löschzeit.

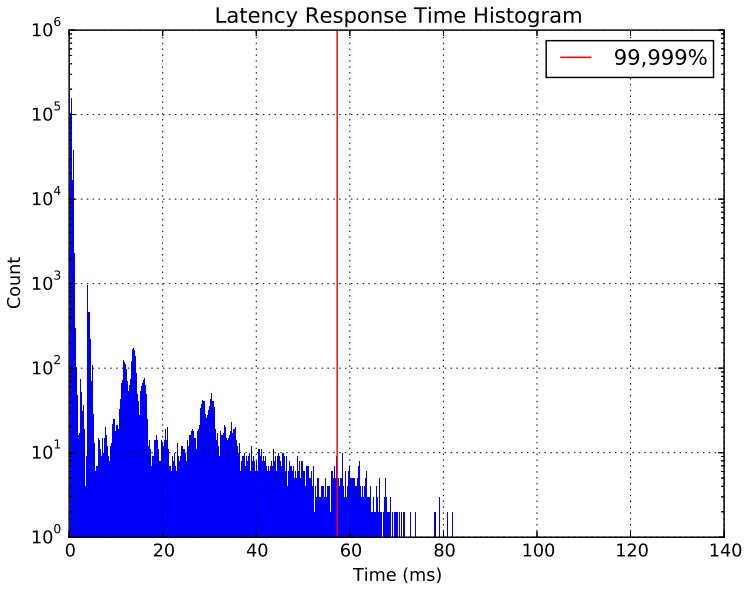


Abbildung 7.7.: Testergebnisse für den 59s-Test(Latenz) für UFS von Hersteller 1

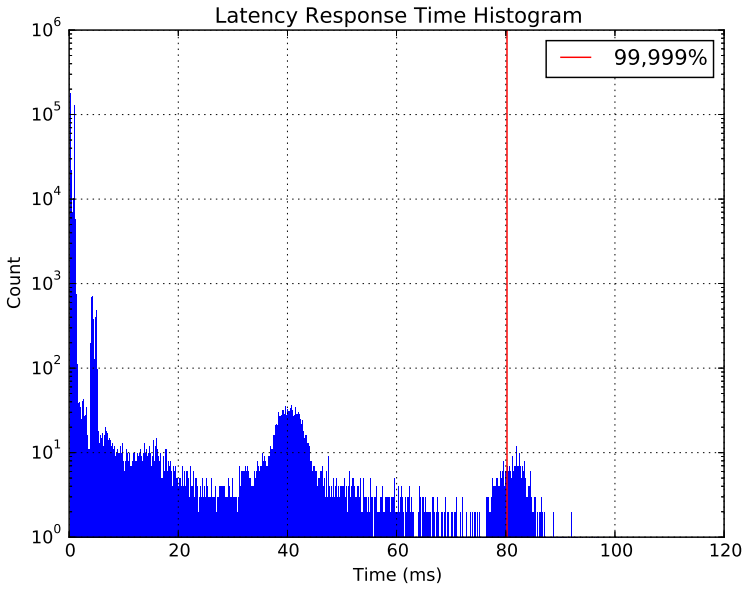


Abbildung 7.8.: Testergebnisse für den 59s-Test(Latenz) für UFS von Hersteller 2

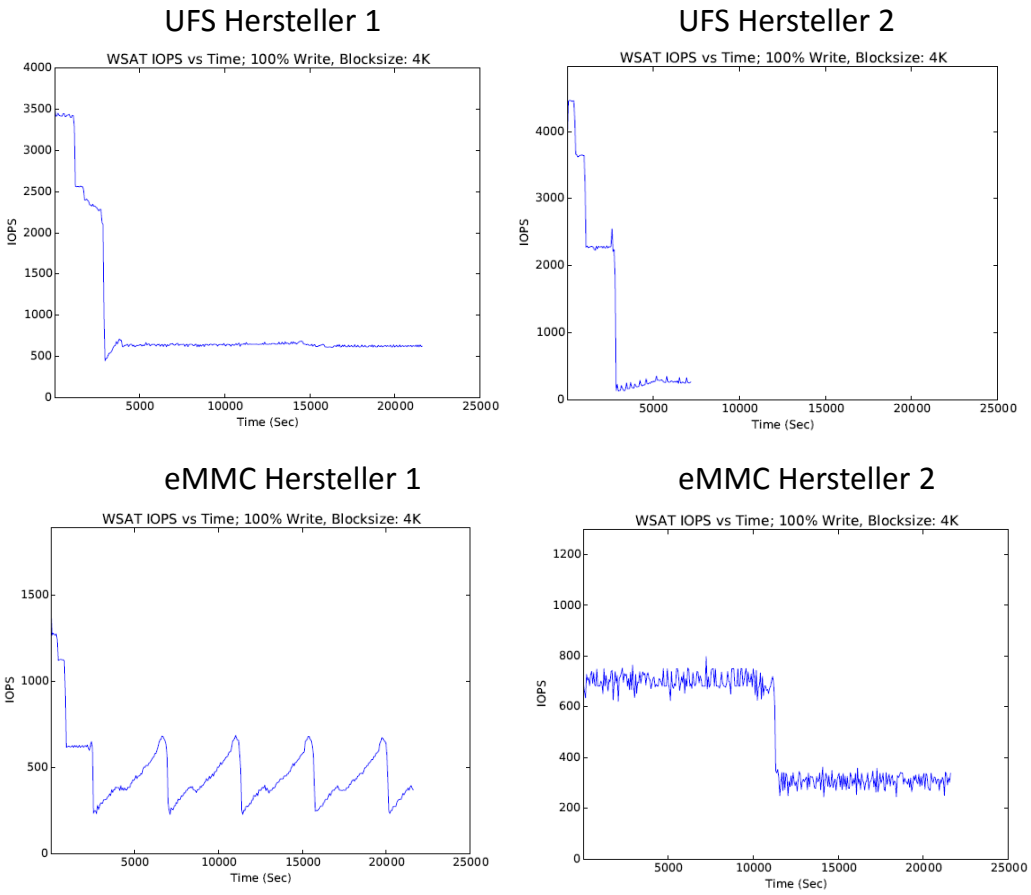


Abbildung 7.9.: Testergebnisse für den WSAT-Test für beide UFS- und eMMC-Speicher

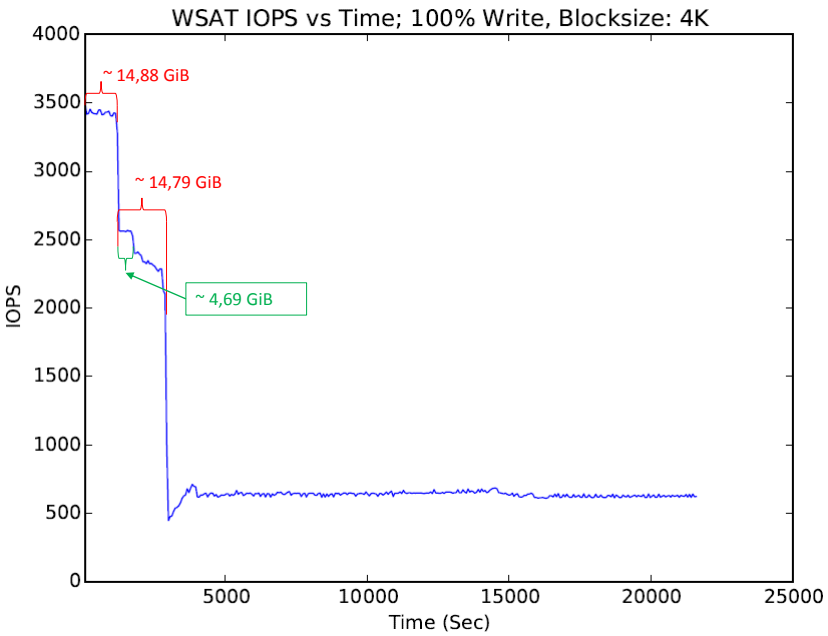


Abbildung 7.10.: Ergebnisse für des WSAT-Tests für UFS 1 mit geschriebener Datenmenge

7.2.6. WAF

Für beide UFS-Speicher wurde ein WAF von ca. 18 gemessen. Wichtig zu erkennen ist, dass dieser Wert für einen absoluten Extremfall für den Speicher gemessen wurde. Der Speicher bekommt also keinerlei Leerlaufzeit für „Aufräumarbeiten“ sondern muss diese parallel zu den eintreffenden Schreibanfragen durchführen. Diese Tatsache kann die Strategie der Firmware stark beeinflussen. Es kann beispielsweise sein, dass *Partial Writes* nicht mehr genutzt werden aus Zeit- (Mappingaufwand) und Platzgründen (es könnte nur ein Block auf einem Die frei sein). Betrachtet man noch einmal Abbildung 7.10, so wird klar, dass sich der Speicher beim WAF-Test zum Großteil im letzten Plateau befindet. Der bereits volle Speicher wird nämlich dauerhaft überschrieben und somit kann keinerlei SLC-Caching genutzt werden (keine freien Blöcke). Schlussfolgernd lässt sich anhand der Ergebnisse eigentlich nur festhalten, dass der WAF von 18 wohl eine Art Grenze für UFS-Speicher im Extremfall darstellt. Um dies wirklich klar festzustellen, müssten noch mehr verschiedene Hersteller getestet werden. Aus Zeitgründen (pro WAF-Test bis zu drei Tage Testdauer) konnte der Test für die beiden eMMCs nicht durchgeführt werden. Zu vermuten ist, dass aktuelle eMMCs einen ähnlichen WAF für diesen Extremfall erzielen würden, da in diesem Belastungszustand die selben Probleme auf beide Speichervarianten zu bewältigen sind.

7.2.7. Stromverbrauch

Abbildung 7.11 zeigt die Resultate des Strom-Tests für beide getesteten eMMC- und UFS-Speicher. Bei allen Speichern verbraucht der Lesevorgang weniger Strom, als der Schreibvorgang. Dies entspricht dem klassischen NAND-Verhalten. Beim Schreiben werden mehrere Schreib-Pulse verwendet um Elektronen auf das Floating Gate der Flash-Zelle zu bringen. Beim Lesen wird lediglich eine Stromstärke gemessen. Hier ist also weniger elektrischer Arbeit

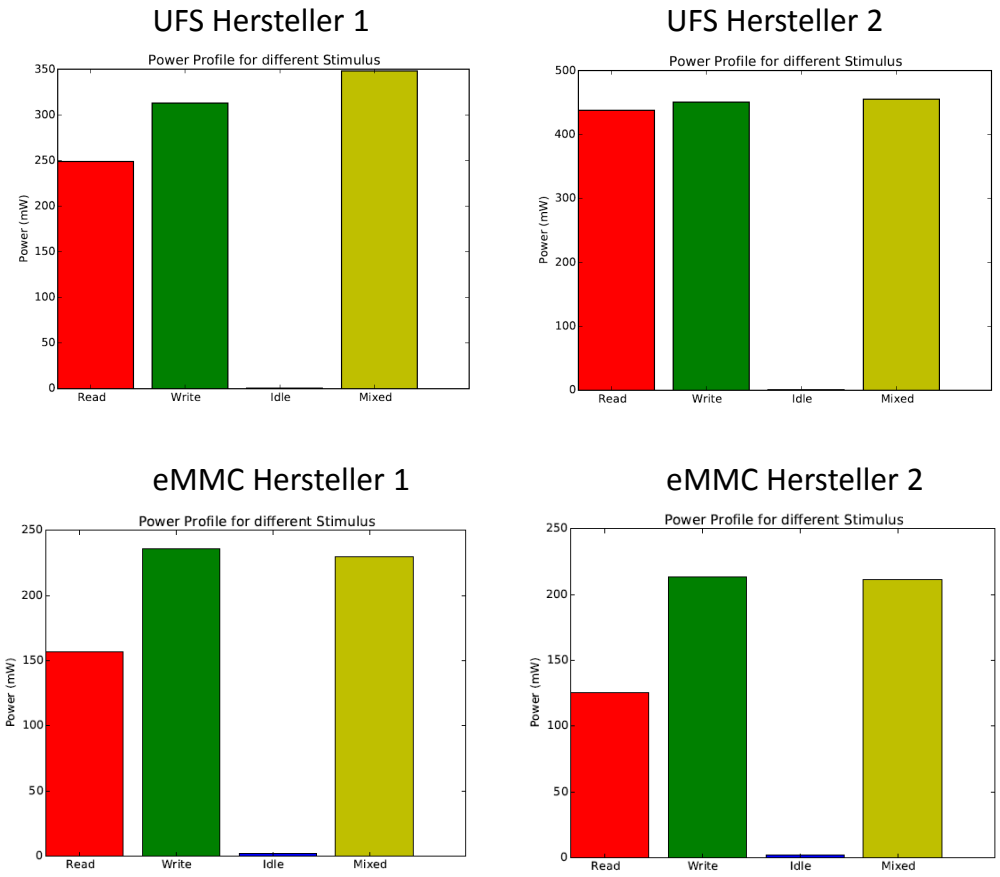


Abbildung 7.11.: Testergebnisse für den Strom-Test für beide UFS- und eMMC-Speicher

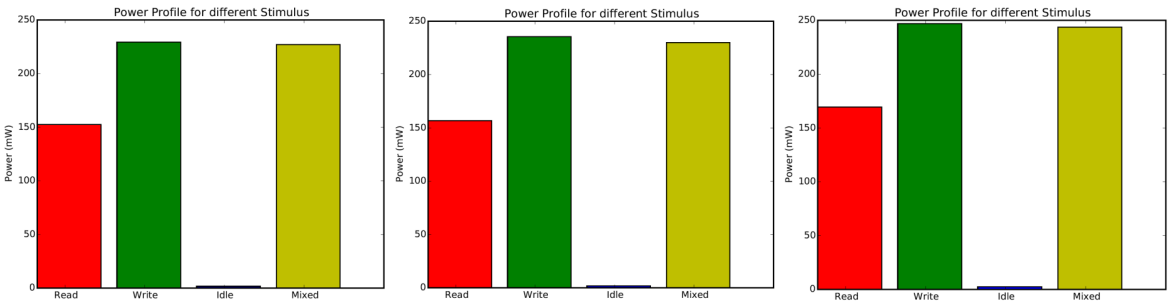


Abbildung 7.12.: Strom-Test für eMMC 2 unter verschiedenen Temperaturbedingungen (-20, 20, 65 Grad Celsius von links nach rechts)

notwendig. Generell verbrauchen die eMMCs bei allen Teststimuli deutlich weniger Strom, als die UFS-Speicher. Nur im Leerlauf verbrauchen UFS-Speicher deutlich weniger (Faktor 4 weniger: 0,4 mW zu 1,83 mW). Sie wurden, wie schon erwähnt, für den Mobiltelefonmarkt auf einen niedrigen Stromverbrauch im Leerlauf optimiert.

Unter den beiden UFS-Speichern gibt es markante Unterschiede in der elektrischen Leistung. Der UFS 1 mit 2D-Technologie und zwei physikalischen Dies verbraucht beim Lesen fast die Hälfte (ca. 56 %) an elektrischer Leistung verglichen mit UFS 2 (3D-Technologie, 1 Die). Auch beim Leerlauf, Schreiben und beim gemischten Workload verbraucht UFS 1 deutlich weniger als UFS 2. Hier könnte entweder der Unterschied von 3D- zu 2D-Technologie, 1 Die zu 2 Dies oder beides verantwortlich sein. Zu beachten ist hier, dass es sich um Speicher selber Kapazität handelt. Die „veraltete“ 2D-Technologie scheint hier deutlich bessere Ergebnisse zu liefern. Das könnte auf eine noch nicht effektive Nutzung der 3D-Technologie hindeuten. Da das Thema noch so neu ist, kann anders als bei der 2D-Technologie, noch nicht auf einen jahrelangen Erfahrungsschatz aufgebaut werden.

Die Unterschiede in der elektrischen Leistung bei den beiden eMMCs können nicht bewertet werden, da es sich um Speicher verschiedener Kapazitäten handelt. Man kann erkennen, dass der Speicher größerer Kapazität (Hersteller 2) stromsparender agiert. Dies könnte auf ein späteres Einsetzten des Garbage Collectors aufgrund mehr Kapazität zurückzuführen sein. Allerdings müssten für valide Erkenntnisse noch weitere Tests durchgeführt werden.

Eine interessante Erkenntnis ist, dass der Stromverbrauch im Leerlauf bei beiden Speichervarianten sehr niedrig im Vergleich zu den anderen Teststimuli ist. Betrachtet man die verbrauchte Leistung von USB-Speichern auf Abbildung 3.6, so erkennt man dass hier der Unterschied zwischen Leerlauf und Schreiben oder Lesen sehr gering ist. Grund dafür ist die Art und Weise, wie ein USB-Speicher betrieben wird. Ein USB-Gerät kann auch ein Gerät, wie eine Maus oder Tastatur sein. Aus diesem Grund muss das Gerät immer mit voller Spannung versorgt werden. Eine Maus darf beispielsweise nicht in einem Schlafzustand sein, da es sonst zu unerwünschten Wartezeiten bei der Benutzung nach einer Pause käme. Somit wäre ein USB-Stick als persistentes Speichermedium in einem Automobil extrem ineffizient.

7.2.8. Temperatureinfluss

Um den Temperatureinfluss auf einen NAND-Speicher zu testen, wurde der eMMC von Hersteller 1 bei drei verschiedenen Temperaturen getestet: -20, 20 und 65 Grad Celsius. Die spezifizierten Extremtemperaturen (-40 und 80 Grad) konnten aufgrund der Tempera-

turbeschränkung des Testboards nicht getestet werden. Es wurden jeweils der IOPS- und der Stromtest durchgeführt. Damit konnte abgeprüft werden, ob es einen Einfluss auf eine Performance-Metrik (IOPS) und auf die Effizienz-Metrik Stromverbrauch gibt. Um verfälschte Ergebnisse durch Vorschädigung auszuschließen, wurde für jede Temperatur ein neuer eMMC des gleichen Typs verwendet. Bei der Performance konnte keinerlei Einfluss festgestellt werden. Alle gemessenen Werte befinden sich im selben Fenster. Beim Stromverbrauch jedoch ergab sich ein Trend, der auf Abbildung 7.12 zu erkennen ist. Hier sieht man die drei Strom-Plots der Stromtests bei verschiedenen Temperaturen. Es zeigt sich für alle Teststimuli mit ansteigender Temperatur ein leichter Anstieg in der elektrischen Leistung. Zwischen -20 und 20 Grad nimmt die Leistung für alle Teststimuli im Schnitt sehr gering zu (ca. 3%). Zwischen Zimmertemperatur und 65 Grad nimmt die verbrauchte Leistung jedoch spürbar zu (ca. 11%). Es scheint, als ob der Speicher in zunehmend wärmerem Bereich zunehmend mehr Leistung verbraucht und als ob die Steigung der Leistungskurve zur höheren Temperatur steiler wird. Dies kann durch den Effekt von Blindströmen erklärt werden (Quelle: Hersteller 1). Je heißer der Speicher wird, desto mehr treten im Material unerwünschte Blindströme, deren Leistung nicht genutzt wird. Folglich muss der Speicher mehr Leistung aufwenden, um diese Blindleistung auszugleichen. Die Blindströme scheinen jedoch erst ab einer bestimmten Grenze stärker zuzunehmen. Um diese Grenze auszuloten müssten noch mehr verschiedene Temperaturen getestet und somit eine Temperaturkurve erzeugt werden. Dies sprengt jedoch den Rahmen dieser Arbeit. Dass die Temperatur jedoch Einfluss auf das Speicherverhalten hat, ist eine wichtige Erkenntnis und macht den Parameter Temperatur essentiell beim Testen der Speichereignung im automotiven Umfeld. Es könnte durchaus sein, dass es auch bei Performance-Metriken bei sehr hohen oder sehr tiefen Temperaturen zu Leistungsverschiebungen kommt.

7.3. Zusammenfassung

Nun können noch die wichtigsten Erkenntnisse zusammengefasste werden. Der wahlfreie Zugriff (Schreiben oder Lesen) ist deutlich langsamer als der sequentielle. Dies entspricht den Erkenntnissen der Quelle in 3.2.4 und war deswegen zu erwarten. Grund dafür war, dass die Daten beim wahlfreien Zugriff auf viele verschiedene Blocks verteilt werden und somit das Überschreiben eines Blocks viele einzelne Blocks gleichzeitig betrifft. Im Gegensatz dazu steht der sequentielle Zugriff, bei dem die Daten viel stärker konzentriert sind.

Die Pausen zwischen den Belastungszeiträumen haben enormen Einfluss auf die Testergebnisse. Für weitere Tests sollte diese Zeit genau bedacht werden, um die gewünschten Ergebnisse zu produzieren. Eine noch längere Pause würde sicher stellen, dass der getestete Speicher nach jeder Belastung alle „Aufräumarbeiten“ abgeschlossen hätte. Eine kürze Pause würde zu einer noch stärken Belastung der FTL führen und möglicherweise in das letzte Plateau des WSAT-Tests. Diese Ergebnisse würden jedoch nicht sehr sinnvoll sein, da eigentlich der SS und nicht der Worst-Case getestet werden soll.

Generell wurde festgestellt, dass sich zwei Speicher der selben Speichervariante und -Kapazität sehr in der Effektivität (elektrische Leistung) und in der Performance unterscheiden können. Dass die Unterschiede so groß sein könnten, war vor den Tests nicht zu erwarten, da der Fokus zu stark auf der Bandbreite der Außenschnittstelle lag. Bei dauerhafter Belastung verschiebt sich das „Bottleneck“ allerdings zunehmend auf die NAND-Schnittstelle. Dann wird es wichtig, wie viele physikalische NAND-Dies und somit parallele Schnittstellen

zu den NAND-Arrays vorhanden sind. Es müssen auch die Speicher-Konfiguration (TLC, MLC oder SLC) und die Technologie (2D- vs. 3D-Technik) beachtet werden. Um wirklich die genaue Auswirkung jeden Parameters zu erforschen, müssen alle anderen Parameter gleich sein. Dies bedarf zusätzlichen Tests. Der Workload für den WAF-Test müsste angepasst werden, um realistische Ergebnisse zu erzielen.

Es wurde außerdem festgestellt, dass die Temperatur einen klaren Einfluss auf die Funktionalität des Speichers hat und somit muss dieser Parameter bei allen Tests eine Rolle spielen. Eine der Haupterkenntnisse ist der enorme Performance-Einfluss des *Pseudo-SLC-Caches*. Gerade bei TLC-Speichern wird durch diese Maßnahme enorm an Performance gewonnen. Allerdings zu Lasten der Langlebigkeit, da die Information immer erst in den Cache und dann als persistente Daten geschrieben werden (beides NAND). Gerade wenn dieses Verfahren bekannt ist, wird es klar, dass die Active Range für kommende Tests angepasst werden sollte. Je kleiner die Active Range und damit mehr Reserveblöcke desto länger kann der Speicher auf einem hohen Leistungslevel performen. Der SLC-Cache-Bereich kann dann für einen längeren Belastungszeitraum genutzt werden. Mit einer Active Range von 100 % und einem vollgeschriebenen Speicher können nur noch sehr kurze hohe Burst-Werte erzielt werden, da der SLC-Cache-Bereich minimal klein ist. Wäre die Active Range niedriger für die Tests dieser Arbeit angesetzt gewesen, so wären wahrscheinlich für sämtliche Performance-Metriken höhere Werte erzielt worden. Dies sollte für zukünftige Tests genauestens bedacht werden.

7.4. Ausblick

Nun soll zum Abschluss dieser Arbeit noch ein kurzer Ausblick auf mögliche zukünftige Anpassungen der Testmethodik folgen. Aus den Erkenntnissen der Zusammenfassung geht hervor, dass die aktuelle Parameterwahl die Performance-Werte im SS des Speichers abgegriffen hat und ein minimaler Bereich für den SLC-Puffer und letztendlich generell für die FTL zur Verfügung gestellt wurde. Es wurden also Werte gemessen, die klar an der unteren Performance-Grenze des Speichers liegen. Des weiteren waren die Testbelastungen entweder sehr statisch (pures Lesen oder Schreiben) oder sehr zufällig (gemischte Workloads). Es kann in zwei Richtungen optimiert werden: Größere Nähe zur gewünschten Anwendung oder den Benchmark-Aspekt optimieren. Beide Varianten wären denkbar. Je näher der Workload an der tatsächlich genutzten Anwendung ist, desto zutreffender und wertvoller sind die gemessenen Werte für die Eignungsprüfung eines Speichers im Hinblick auf ein bestimmten Speicherlastenheftes. Für den Zweck eines Benchmarks allerdings waren die Testreize gut geeignet, weil die Vergleichbarkeit untereinander maximiert wurde.

Soll die Eignungsprüfung optimiert werden, so sollte der Workload mehr in Richtung Applikationsworkload gebracht werden, d.h. also es sollte keine vollkommene Wahlfreiheit genutzt werden. Stattdessen könnten exakte Profile der Testapplikation erstellt werden und für die verschiedenen Speicher als Workload genutzt werden. Auch die Parameterwahl sollte dann näher an der Anwendung sein, als es bei einem Benchmark der Fall wäre. Beispielsweise müsste die Active Range niedriger angesetzt werden, um der FTL mehr Ressourcen für ihre Arbeit zu garantieren und die „Strategie“ des Speicher-Controllers besser auszunutzen. Auch die Pausen zwischen den Belastungen sollten höher angesetzt werden.

Sollte mehr auf den Benchmark-Aspekt der Methodik optimiert werden, so bieten sich auch hier einige Möglichkeiten. Es könnten beispielsweise neue Parameter eingeführt wer-

den. Ein Beispiel hierfür wäre ein Filesystem zu nutzen und hier verschiedene zu testen. Grundsätzlich sollte für zukünftige Benchmarks so gut wie möglich sichergestellt werden, dass ein Parameter isoliert getestet wird (z.B. nur 2-D mit 3-D vergleichen). Alle anderen Parameter sollten also gleich sein, sonst werden eventuell falsche Schlüsse aus den Ergebnissen gezogen. Außerdem könnten neue Tests, die speziell auf den eingebetteten Bereich zugeschnitten sind, hinzugefügt werden. Ein denkbares Beispiel wäre den WSAT-Test aufzugreifen und einen *Flash-Test* daraus zu machen. Anstatt die Performance über Zeit zu überwachen, würde man die totale geschriebene Datenmenge aufzeichnen. So kann herausgefunden werden, welcher Speicher beim Programmieren (vor dem Löten auf die Plattform) besonders schnell ist. Eine niedrige Flash-Zeit kann Kosten einsparen, da der Produktionsprozess verkürzt wird. Auch denkbar wäre es Optimierungstests zu konzipieren. Statt die Leistung nur im SS zu messen, könnten die stattdessen die optimalen Parameter gefunden werden, um möglichst lange eine bestimmte Performance zu halten.

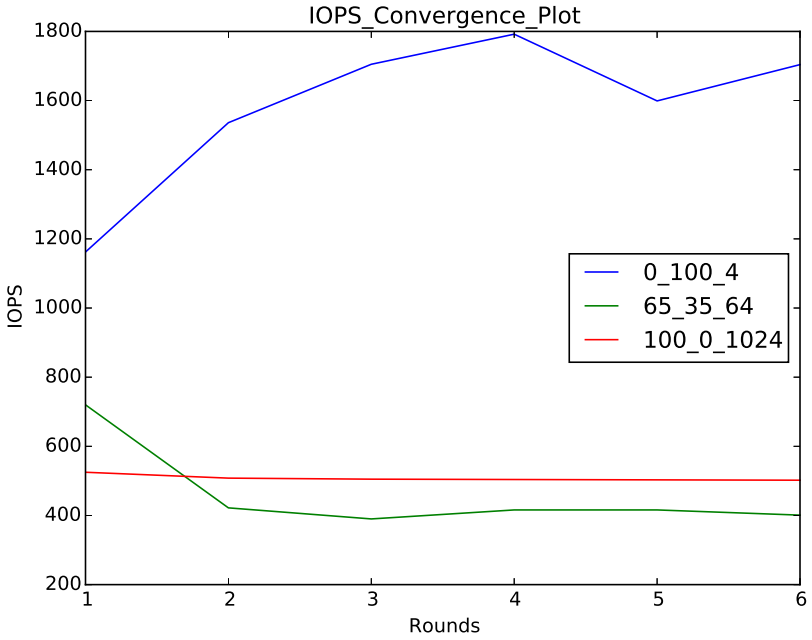
Würden beide Aspekte optimiert könnte sich das auf zwei Weisen positiv auf das Endprodukt auswirken. Die anwendungsnahe Eignungsprüfung könnte sofort genutzt werden um den richtigen Speicher für die direkt anstehenden, schon entwickelten Anwendungen, zu finden. Es kann also die bestehende Anwendung auf der optimalen Speicher-Hardware ausgeführt und somit Komplikationen verhindert werden. Der Benchmark und somit die verstärkte Erforschung des Speicherverhaltens und die Findung von Optimal-Szenarien wirken sich zeitlich versetzt aus. Das gesammelte Wissen kann noch nicht fertige Anwendungen im Vorfeld beeinflussen. Die Software-Entwickler können also durch das Spezial-Wissen der Speicherexperten ihre Applikationen so konzipieren, dass sie im optimalen Leistungsbereich der verwendeten Speicher arbeiten. Es kommt also zu Synergien in beide Richtungen. Durch den Austausch der zwei Parteien können aktuelle Applikationen besser genutzt werden und zukünftige schon intelligenter konzipiert werden. Somit wird für den Besitzer des modernen Automobils das bestmögliche Fahrerlebnis durch die fortschrittlichsten Applikationen ermöglicht.

A. Anhang UFS Hersteller 1 Messdaten

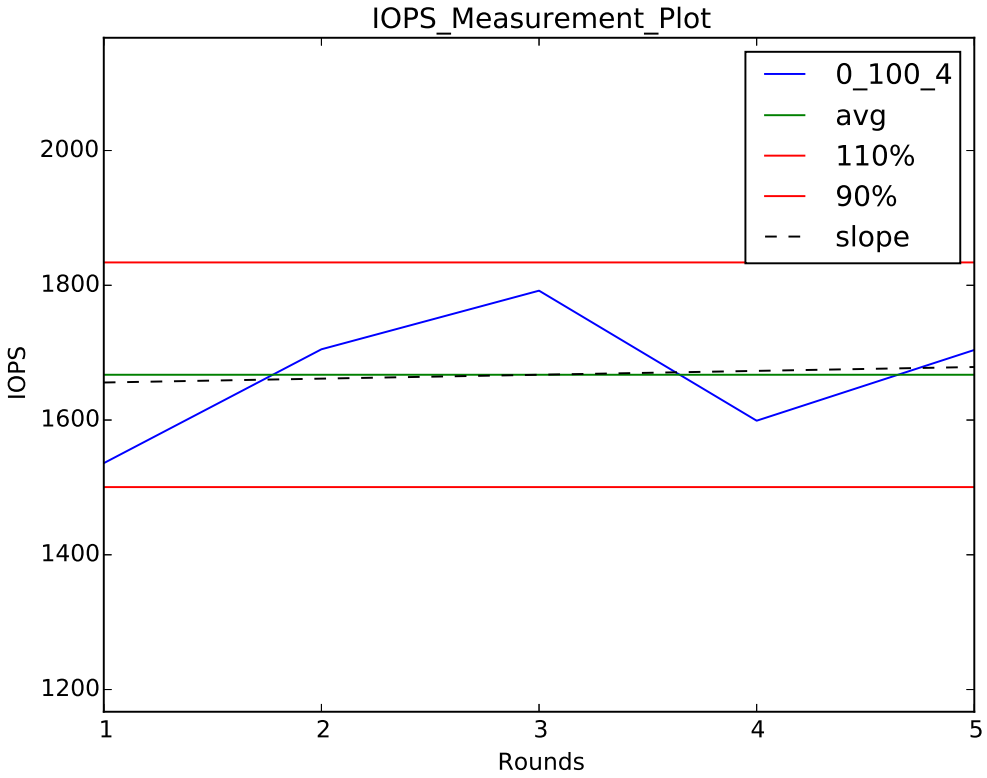
Die folgenden Grafiken zeigen den vollständigen Output, den der Prototyp bei einem Benchmark-Test produziert.

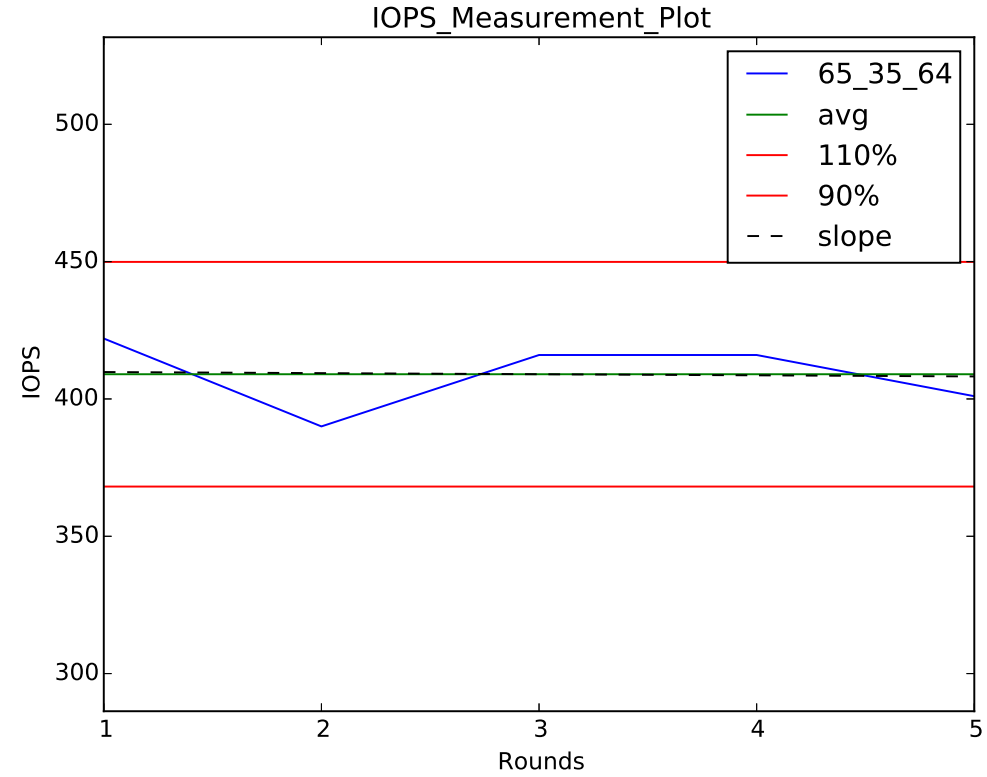
A.1. IOPS

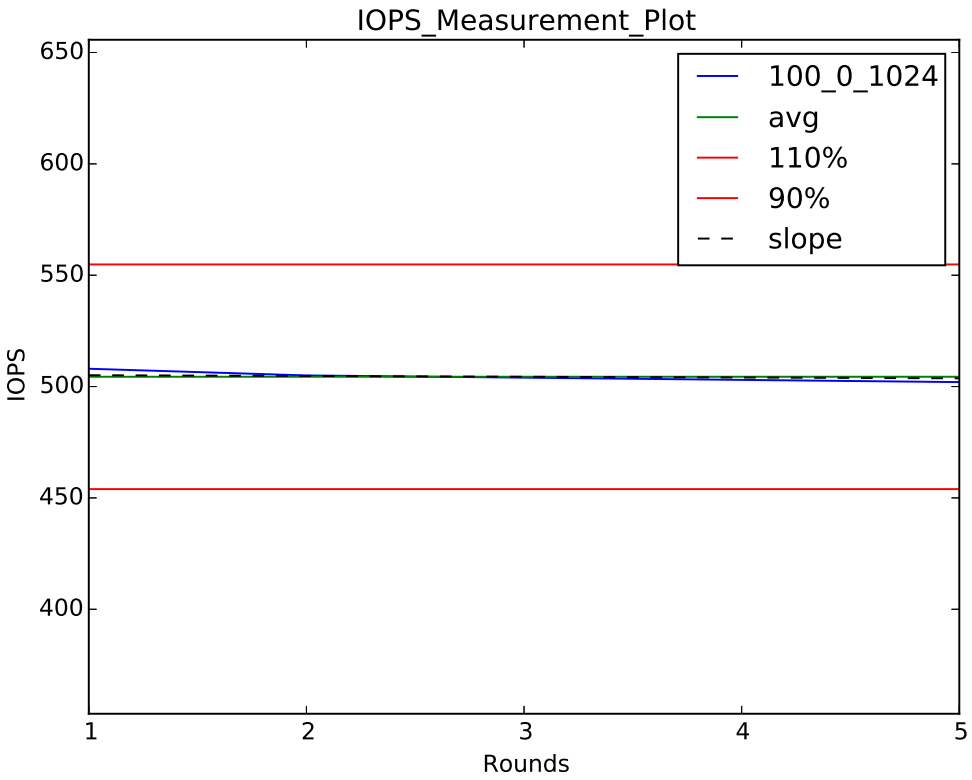
Alle Output-Dateien des IOPS-Tests:

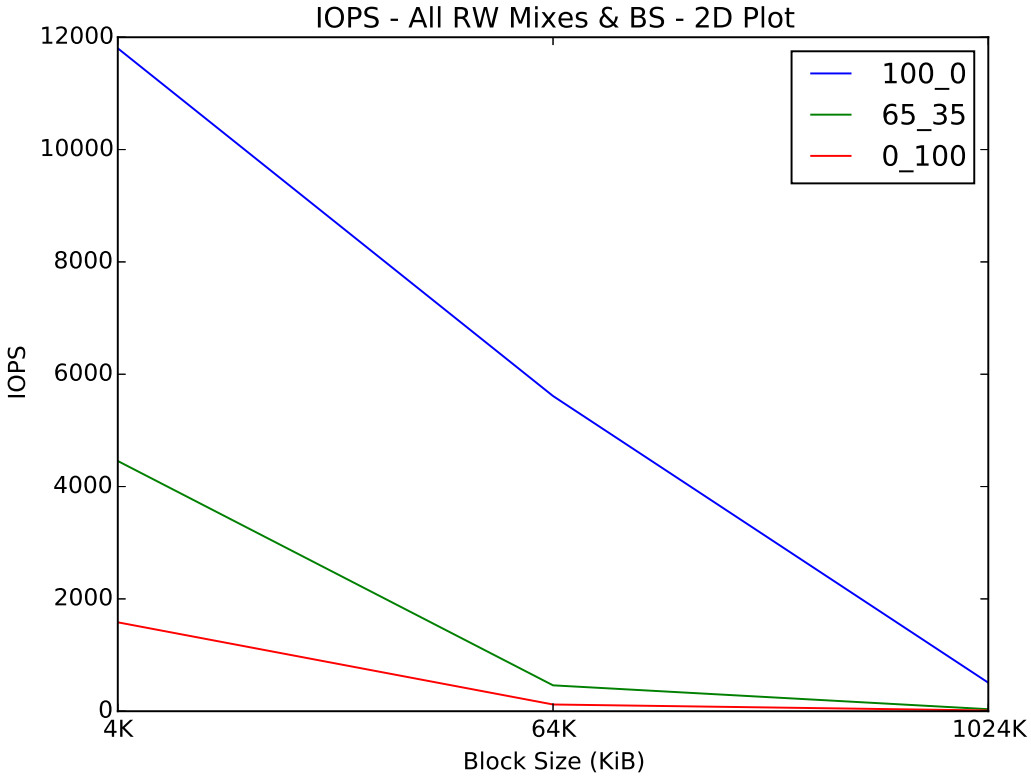


	0_100_4	65_35_64	100_0_1024
Measurment Window	2 - 6	2 - 6	2 - 6
Ave Value	1667.20	409.00	504.40
Allowed Range	1500.48 - 1833.92	368.1 - 449.9	453.96 - 554.84
Measured Range	1536.00 - 1792.00 (passed)	390.00 - 422.00 (passed)	502.00 - 508.00 (passed)
Slope BLF	23.00 (passed)	-1.60 (passed)	-1.40 (passed)
Correlation Coeff BLF	0.36	-0.19	-0.96


IOPS_Measurement_Plot
0_100_4
avg
110%
90%
slope
2000
1800
1600
1400
1200
IOPS
1
2
3
4
5
Rounds


IOPS_Measurement_Plot
65_35_64
avg
110%
90%
slope
500
450
400
350
300
IOPS
1
2
3
4
5
Rounds


IOPS_Measurement_Plot
650
600
550
500
450
400
IOPS
1
2
3
4
5
Rounds
100_0_1024
avg
110%
90%
slope

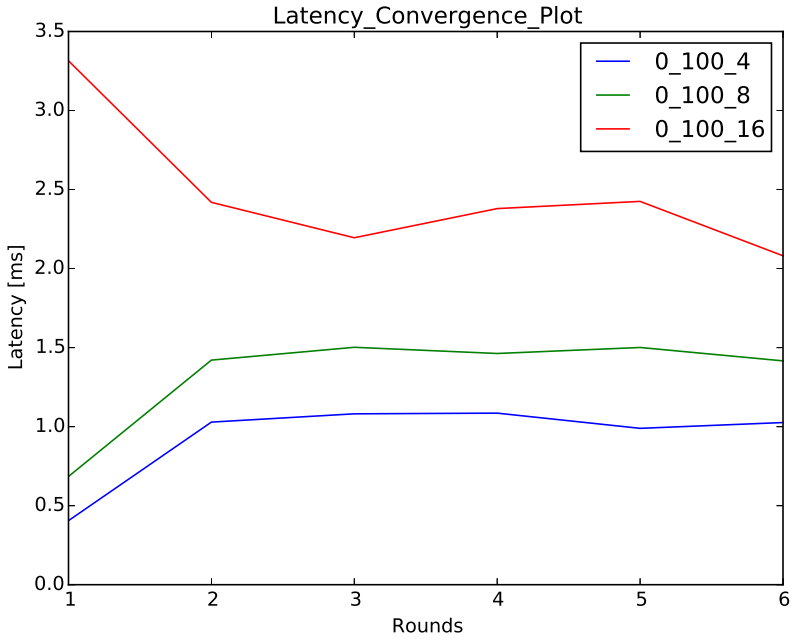

IOPS - All RW Mixes & BS - 2D Plot
12000
10000
8000
6000
4000
2000
0
IOPS
4K
64K
1024K
Block Size (KiB)
100_0
65_35
0_100

IOPS(in) - All RW Mix & BS - Tabular Data

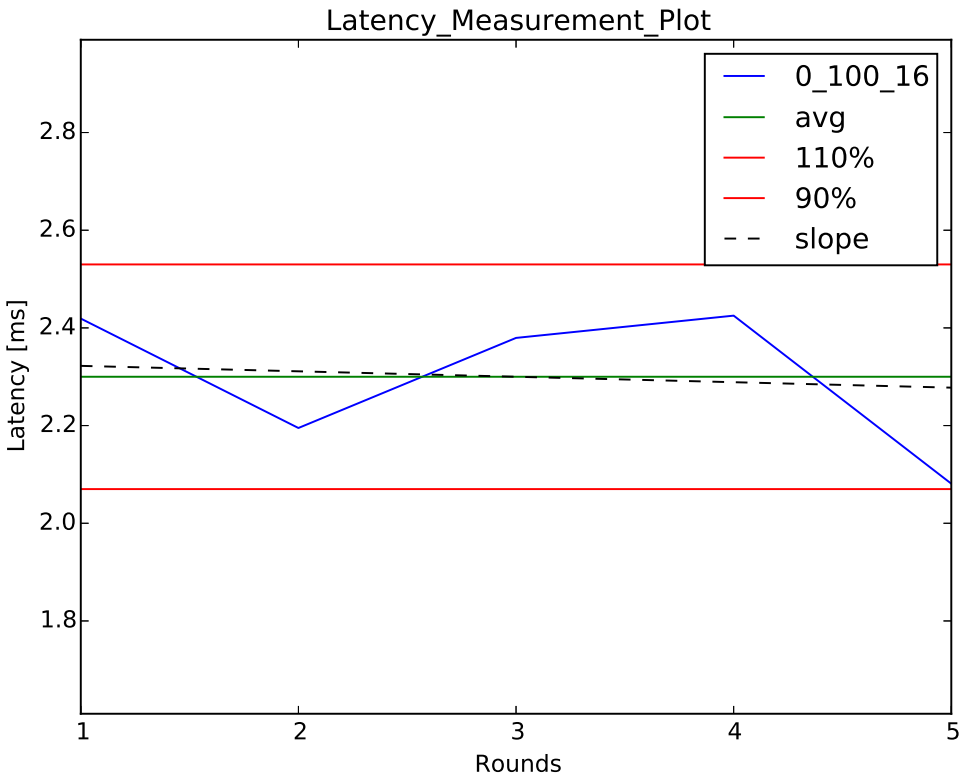
	100_0	65_35	0_100
4K	11803.0	4455.333	1583.0
64K	5612.0	460.833	119.833
1024K	507.833	36.0	13.167

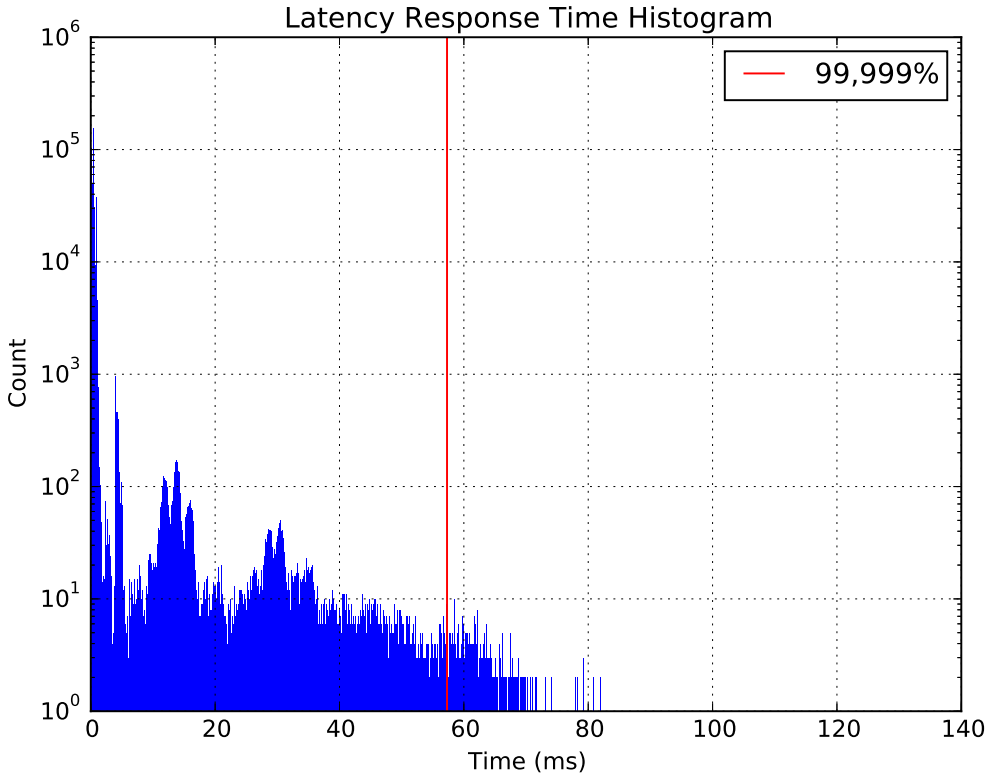
A.2. Latenz

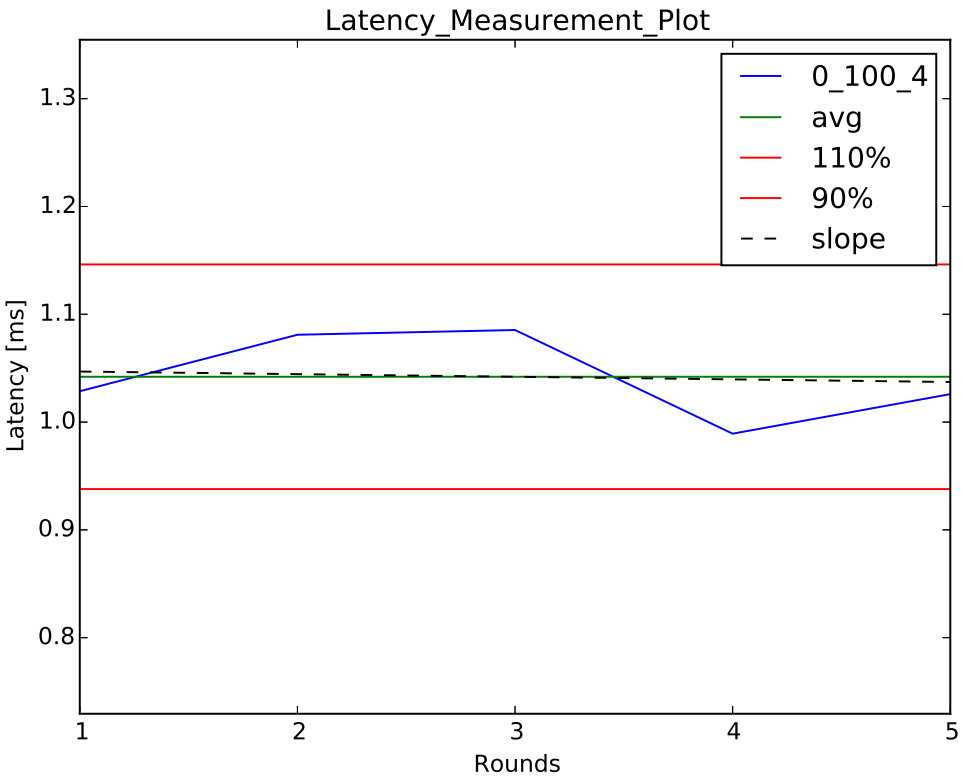
Alle Output-Dateien des Latenz-Tests:

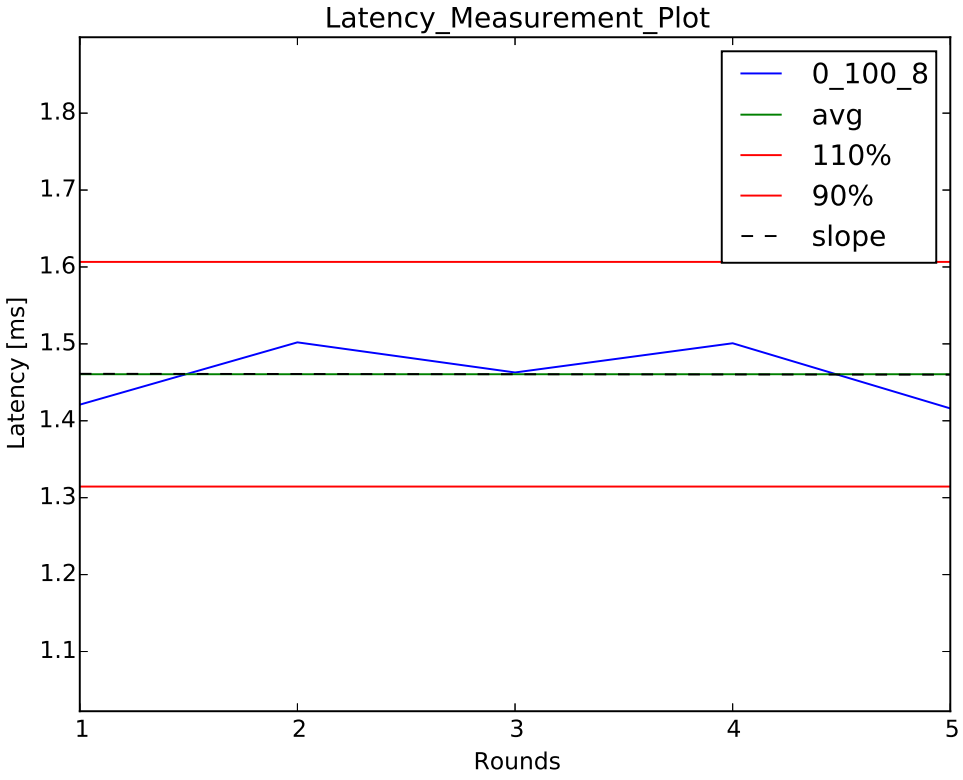


	0_100_4	0_100_8	0_100_16
Measurment Window	2 - 6	2 - 6	2 - 6
Ave Value	1.04	1.46	2.30
Allowed Range	0.93783470238 - 1.14624241402	1.31447589714 - 1.60658165206	2.06997081066 - 2.52996432414
Measured Range	0.99 - 1.09 (passed)	1.42 - 1.50 (passed)	2.08 - 2.43 (passed)
Slope BLF	-0.01 (passed)	-0.00 (passed)	-0.04 (passed)
Correlation Coeff BLF	-0.38	-0.04	-0.46


Latency_Measurement_Plot
0_100_16
avg
110%
90%
slope
Latency [ms]
2.8
2.6
2.4
2.2
2.0
1.8
1
2
3
4
5
Rounds


Latency Response Time Histogram
99,999%
Count
10^6
10^5
10^4
10^3
10^2
10^1
10^0
0
20
40
60
80
100
120
140
Time (ms)


Latency_Measurement_Plot
0_100_4
avg
110%
90%
slope
Latency [ms]
1.3
1.2
1.1
1.0
0.9
0.8
1
2
3
4
5
Rounds


Latency_Measurement_Plot
0_100_8
avg
110%
90%
slope
Latency [ms]
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
2
3
4
5
Rounds

Latency(5nines in msec) - All RW Mix & BS - Tabular Data

	100_0	65_35	0_100
4K	1.141	32.059	52.137
8K	1.12	32.026	52.148
16K	1.128	32.548	52.046

Latency(max_lat in msec) - All RW Mix & BS - Tabular Data

	100_0	65_35	0_100
4K	5.718	52.375	98.031
8K	5.385	49.822	76.782
16K	6.06	50.164	77.31

Latency(avg_lat in msec) - All RW Mix & BS - Tabular Data

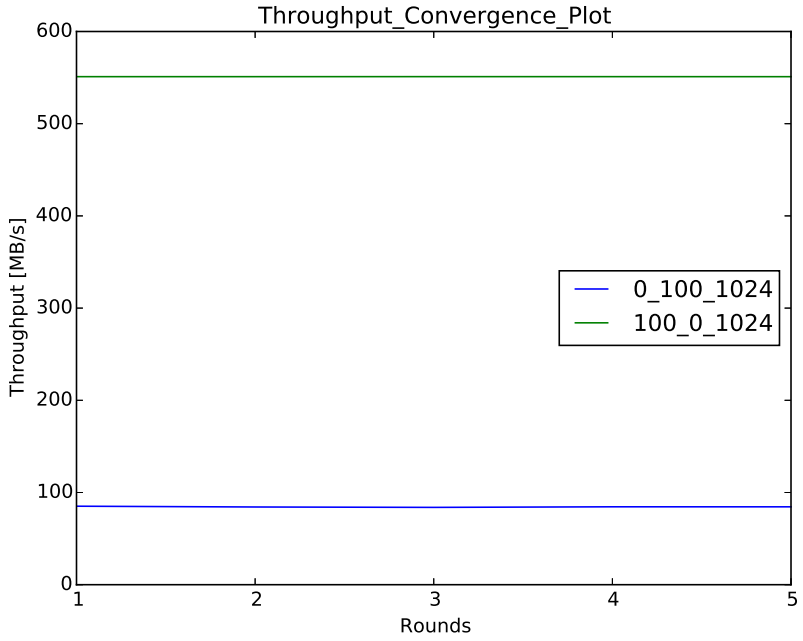
	100_0	65_35	0_100
4K	0.401	0.428	0.936
8K	0.393	0.404	1.331
16K	0.441	0.51	2.469

Latency(iops in iops) - All RW Mix & BS - Tabular Data

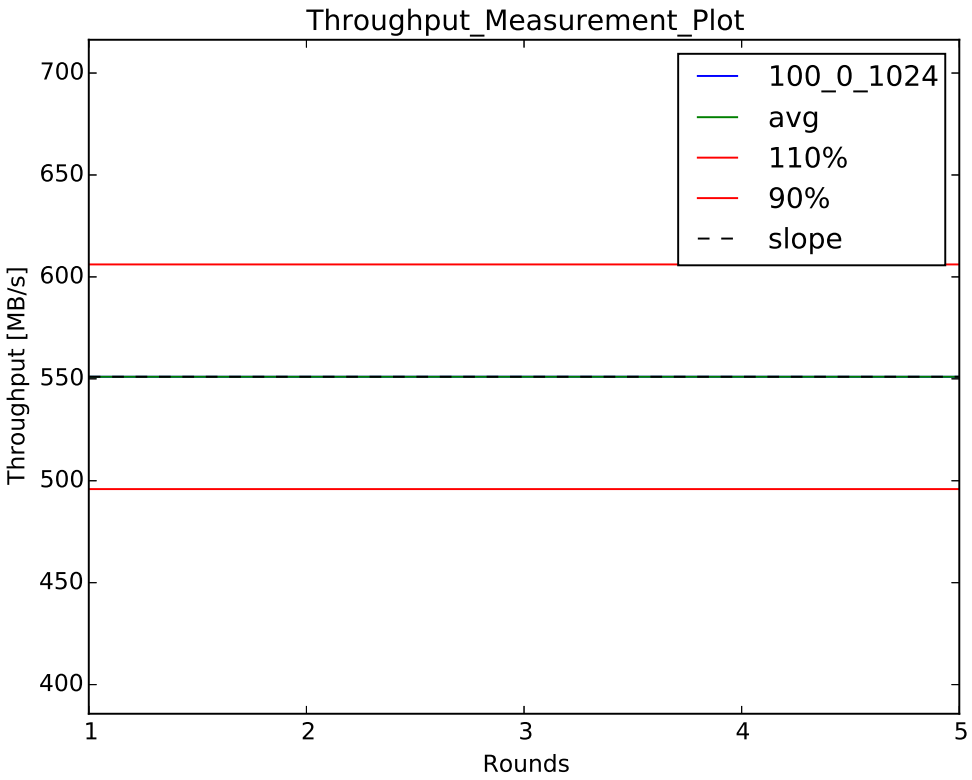
	100_0	65_35	0_100
4K	2457.167	1245.0	1194.5
8K	2520.333	1337.667	805.667
16K	2245.667	1083.0	410.667

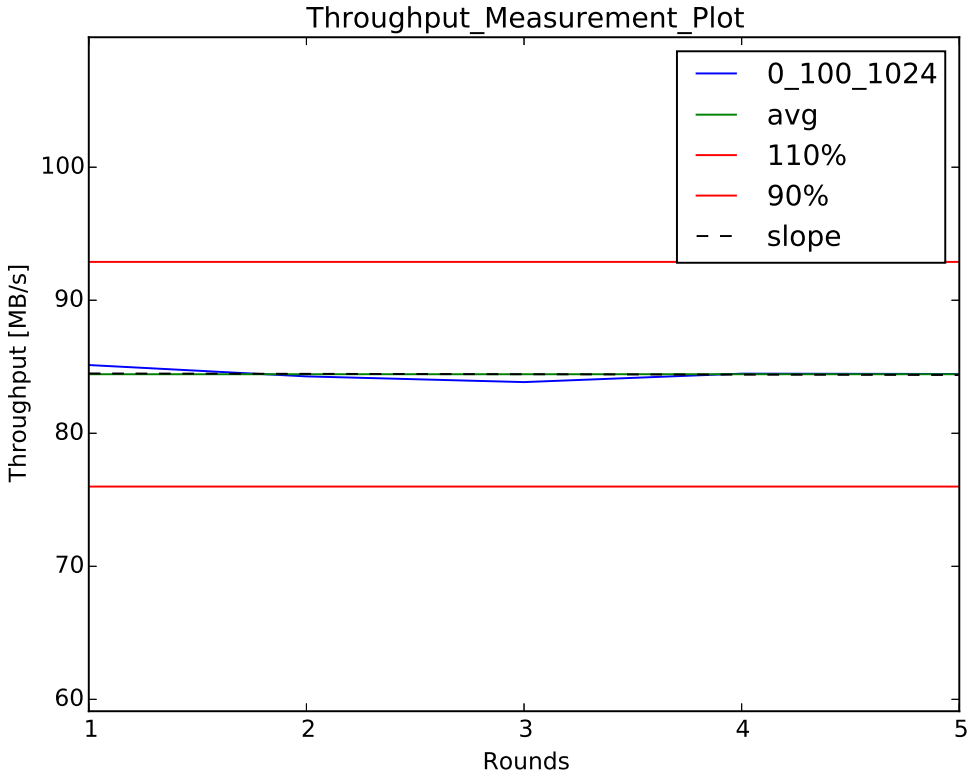
A.3. Durchsatz

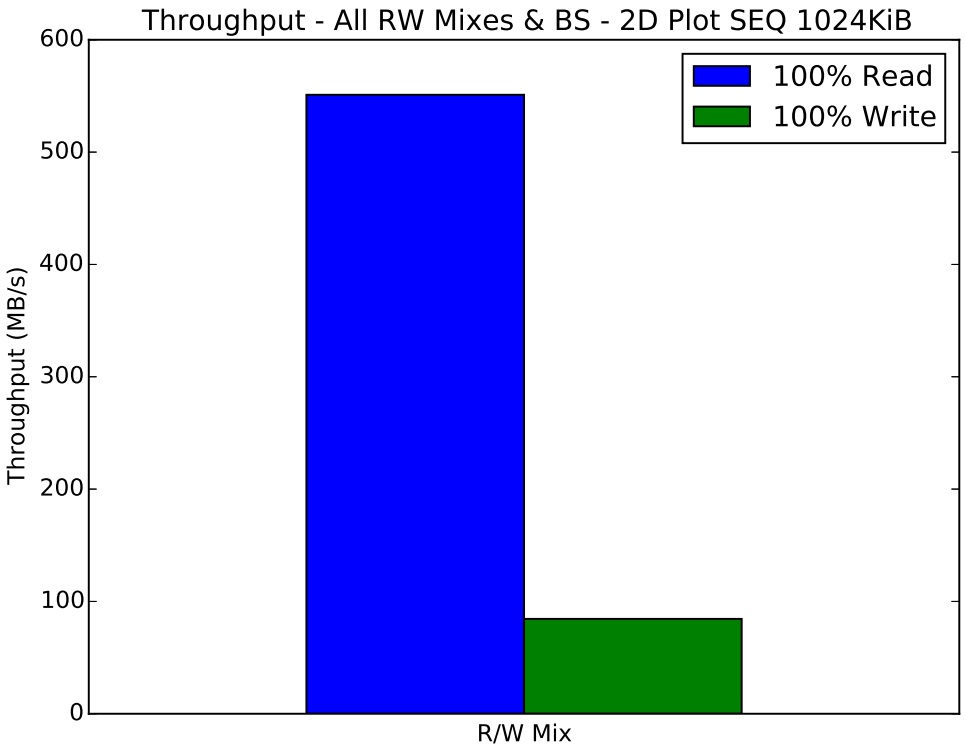
Alle Output-Dateien des Durchsatz-Tests:



	0_100_1024	100_0_1024
Measurment Window	1 - 5	1 - 5
Ave Value	84.44	551.02
Allowed Range	75.99439872 - 92.88204288	495.91351296 - 606.11651584
Measured Range	83.85 - 85.13 (passed)	550.99 - 551.04 (passed)
Slope BLF	-0.11 (passed)	-0.00 (passed)
Correlation Coeff BLF	-0.39	-0.26


Throughput_Measurement_Plot
100_0_1024
avg
110%
90%
slope
Throughput [MB/s]
700
650
600
550
500
450
400
1
2
3
4
5
Rounds


Throughput_Measurement_Plot
0_100_1024
avg
110%
90%
slope
Throughput [MB/s]
100
90
80
70
60
1
2
3
4
5
Rounds

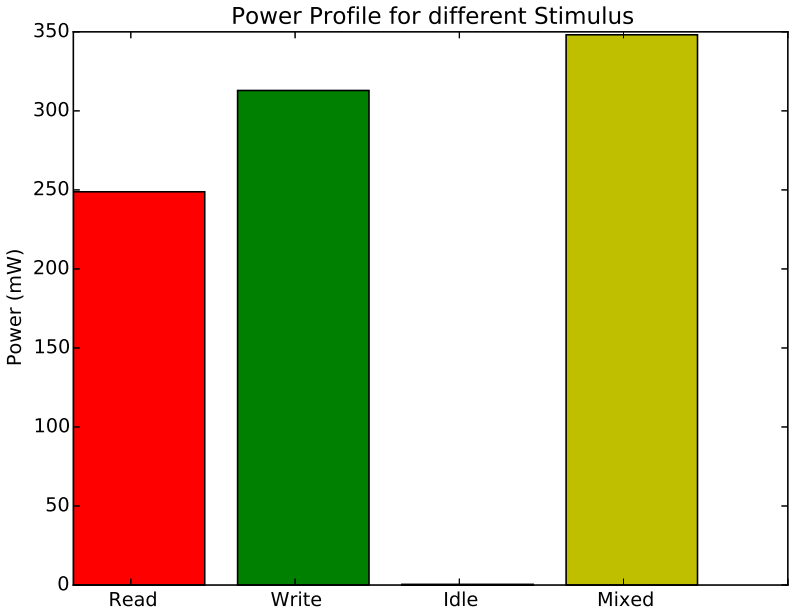

Throughput - All RW Mixes & BS - 2D Plot SEQ 1024KiB
600
500
400
300
200
100
0
Throughput (MB/s)
R/W Mix
100% Read
100% Write

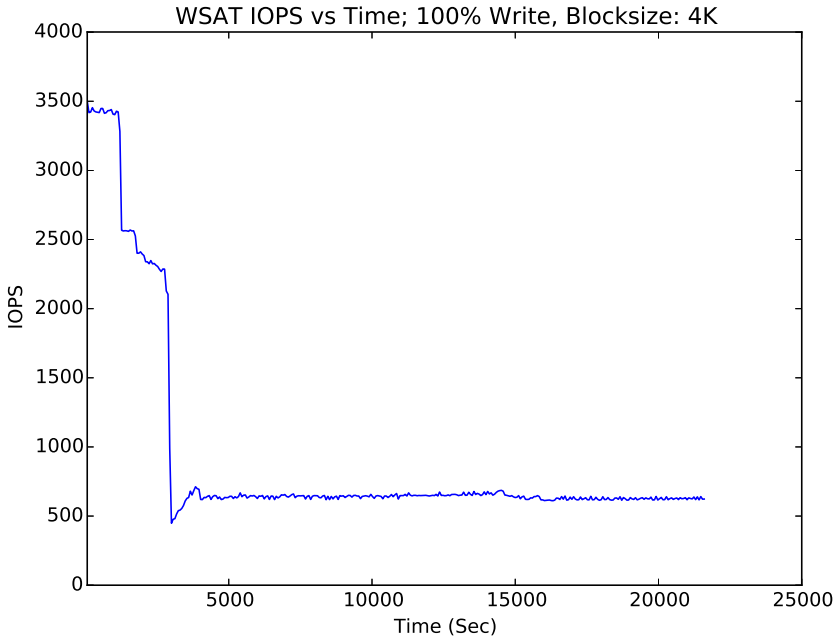
Throughput(in (MB/sec)) - All RW Mix & BS - Tabular Data

	100_0	0_100
1024K	551.015	84.438

A.4. Stromtest und WSAT

Alle Output-Dateien des Stromtests und des WSAT:





A.5. Berichte

Alle Berichte, die nach einer kompletten Testreihe ausgeben werden:

```
1
2   _____________
3   RESULT REPORT
4   _____________
5
6   ----------Diagnosis Section----------
7   Feature Support
8
9
10  SecLU: Passed
11  FFU: Passed
12  PSA: Failed
13  DevLifeSpan: Failed
14  OOData: Failed
15  NormalType: Passed
16  SystemCode: Passed
17  Non_Persistent: Passed
18  Enhanced_Type1: Passed
19  Enhanced_Type2: Passed
20  RPMB: Passed
21
22
23  Inherent Information
24
25  UserCap: Expected Value: 0  Diagnosed Value: 32006733824  ---> Passed
26  MaxLU: Expected Value: 0    Diagnosed Value: 8  ---> Passed
27  NumSWPA: Expected Value: 0  Diagnosed Value: 32  ---> Passed
28  SysCodeMaxSize: Expected Value: 0   Diagnosed Value: 32006733824  ---> Passed
29  NonPersistentMaxSize: Expected Value: 0     Diagnosed Value: 32006733824  ---> Passed
30
31
32  ----------Performance Section----------
33
34  IOPS:
35
36  RW-Mix: 100_0 BlockSize: 4  Expected Value: 0 Measured Value: 11803.0 ---> Passed
37  RW-Mix: 100_0 BlockSize: 64  Expected Value: 0 Measured Value: 5612.0 ---> Passed
38  RW-Mix: 100_0 BlockSize: 1024  Expected Value: 0 Measured Value: 507.833333333 --->
    Passed
39  RW-Mix: 0_100 BlockSize: 4  Expected Value: 0 Measured Value: 1583.0 ---> Passed
40  RW-Mix: 0_100 BlockSize: 64  Expected Value: 0 Measured Value: 119.833333333 --->
    Passed
41  RW-Mix: 0_100 BlockSize: 1024  Expected Value: 0 Measured Value: 13.1666666667 --->
    Passed
42  RW-Mix: 65_35 BlockSize: 4  Expected Value: 0 Measured Value: 4455.33333333 --->
    Passed
43  RW-Mix: 65_35 BlockSize: 64  Expected Value: 0 Measured Value: 460.833333333 --->
    Passed
44  RW-Mix: 65_35 BlockSize: 1024  Expected Value: 0 Measured Value: 36.0 ---> Passed
45
46  Write Amplification Factor:
47
48  Expected WAF: 0
49  Measured WAF: 18.114987715   ---> Passed
50
51  Throughput (MB/s):
52
53  RW-Mix: 0_100 BlockSize: 1024  Expected Value: 0 Measured Value: 84.4382208 --->
    Passed
54  RW-Mix: 100_0 BlockSize: 1024  Expected Value: 0 Measured Value: 551.0150144 --->
    Passed
55
56  Latency (msec):
57
58  RW-Mix: 100_0 BlockSize: 4  Expected Value: 0 Measured Value: 0.401109464333 --->
    Passed
59  RW-Mix: 100_0 BlockSize: 8  Expected Value: 0 Measured Value: 0.3925817145 ---> Passed
60  RW-Mix: 100_0 BlockSize: 16  Expected Value: 0 Measured Value: 0.441287303333 --->
    Passed
61  RW-Mix: 0_100 BlockSize: 4  Expected Value: 0 Measured Value: 0.935900028333 --->
    Passed
62  RW-Mix: 0_100 BlockSize: 8  Expected Value: 0 Measured Value: 1.33123982833 --->
    Passed
```

```
63  RW-Mix: 0_100 BlockSize: 16  Expected Value: 0 Measured Value: 2.4691110325 --->
    Passed
64  RW-Mix: 65_35 BlockSize: 4  Expected Value: 0 Measured Value: 0.427520497392 --->
    Passed
65  RW-Mix: 65_35 BlockSize: 8  Expected Value: 0 Measured Value: 0.404133647983 --->
    Passed
66  RW-Mix: 65_35 BlockSize: 16  Expected Value: 0 Measured Value: 0.509692284479 --->
    Passed
67
68  ----------Power Section----------
69
70  Average Consumed Power (mW):
71
72  Read: Expected Value: 0 Measured Value: 248.80542989850954   ---> Passed
73  Write: Expected Value: 0    Measured Value: 312.9047465141357    ---> Passed
74  Idle: Expected Value: 0 Measured Value: 0.43285668383462195  ---> Passed
75  Mixed: Expected Value: 0    Measured Value: 348.1271586959069    ---> Passed
```

Abbildung A.1.: Ergebnisbericht für UFS-Benchmark

```
1
2   _____________
3   SETUP REPORT
4   _____________
5
6        TEST PLATFORM HARDWARE
7
8   Manufacturer/Model: HowChip HC7420-UFSeMM
9   CPU: Exynos 7420 ARM Cortex A57 Quad, A53 Quad @1.8GHz
10  Memory: LPDDR4 3GB 1600MHz POP type
11  Primary Storage: SD Card (SD 3.0)
12
13       TEST PLATFORM SOFTWARE
14
15  Operating System: Ubuntu 15.04, Kernel 3.10.61
16  Filesystem: EXT4
17  Test Software: Fio 3.7
18  Interface UFS: UFS 2.0, HS-Gear 2   Interface eMMC: eMMC 5.1, HS400
19
20       DEVICE UNDER TEST
21
22  Vendor: ********
23  Model: ********
24  Rev: 0100
25  Unit serial number: ********
26  User Capacity: 32006733824 Bytes
27
28       TEST SETUP
29
30  IOPS
31
32  Preconditioning Parameters
33
34  Data Pattern: Random
35  AR: 100%
36  Device Independent Workload: SEQ 1024 KiB W
37  Duration: 2X User Capacity
38  Process Count: 8; Thread Count: 1, Queue Depth: 1
39
40  Test Parameters
41
42  Data Pattern: Random
43  AR: 100%
44  RW-Mixes: 100_0, 65_35, 0_100
45  Block Sizes: 4K, 64K, 1024K
46  Duration: 60 sec Loops; up to Steady State or 25 Rounds
47  Device Dependent Workload: Test Loop for all Combinations of RW-Mixes and BlockSizes
48  Process Count: 8; Thread Count: 1, Queue Depth: 1
49  Access Pattern: RND
50
51  WAF
52
53  Preconditioning Parameters
54
55  Follow up to IOPS --> no separate Preconditioning
56
57
58  Test Parameters
59
60  Data Pattern: Random
61  AR: 100%
62  RW-Mixes: 0_100
63  Block Sizes: 4K, 16K, 64K, 1024K
64  Duration: 60 sec Loops; up to Steady State or 25 Rounds
65  Device Dependent Workload: Test Loop for all Combinations of RW-Mixes and BlockSizes
66  Process Count: 8; Thread Count: 1, Queue Depth: 1
67  Access Pattern: RND
68
69  Throughput
70
71  Preconditioning Parameters
72
73  Data Pattern: Random
```

```
74  AR: 100%
75  Device Independent Workload: SEQ 1024 KiB W
76  Duration: 2X User Capacity
77  Process Count: 8; Thread Count: 1, Queue Depth: 1
78
79  Test Parameters
80
81  Data Pattern: Random
82  AR: 100%
83  RW-Mixes: 100_0, 0_100
84  Block Sizes: 1024K
85  Duration: 60 sec Loops; up to Steady State or 25 Rounds
86  Device Dependent Workload: Test Loop for all Combinations of RW-Mixes and BlockSizes
87  Process Count: 8; Thread Count: 1, Queue Depth: 1
88  Access Pattern: SEQ
89
90
91  Latency
92
93  Preconditioning Parameters
94
95  Data Pattern: Random
96  AR: 100%
97  Device Independent Workload: SEQ 1024 KiB W
98  Duration: 2X User Capacity
99  Process Count: 8; Thread Count: 1, Queue Depth: 1
100
101 Test Parameters
102
103 Data Pattern: Random
104 AR: 100%
105 RW-Mixes: 100_0, 65_35, 0_100
106 Block Sizes: 4K, 8K, 16K
107 Duration: 60 sec Loops; up to Steady State or 25 Rounds
108 Device Dependent Workload: Test Loop for all Combinations of RW-Mixes and BlockSizes
109 Process Count: 8; Thread Count: 1, Queue Depth: 1
110 Access Pattern: RND
111
112
113 WSAT
114
115 Preconditioning Parameters
116
117 No Preconditioning
118
119
120 Test Parameters
121
122 Data Pattern: Random
123 AR: 100%
124 RW-Mixes: 0_100
125 Block Sizes: 4K
126 Duration: 6 hours
127 Device Dependent Workload: Test Loop for all Combinations of RW-Mixes and BlockSizes
128 Process Count: 8; Thread Count: 1, Queue Depth: 1
129 Access Pattern: RND
130
131
```

Abbildung A.2.: Setup-Bericht für UFS-Benchmark

Abbildungsverzeichnis

Literaturverzeichnis

[ARI16] ARITOME, SEIICHI: *NAND flash memory technologies.* Hoboken : Wiley IEEE Press, 2016 (IEEE press series on microelectronic systems). – ISBN 978–1119132608

[BCMV03] Bez, R. ; Camerlenghi, E. ; Modelli, A. ; Visconti, A.: Introduction to flash memory. In: *Proceedings of the IEEE* 91 (2003), Nr. 4, S. 489–502. http://dx.doi.org/10.1109/JPROC.2003.811702. – DOI 10.1109/JPROC.2003.811702. – ISSN 0018–9219

[BU10] Brinkschulte, Uwe ; Ungerer, Theo: *Mikrocontroller und Mikroprozessoren.* Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2010 (eXamen.press). http://dx.doi.org/10.1007/978-3-642-05398-6. http://dx.doi.org/10.1007/978-3-642-05398-6. – ISBN 978–3642053979

[Che08] Chen, Yuan: Flash Memory Reliability NEPP 2008 Task Final Report. In: *NASA Electronic Parts and Packaging (NEPP) Program* (2008)

[Cou16] Coughlin, Tom: The Memory of Cars [The Art of Storage]. In: *IEEE Consumer Electronics Magazine* 5 (2016), Nr. 4, S. 121–125. http://dx.doi.org/10.1109/MCE.2016.2590238. – DOI 10.1109/MCE.2016.2590238. – ISSN 2162–2248

[Des10] Desnoyers, Peter: Empirical evaluation of NAND flash memory performance. In: *ACM SIGOPS Operating Systems Review* 44 (2010), Nr. 1, S. 50. http://dx.doi.org/10.1145/1740390.1740402. – DOI 10.1145/1740390.1740402. – ISSN 01635980

[EAW] Ekker, Neal ; Amer, Khaled ; Wassenberg, Paul: Solid State Storage: Performance Test Specification Whitepaper: Storage Networking Industry Association (SNIA). 2010

[Ent] Entegris: Considerations for Improving 3D NAND Performance, Reliability, and Yield. https://www.entegris.com/content/dam/web/resources/white-papers/whitepaper-improving-3d-nand-performance-8443.pdf

[Gua13] Guanying Wu: *Performance and Reliability Study and Exploration of NAND Flash-based Solid State Drives: Zugl. Dissertation an der Virginia Commonwealth University.* Virginia, USA : VCU Scholars Compass, 2013

[Int10a] International Committe for Information Technology Standards: *SCSI Block Commands - 3 (SBC-3).* Revision 24. 05.08.2010

[Int10b] International Committe for Information Technology Standards: *SCSI Architecture Model - 5.* Revision 05. 19.05.2010

[Int10c] InterNational Committee for Information Technology Standards: *SCSI Primary Commands -4*. Revision 27. 11.10.2010

[Int17] Intel Corporation ONFI Workgroup: *Open NAND Flash Interface*. 4.1. 12.12.2017

[J. 09] J. Boukhobza, K. R.: A sequential workload performance study of embedded NAND flash memories. In: *Proceedings of the 2009 International Conference on Electrical Engineering, ICEE, Boumerdes, Algerien* (2009)

[JED] JEDEC Solid State Technology Association: *Universal Flash Storage (UFS) 2.1*

[JED15] JEDEC Solid State Technology Association: *Embedded Multi-Media Card (eMMC) Electrical Standard (5.1)*. 01.02.2015

[Kha10] Khalifa Rouis: Performance Evaluation Study of NAND Flash memory-based Storage Systems. In: *Universite de Bretagne Occidentale* (2010)

[Kim] Kim, Eden: SSD Performance - A Primer: An Introduction to Solid State Drive Performance, Evaluation and Test: Storage Networking Industry Association (SNIA). 2013

[KRKC11] Kwon, Se J. ; Ranjitkar, Arun ; Ko, Young-Bae ; Chung, Tae-Sun: FTL algorithms for NAND-type flash memories. In: *Design Automation for Embedded Systems* 15 (2011), Nr. 3-4, S. 191–224. http://dx.doi.org/10.1007/s10617-011-9071-9. – DOI 10.1007/s10617–011–9071–9. – ISSN 0929–5585

[L.] L. Bouganim, B. Jonsson, P. Bonnet: uFLIP: Understanding flash IO patterns. In: *Proceedings of the 4th Biennial Conference on Innovative Data Systems Research (CIDR09)*

[LKB+17] Lee, Eunji ; Kim, Julie ; Bahn, Hyokyung ; Lee, Sunjin ; Noh, Sam H.: Reducing Write Amplification of Flash Storage through Cooperative Data Management with NVM. In: *ACM Transactions on Storage* 13 (2017), Nr. 2, S. 1–13. http://dx.doi.org/10.1145/3060146. – DOI 10.1145/3060146. – ISSN 15533077

[MFL14] Ma, Dongzhe ; Feng, Jianhua ; Li, Guoliang: A survey of address translation technologies for flash memories. In: *ACM Computing Surveys* 46 (2014), Nr. 3, S. 1–39. http://dx.doi.org/10.1145/2512961. – DOI 10.1145/2512961. – ISSN 03600300

[Mic16] Micheloni, Rino: *3D Flash Memories*. 2016 http://dx.doi.org/10.1007/978-94-017-7512-0. – ISBN 9789401775106

[MSCT14] Meena, Jagan S. ; Sze, Simon M. ; Chand, Umesh ; Tseng, Tseung-Yuen: Overview of emerging nonvolatile memory technologies. In: *Nanoscale research letters* 9 (2014), Nr. 1, S. 526. http://dx.doi.org/10.1186/1556-276X-9-526. – DOI 10.1186/1556–276X–9–526. – ISSN 1931–7573

[OBO+13] Ouarnoughi, Hamza ; Boukhobza, Jalil ; Olivier, Pierre ; Plassart, Loic ; Bellatreche, Ladjel: Performance analysis and modeling of SQLite embedded databases on flash file systems. In: *Design Automation for Embedded Systems* 17 (2013), Nr. 3-4, S. 507–542. http://dx.doi.org/10.1007/s10617-014-9149-2. – DOI 10.1007/s10617–014–9149–2. – ISSN 0929–5585

[OSSP08] O'Brien, Kyle ; Salyers, David C. ; Striegel, Aaron D. ; Poellabauer, Christian: Power and performance characteristics of USB flash drives. In: *2008 IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks*. Piscataway : I E E E, Jan. 2008. – ISBN 978–1–4244–2099–5, S. 1–4

[Pri14] Prince, Betty: *Vertical 3D memory technologies*. 2014 http://onlinelibrary.wiley.com/book/10.1002/9781118760475. – ISBN 1118760514

[Sam] Samsung: NAND-Grundlagen. http://www.samsung.com/at/business-images/resource/white-paper/2013/11/samsung_nand_basics_whitepaper-0.pdf

[Sto15] Storage Networking Industry Association: *Solid State Storage (SSS) Performance Test Specification (PTS) Client*. https://www.snia.org/tech_activities/standards/curr_standards/pts. Version: 1.2, 29.05.2015

[SZ16] Schäuffele, Jörg ; Zurawka, Thomas: *Automotive Software Engineering*. Wiesbaden : Springer Fachmedien Wiesbaden, 2016. http://dx.doi.org/10.1007/978-3-658-11815-0. http://dx.doi.org/10.1007/978-3-658-11815-0. – ISBN 978–3–658–11814–3