# Entwurf, modellgestützte Exploration und Optimierung einer heterogenen FPGA-basierten Architektur zur Bildmerkmalsextraktion

Der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des akademischen Grades
Doktor-Ingenieur
(abgekürzt: Dr.-Ing.)
genehmigte Dissertation

von Herrn
Dipl.-Ing. Julian Hartig

geboren am 26.06.1989
in Hannover

2021

| | |
|---|---|
| 1. Referent: | Prof. Dr.-Ing. Holger Blume |
| 2. Referent: | Prof. Dr.-Ing. Diana Göhringer |

Tag der Promotion: 25. Juni 2021

*Die Entstehung dieser Arbeit*
*wurde in Teilen gefördert durch ein*
*Stipendium der Bosch-Forschungsstiftung.*

# Kurzfassung

Die Architekturoptimierung in einem Entwurfsraum mit vielen Parametern stellt den Designer eines eingebetteten Systems immer wieder vor beträchtliche Herausforderungen. Ein systematisches Vorgehen während der Entwurfsraumexploration ist von essentieller Bedeutung und Gegenstand der Forschung auf diesem Gebiet. Insbesondere die bestehende Heterogenität der Systeme mit unterschiedlichen Architekturblöcken, wechselnden und anspruchsvollen Anwendungen sowie dynamischen Performance-Anforderungen erhöhen die Designkomplexität, was das Erreichen einer optimalen Systemrealisierung erschwert. Aufgrund der Größe des sich ergebenden Entwurfsraums lässt sich in der Realität nicht jede Parameterkombination tatsächlich implementieren. Das realistische Durchsuchen des Architekturentwurfsraums nach optimalen Designvarianten wird nur durch Abstraktion möglich, d. h. einer Systembetrachtung auf höheren Entwurfsebenen mithilfe geeigneter Modellbeschreibungen. Durch eine parametrisierte Modellierung eines Architekturblocks lässt sich auch ohne direkte, prototypische Realisierung und für einen bestimmten Parametersatz dieses Blocks eine Bewertung vornehmen. Derartige Methoden, die sich eine Architekturmodellierung zu Optimierungszwecken zu Nutze machen, haben für die Praxis höchste Relevanz.

Im Rahmen dieser Arbeit wurde daher eine Methodik zur modellgestützten Exploration und Optimierung einer heterogenen FPGA-Architektur entwickelt und demonstriert. Die betrachtete Anwendung in diesem Kontext bildet die Bildverarbeitung, daher besteht die Architektur aus einem VLIW-Prozessor mit angekoppeltem dediziertem Beschleuniger, was einen typischen Aufbau für Algorithmen aus diesem Feld darstellt. Der verwendete Algorithmus Scale-Invariant Feature Transform (SIFT) ist dabei eine klassische, sehr komplexe Variante aus dem Feld der Bildmerkmalsextraktion mit hoher algorithmischer Qualität bei gleichzeitigem großem Rechenleistungs- und Speicherbedarf.

Die Exploration des Architekturentwurfsraums erfolgt anhand einer Kombination aus prototypischen Implementierungen und Auswertungen von analytischen Modellgleichungen. Neben einem architekturbasierten Laufzeitmodell wurde ein regressionsbasiertes Verlustleistungsmodell erarbeitet. Durch die modellgestützte Auswertung von Designvarianten können Teile des Entwurfsraums detaillierter betrachtet werden, als dies rein prototypisch oder simulativ möglich wäre. Mit der Architekturoptimierung können der Energiebedarf pro Bild und die dabei entstehende Verlustleistung bei gleichzeitig steigendem Durchsatz und moderatem Ressourcenzuwachs gesenkt werden. Zudem wird durch dynamische Anpassung der Taktfrequenzen des Beschleunigers und des VLIW-Prozessors auf Basis von durch die Modelle bestimmten Parametersätzen weiteres Optimierungspotential aufgezeigt.

Es kann in dieser Arbeit gezeigt werden, dass das demonstrierte, methodische Vorgehen einen strukturierten Weg zur Exploration des Entwurfsraums einer Architektur und dessen Optimierung bildet. Der entwickelte Modellierungsansatz bietet dabei eine konzeptionelle Grundlage für weitere Anwendungen und Architekturen.

*Schlagworte — Bildmerkmalsextraktion, Scale-Invariant Feature Transform, Architekturen zur Bildverarbeitung, modellgestützte Optimierung, Entwurfsraumexploration*

# Abstract

The architecture optimization in a design space with many parameters keeps issuing considerable challenges to the designer of an embedded system. A systematic approach during design space exploration is essential and an object of research of this field. Especially, the existing heterogeneity of the systems with different architectural blocks, variable and challenging applications as well as dynamic performance requirements increase design complexity, which complicates achieving an optimal system realization. Due to the size of the arising design space, not every combination of design parameters can be implemented in practice. The realistic searching of the architectural design space for optimal design variants becomes possible only by abstraction, i.e., a system view on higher design levels with the help of appropriate model descriptions. By parameterized modeling of an architectural block, an evaluation of this block for a specific set of parameters can be done without immediate, prototypical realization. Such methods, which use architecture models for optimization purposes, are highly relevant in practice.

Therefore, in this thesis, a methodology for model-supported exploration and optimization of a heterogeneous FPGA architecture has been developed and demonstrated. The examined application in this context is image processing. Hence, the architecture consists of a VLIW processor combined with a dedicated accelerator, which is a typical setup for algorithms of this kind. The used Scale-Invariant Feature Transform (SIFT) algorithm is a classic, very complex version from the area of image feature extraction with high algorithmic quality at high computational and memory demands.

The exploration of the architectural design space is done using a combination of prototypical implementations and evaluation of analytical model equations. Next to an architecture-based runtime model, a regression-based power model has been created. Through the model-supported evaluation of design variants, it is possible to explore parts of the design space in more detail than it would be possible by prototypes or simulation only. By architecture optimization, the energy per frame and the arising power consumption can be decreased at enhanced throughput and moderate hardware resource increase. In addition, by dynamic adaptation of the clock frequencies of the VLIW processor and the accelerator based on model-generated parameter sets, further optimization potential can be revealed.

In this work, it can be shown, that the presented, methodical approach forms a structured way for exploration of the design space of an architecture and its optimization. The developed modeling approach offers a conceptual base for further applications and architectures.

*Keywords — feature extraction, scale-invariant feature transform, image processing architectures, model-supported optimization, design space exploration*

# Inhaltsverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| **ASIC** | Application-Specific Integrated Circuit |
| **ASIP** | Application-Specific Instruction-set Processor |
| **BRIEF** | Binary Robust Independent Elementary Features |
| **BRISK** | Binary Robust Invariant Scalable Keypoints |
| **CMOS** | Complementary Metal Oxide Semiconductor |
| **CORDIC** | Coordinate Rotation Digital Computer |
| **CPU** | Central Processing Unit |
| **DMA** | Direct Memory Access |
| **DoG** | Difference-of-Gaussian |
| **DSP** | Digital Signal Processor |
| **FAST** | Features from Accelerated Segment Test |
| **FIFO** | First-In First-Out Speicher |
| **FIR** | Finite Impulse Response |
| **FPGA** | Field Programmable Gate Array |
| **fps** | Frames per Second |
| **FSM** | Finite State Machine |
| **FU** | Functional Unit |
| **GPU** | Graphics Processing Unit |
| **GUI** | Graphical User Interface |
| **LoG** | Laplacian-of-Gaussian |
| **MMCM** | Mixed-Mode Clock Manager |

| | |
|---|---|
| **NOP** | No Operation |
| **ORB** | Oriented FAST and Rotated BRIEF |
| **PLL** | Phase-Locked Loop |
| **RANSAC** | RAndom SAmple Consensus |
| **RAM** | Random Access Memory |
| **SDE** | SIFT Detection Engine |
| **SIFT** | Scale-Invariant Feature Transform |
| **SLM** | Straight Line Microcode |
| **SoC** | System on Chip |
| **SRAM** | Synchronous Random Access Memory |
| **SURF** | Speeded Up Robust Features |
| **TLM** | Transaction Level Modelling |
| **VLIW** | Very Long Instruction Word |
| **VLSI** | Very Large Scale Integration |

# 1 Einleitung

Die Entwicklung eines modernen eingebetteten Systems stellt eine beträchtliche Herausforderung dar. Die Systemspezifikation beinhaltet oft mehrere miteinander in Konflikt stehende Anforderungen. Ein typisches Designziel ist die Maximierung der Rechenleistung bei gleichzeitiger Minimierung der Verlustleistung und der Kosten. Dabei lässt sich eine erhöhte Rechenleistung je nach Anwendung meist nur durch erhöhten Hardwareaufwand und somit steigende Kosten erreichen. Zudem erzeugt ein auf Rechenleistung optimiertes System eine erhöhte Verlustleistung, sodass eine leistungsfähigere Kühlung nötig wird, da die entstehende Hitze die Systemlebensdauer ansonsten verringern würde. In batteriebetriebenen Systemen kommt zusätzlich dem Energiebedarf zur Erfüllung einer bestimmten Aufgabe besondere Bedeutung zu, da dadurch die Zeit bis zum Nachladen der Batterie bestimmt wird.

Dem Designer stehen zur Systemrealisierung und Erreichen der Spezifikation eine Vielzahl an möglichen *Implementierungsalternativen* zur Verfügung, wie der verwendeten Computerarchitektur und der Halbleitertechnologie, in der der IC hergestellt wird. Da zukünftige Anwendungen, die auf einem eingebetteten System implementiert werden sollen, vielschichtig sind, in ihrer Komplexität und Rechenleistungsanforderung stetig steigen und zudem zum Zeitpunkt der Hardwareentwicklung oft noch nicht vollständig bekannt sind, beinhalten die meisten Systeme *heterogene* Hardwarestrukturen, d. h. verschiedene Architekturblöcke. Neben unterschiedlichen Formen von programmierbaren Prozessoren werden dedizierte Schaltungsteile für spezielle Aufgaben implementiert. Diese sind besonders durchsatzstark und energieeffizient, allerdings deutlich weniger flexibel als ein Prozessor, dessen Software zu einem späteren Zeitpunkt noch geändert werden kann. Darüber hinaus spielen Aspekte wie Time-to-Market und Testbarkeit bei der Auswahl der richtigen Implementierungsalternative eine Rolle.

Die genannten Kriterien spannen den sogenannten *Entwurfsraum* auf, in dem sich der Designer beim Systementwurf bewegt. Die Navigation im multidimensionalen Entwurfsraum wird als *Entwurfsraumexploration* bezeichnet und ist schematisch in Abb. 1.1 dargestellt. Der Designer muss verschiedene Aspekte des Entwurfs gegeneinander abtauschen und aus den sich ergebenden Implementierungsalternativen eine Auswahl treffen. Dabei ist insbesondere zu Beginn des Entwurfsprozesses der Entwurfsraum noch besonders groß. Ein systematisches Vorgehen während des Systementwurfs ist von essentieller Bedeutung.

In der Praxis bestimmt die Erfahrung des Designers oft schon zu einem frühen Entwurfszeitpunkt in groben Zügen die Architekturblöcke, die in einem eingebetteten System zum Erfüllen der Spezifikation notwendig sind, sodass eine architekturübergreifende

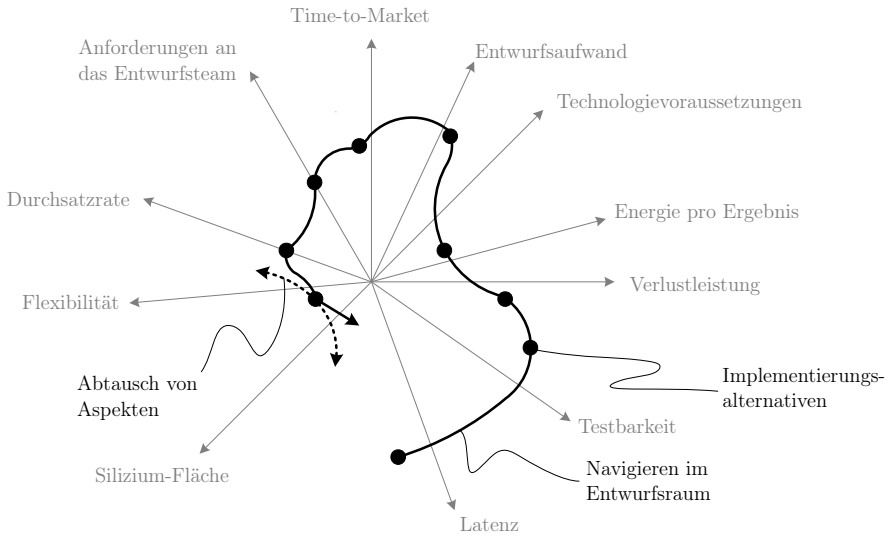



**Abbildung 1.1:** Entwurfsraumexploration: Navigieren im multidimensionalen Entwurfsraum (nach [Nol04])

Entwurfsraumexploration nicht durchgeführt wird. Doch selbst nach dieser Verkleinerung des Entwurfsraums besitzt ein System noch viele Parameter und die optimalen Implementierungsalternativen sind unbekannt.

Aufgrund der Größe eines Entwurfsraums lässt sich in der Realität nicht jede Parameterkombination tatsächlich implementieren. Ein sinnvolles Durchsuchen des Entwurfsraums wird nur durch Abstraktion möglich, d. h. durch das Betrachten des Systems auf höheren Entwurfsebenen mithilfe geeigneter Modellbeschreibungen. Durch eine parametrisierte Modellierung eines Architekturblocks für eine bestimmte Anwendung lässt sich auch ohne direkte Realisierung und für einen bestimmten Parametersatz dieses Blocks eine Bewertung vornehmen.

Das übergeordnete Ziel bei der Exploration des Entwurfsraums einer Architektur sollte die Quantifizierung und volle Ausnutzung des *intra-Architektur-Optimierungspotentials* sein. Dieses liegt nach [Blu08] bezogen auf das ATE-Produkt (Produkt aus erforderlicher Siliziumfläche, Taktperiode pro berechnetem Ergebnis und Energie pro Ergebnis) etwa innerhalb einer Größenordnung. Ein modellgestütztes Vorgehen kann die Architekturoptimierung beschleunigen und unterstützen.

Die beschriebene abstrakte Methodik kann dabei nur im Kontext einer anspruchsvollen realen Anwendung sinnvoll untersucht werden.

### Motivation der Anwendung

Ein großer aktueller Technologietreiber ist das Forschungsfeld des autonomen Fahrens [Jan17]. Zentrale Sensoren, um das Umfeld des Fahrzeugs zu überwachen, sind u.

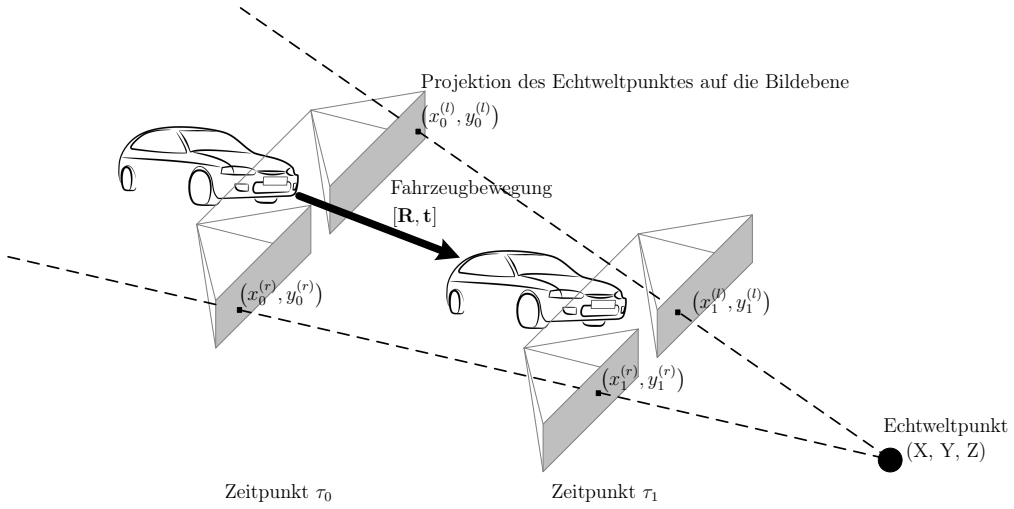



**Abbildung 1.2:** Merkmalsbasierte Eigenbewegungsschätzung eines Fahrzeugs

a. eine Vielzahl von Kameras. Mit ihnen können Kernfunktionen wie z. B. eine virtuell errechnete Ansicht von oben auf das Fahrzeug (engl. *Topview*), eine Schätzung der Bewegung des Fahrzeugs (engl. *Egomotion*) oder Objekterkennung realisiert werden. Klassische algorithmische Methoden basieren dabei oft auf der Extraktion von Bildmerkmalen [Sze11], d. h. markanten Punkten im Bild, die innerhalb einer Bildsequenz mit hoher Zuverlässigkeit wiedergefunden werden können. Dies ist in Abb. 1.2 am Beispiel der Eigenbewegungsschätzung illustriert. In einem Fahrzeug ist im Bereich der Frontscheibe eine Stereokamera platziert. Ein Echtweltpunkt $(X, Y, Z)$ projiziert sich zum Zeitpunkt $\tau_0$ entlang eines Strahls auf die beiden Bildebenen der Kamera. Durch Algorithmen zur Bildmerkmalsextraktion lassen sich die beiden Abbildungen des Echtweltpunktes an den Pixelpositionen $\left(x_0^{(l)}, y_0^{(l)}\right)$ und $\left(x_0^{(r)}, y_0^{(r)}\right)$ wiederfinden und einander zuordnen. Erfolgt diese Zuordnung zusätzlich zu einem zweiten Zeitpunkt $\tau_1$ und für mehrere unterschiedliche Echtweltpunkte, so können aus der Verschiebung der Pixelpositionen der Merkmale die rotatorischen und translatorischen Bewegungsparameter $[R, t]$ des Fahrzeugs geschätzt werden [Gei11].

Den ersten Teilschritt der algorithmischen Kette bildet dabei die Bildmerkmalsextraktion. Da diese auf Pixelebene arbeitet, muss die zugrunde liegende Computerarchitektur in der Lage sein, die große anfallende Rate an Eingangsdaten zu verarbeiten. Dazu sind eine hohe Rechenleistung und parallele Datenverarbeitung erforderlich. Zudem unterliegen eingebettete Systeme im Fahrzeug einem geringen Verlustleistungsbudget von wenigen Watt, sodass FPGAs oder ASICs als Hardwareplattformen trotz ihres komplexeren Entwurfsprozesses im Vergleich zur Verwendung eines Standardprozessors notwendig sind.

## 1.1 Zielsetzung der Arbeit

Die Ziele der vorliegenden Arbeit lassen sich in die zwei Teilaspekte Methodik und Architektur unterteilen.

- Bei der Optimierung einer Architektur für eine Anwendung steht der Designer ab einem gewissen Zeitpunkt aufgrund der Vielzahl an Designparametern vor einem großen Entwurfsraum. Im Rahmen dieser Arbeit soll eine modellgestützte Methodik untersucht werden, die die Exploration des Entwurfsraums mit dem Ziel einer Architekturoptimierung vereinfachen kann. Neben einer Nutzung eines Architekturmodells zur Designzeit soll darüber hinaus betrachtet werden, inwieweit ein Architekturmodell auch zur Laufzeit durch die Realisierung eines adaptiven Systemverhaltens zur Optimierung beitragen kann.
- Als Grundlage für die Methodik soll in dieser Arbeit eine FPGA-Architektur für eine anspruchsvolle Anwendung aus dem Bereich der Bildverarbeitung, der Bildmerkmalsextraktion, entworfen werden. Aufgrund der Komplexität des Algorithmus muss eine Partitionierung auf heterogene Architekturblöcke vorgenommen werden. Es soll ein hoher Datendurchsatz erreicht werden können. Neben einer parametrisierten Implementierung zum späteren Abtausch verschiedener Optimierungskriterien soll eine möglichst präzise Architekturmodellierung erarbeitet werden.

## 1.2 Aufbau der Arbeit

Kapitel 2 geht auf methodische Grundlagen der Entwurfsraumexploration ein und stellt den Stand der Forschung auf dem Gebiet der Bildmerkmalsextraktion dar. Dabei wird auch der in dieser Arbeit verwendete Bildmerkmalsalgorithmus im Detail präsentiert. Kapitel 3 stellt in der Literatur bestehende Architekturen zur Bildmerkmalsextraktion dar und beschreibt die im Rahmen dieser Arbeit entworfene, heterogene FPGA-Architektur, welche einen dedizierten Beschleuniger und einen VLIW-Prozessor beinhaltet, der in seiner Konfiguration auf die Anwendung angepasst wird. Ein Modellierungsansatz für das Gesamtsystem hinsichtlich Laufzeit und Verlustleistung wird in Kapitel 4 erarbeitet. Anschließend erfolgt in Kapitel 5 auf Basis der abgeleiteten Modelle die modellgestützte Optimierung und Exploration des Entwurfsraums der FPGA-Architektur. Zudem wird der Einfluss einer dynamischen Anpassung von Designparametern zur Laufzeit des Systems untersucht. Kapitel 6 fasst die Arbeit zusammen.

# 2 Grundlagen und Stand der Forschung

## 2.1 Entwurfsraumexploration

Moderne eingebettete Systeme müssen typischerweise eine Vielzahl von strikten, oft miteinander in Konflikt stehenden Anforderungen erfüllen [Pim17]. Dem Designer eröffnet sich dabei zunächst ein großer *Entwurfsraum*. Durch Designentscheidungen muss er im Entwurfsraum navigieren und dabei verschiedene Parameter gegeneinander abwägen, um eine optimale Lösung zu finden. Unter dem Namen *Entwurfsraumexploration* wurden Methoden entwickelt, um die Komplexität des Entwurfsprozesses beherrschbar zu machen und den Designprozess bereits in einem frühen Stadium zu unterstützen [Gri04]. Dazu sollen in den folgenden Abschnitten einige ausgewählte Grundlagen und Begriffsdefinitionen (Abschnitt 2.1.1) und ein kurzer Überblick über die Methodik (Abschnitt 2.1.2) gegeben werden.

### 2.1.1 Definitionen

Zur Beschreibung des Entwurfsraums eines Systems unterscheidet [Gri04] zwischen *Problemraum* und *Lösungsraum*. Der Problemraum wird durch die Systemeigenschaften aufgespannt. Im Kontext einer Speicherarchitektur können dies die Anzahl an Cache-Leveln, die Größe jedes Levels oder der Caching-Algorithmus sein. Diese Eigenschaften werden im Folgenden *Designparameter* genannt. Die Komplexität des Problemraums wächst exponentiell mit der Anzahl an Designparametern [Pal13]. Eine Kombination aus Designparametern heißt *Designvariante*.

Dem Problemraum steht der Lösungsraum gegenüber. Dieser ergibt sich aus den Systemanforderungen bzw. Designkriterien der Entwurfsraumexploration, welche nach [Pal13] im Folgenden als *Zielindikatoren* bezeichnet werden. Ein primärer Zielindikator ist im Kontext des Designs von integrierten Schaltungen z. B. die Performance, die Verlustleistung, die Chipfläche oder die Kosten. Ein sekundärer Zielindikator hingegen bezieht sich nur auf einen Teil des Designs oder gibt nur unterstützende Informationen, wie z. B. Busauslastung oder Profiling-Ergebnisse. Eine Exploration des Entwurfsraums wird entweder durch den Problemraum (Abfahren möglicher Designvarianten und Auswahl des Optimums) oder durch den Lösungsraum (Optimierung der Zielindikatoren durch gezieltes Anpassen der Designparameter) motiviert [Gri04].

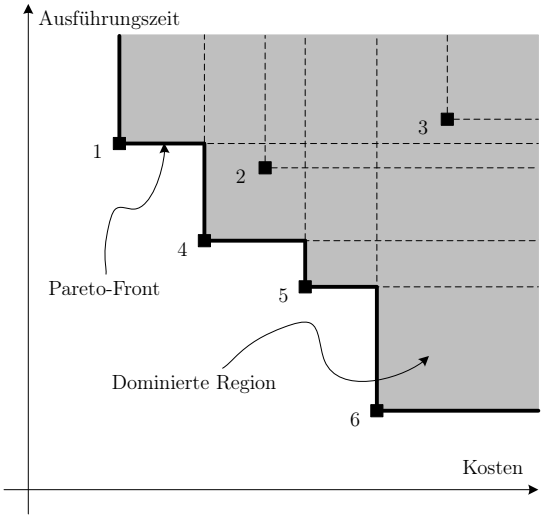



**Abbildung 2.1:** Zweidimensionaler Entwurfsraum mit Pareto-optimalen Designvarianten 1, 4, 5 und 6 (Pareto-Menge) und Pareto-Front (nach [Gri04])

### Dominanz und Pareto-Optimalität

Bei der Optimierung nach nur einem Kriterium kann durch Abfahren der Designparameter, sowie Auswertung und Vergleich des Zielindikators die optimale Designvariante gefunden werden. Meist werden in einer Entwurfsraumexploration jedoch mehrere Zielindikatoren gleichzeitig betrachtet. Ein typisches Optimierungsszenario ist eine Minimierung der Verlustleistung und der Kosten bei gleichzeitiger Maximierung der Performance. Dabei stehen die Zielindikatoren in enger, einander entgegenstehender Wechselwirkung zueinander [Gri04], wodurch nicht alle gleichzeitig optimiert werden können.

Zur Vergleichbarkeit der Ergebnisse einer multikriteriellen Optimierung wird der Begriff der *Pareto-Optimalität* benötigt, welcher hier in Abb. 2.1 anhand eines Beispiels nach [Gri04] erläutert werden soll. Die Abbildung zeigt einen zweidimensionalen Entwurfsraum mit der Ausführungszeit über den Kosten. Eine optimale Designvariante minimiert beide dieser Zielindikatoren. Für jede der exemplarischen Varianten 1-6 ist die dominierte Region eingezeichnet. Eine Designvariante *dominiert* eine andere, wenn sie in mindestens einem Zielindikator besser und in allen anderen mindestens gleich ist [Gri04]. Eine Lösung wird als *Pareto-optimal* bezeichnet, wenn sie von keiner anderen Lösung dominiert wird. Nicht-dominierte Lösungen bilden die sog. *Pareto-Menge* bzw. graphisch eine *Pareto-Front*, auf der keine der Lösungen von einer anderen Lösung dominiert wird. In anderen Worten sind diejenigen Designvarianten Pareto-optimal, die in einem Zielindikator besser und in allen anderen höchstens gleich oder schlechter sind. Im Beispiel hier bilden die Varianten $1, 4, 5, 6$ die Pareto-Menge. Es liegt im Ermessen des Designers, welche Designvariante den besten Interessenausgleich darstellt [Pim17].

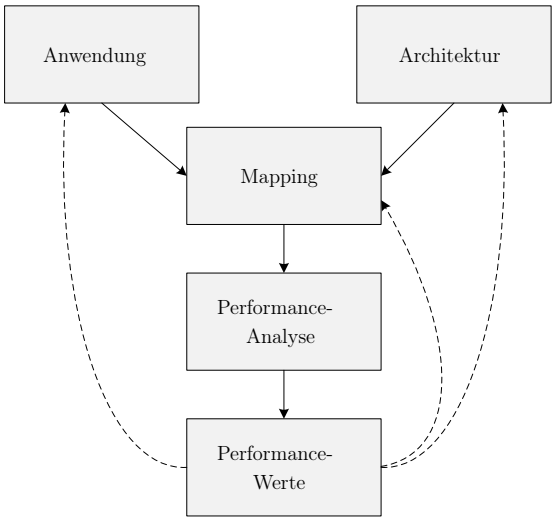



**Abbildung 2.2:** Y-Diagramm zur Beschreibung des typischen Ablaufs während der quantitativen Entwurfsraumexploration nach [Kie97; Keu00]

## 2.1.2 Methodik

### Y-Diagramm

Der systematische Ablauf bei der Entwurfsraumexploration und die Suche nach Pareto-optimalen Designvarianten folgt in der Literatur oft dem Y-Diagramm nach [Kie97; Keu00] (siehe Abb. 2.2). Nach diesem Schema werden die Beschreibungen der Anwendung und der Architektur zunächst getrennt gehalten. Die Abbildung der Anwendung auf die Architektur (Implementierung) erfolgt in einem Mapping-Schritt und wird anschließend während der Performance-Analyse ausgewertet. Diese Auswertung kann abhängig von den zu untersuchenden Zielindikatoren sehr zeitaufwändig sein, da Synthese- oder Compiler-Schritte, sowie (simulierte) Testläufe erforderlich sein können. Die so erhaltenen Performance-Werte werden mit den Randbedingungen und Spezifikationen verglichen, die zuvor an die Anwendung und die Architektur gestellt wurden. Üblicherweise folgen weitere Iterationen mit modifizierter Applikationsbeschreibung, veränderten Architekturblöcken oder anderen Parametern, die den Mapping-Schritt betreffen (z. B. Synthese- oder Compiler-Parameter). [Gri04; Blu08]

Das Vorgehen nach dem Y-Diagramm zur Entwurfsraumexploration und der Suchen nach Pareto-optimalen Designvarianten umfasst zwei grundsätzliche Aspekte [Gri04; Pim17]:

- die Auswertung einer einzelnen Designvariante (Performance-Analyse)
- die Suchstrategie zur Abdeckung des Entwurfsraums (weitere Iterationen)

Auf diese beiden Aspekte wird im Folgenden genauer eingegangen.

### Auswertung einer einzelnen Designvariante

Die auf den Mapping-Schritt folgende Auswertung einer einzelnen Designvariante steht im Kern der Methode. Dieser Schritt ist in jeder Iteration nötig und muss somit unter Umständen sehr oft durchgeführt werden. Realisierungen dieses Schrittes bilden eine Kompromissstellung aus Auswertungsgenauigkeit, -geschwindigkeit und -aufwand [Pim17]. Dabei spielt der Grad der Abstraktion der Systemrepräsentation, die zur Auswertung herangezogen wird, eine große Rolle.

Die höchste Genauigkeit bei gleichzeitig hoher Auswertungsgeschwindigkeit kann durch eine direkte Messung an einem *Systemprototypen* erzielt werden. Der Aufwand zur Realisierung ist dabei meistens zu hoch, da die Komplexität des Entwurfsraums eine Vielzahl von Prototypen und im Fall von VLSI-Architekturen zeitaufwendige Synthesen und kostspielige Chipdesigns erfordern würde. FPGA-Architekturen haben hier insbesondere als spätere Zielplattform aber den Vorteil, dass die Schaltung direkt auf das FPGA abgebildet wird (rapid prototyping) und ein Chipdesign nicht notwendig ist.

Eine Abstraktionsebene höher bieten zyklengenaue *Simulationen* bzw. Simulationen auf Systemebene den Vorteil des geringeren Aufwands. Allerdings ist die Genauigkeit abhängig vom Simulationsverfahren nicht so hoch wie bei einer realen Messung. Zudem ist die Simulationszeit mitunter sehr lang, insbesondere bei hohem Detailgrad der Simulation. Eine Simulation setzt eine ausführbare Beschreibung der Anwendung und der Architektur voraus, welche meist durch die eigentliche Entwurfseingabe zur Verfügung steht. Für die Anwendung sind Beschreibungen in einer Hochsprache, graphenbasierte Beschreibungen oder eine Transaction Level Modeling (TLM) [Cai03] üblich. Architekturen können mit Hardwarebeschreibungssprachen wie VHDL, Verilog und SystemC oder auch Architektur-Templates, die an die Anwendung angepasst werden, beschrieben werden.

Eine dritte Variante der Auswertung stellen abstrakte, *analytische Modelle* des Systems dar [Eec10]. Diese sind in der Lage, Performance-Werte auf hoher Abstraktionsebene abzuschätzen. Die Auswertungsgeschwindigkeit ist dabei sehr hoch. Somit bietet dieser Ansatz gerade für große Entwurfsräume eine sinnvolle Alternative gegenüber Simulationen, da der Entwurfsraums schnell durchsucht werden kann, um interessierende Regionen zu identifizieren, welche danach mithilfe von Simulationen im Detail erkundet werden können. Jedoch besteht der initiale Aufwand der Erstellung dieser Modelle. Zudem können typischerweise nicht alle Eigenschaften eines Systems mit hoher Genauigkeit abgebildet werden. Besteht die genaue Kenntnis der Architektur, so kann eine analytische Modellierung auf Basis der genauen, architekturbasierten Zusammenhänge als sog. *whitebox*-Modell erfolgen. Eine andere Möglichkeit bieten empirische, auf Regression basierende Modelle, sog. *blackbox*-Modelle. Diese benötigen zur Modellerstellung allerdings Trainingsdaten und geben typischerweise weniger Einsicht in das System als architekturbasierte Modelle [Pim17].

Eine Kombination aus simulativen und analytischen Modellen ist ebenfalls üblich. Eine ausführliche Darstellung der Modellierungsansätze von Anwendung und Architektur auf verschiedenen Abstraktionsebenen wird in [Gri04] gegeben.

### Durchsuchen des Entwurfsraums

Das Durchsuchen des Entwurfsraums ist ein multikriterieller Optimierungsprozess, welcher anhand von Zielindikatoren im Ergebnis eine Liste an Pareto-optimalen Designvarianten (Pareto-Menge) liefert. Suchmethoden lassen sich in Aufwand zur Initialisierung, in Konvergenzgeschwindigkeit hin zu einem optimalen Ergebnis sowie in Konfidenz bzw. Vertrauen in das gefundene Ergebnis unterscheiden [Pim17].

Große Entwurfsräume lassen sich niemals vollständig durchsuchen, da die Anzahl der Kombinationsmöglichkeit der Designparameter sehr hoch und eine Auswertung der Varianten in realistischer Zeit nicht durchführbar ist. Somit gestaltet sich auch eine *exakte* Bestimmung der Pareto-Front als schwierig.

Abhilfe schaffen hier heuristische Methoden, welche weniger zeit- und rechenintensiv sind, allerdings nicht gesichert auch immer ein globales Optimum finden [Gri04]. Das Ziel einer heuristischen Durchsuchung des Entwurfsraums ist es, eine möglichst hohe Abdeckung (engl. *coverage*) zu erhalten. Eine *vollständige Suche* durch Abfahren der Designparameter ist in der Praxis aufgrund der Größe des Entwurfsraums nur in limitiertem Maße und auf einzelne Parameter beschränkt möglich. Eine zweite Variante bildet eine *zufallsgesteuerte Suche*, in der die Designvarianten durch zufälliges Abtasten der Parameter ermittelt werden. Der Vorteil liegt hier in der Objektivität der Methode, da der Suchprozess nicht vom Designer beeinflusst wird.

Im Gegensatz dazu steht die *geführte Suche*, die Vorwissen des Designers über die Struktur des Entwurfsraums in das Suchverhalten mit einfließen lässt. Dieses Vorwissen kann auch während des Suchprozesses aufgrund der generierten Ergebnisse aktualisiert werden.

Für den Erfolg der Methodik auch bei großen Entwurfsräumen kommt der Eingrenzung des Entwurfsraums (engl. *pruning*) eine besondere Bedeutung zu. Grundsätzlich kann es sinnvoll sein, die Designparameter zu *quantisieren*, was die Anzahl der möglichen Designvarianten reduziert. Dies ist insbesondere in vollständig zu durchsuchenden Teilbereichen unerlässlich. Darüber hinaus bietet jedes komplexe System die Möglichkeit der *Hierarchisierung*. Die Exploration kann zunächst auf Systemebene mit abstrakten Modellen durchgeführt werden, bevor Teilbereiche mit detaillierteren Verfahren betrachtet werden. Auch die *Systemteilung* in unabhängige Bereiche reduziert die Komplexität. Außerdem ist eine *Beschränkung* des Entwurfsraums anhand von Randbedingungen und Spezifikationen in einem initialen Schritt von Vorteil, sowie das Identifizieren von Corner Cases, um den Aufwand der Suche auf die für die spätere Systemrealisierung relevanten Bereiche zu konzentrieren.

Diese kurze Aufzählung verschiedener Ansätze soll einen groben Überblick der Methodik zur Durchsuchung eines Entwurfsraum geben, um das in dieser Arbeit verwendete Vorgehen einzuordnen. Für eine vollständige, tiefergehende Betrachtung wird an dieser Stelle auf die Überblicksliteratur [Gri04; Pim17] verwiesen.

## 2.2 Bildmerkmalsextraktion

Das menschliche Auge und Gehirn sind außerordentlich gut darin, aus Bildern Informationen zu ziehen. Computersystemen fällt diese Aufgabe ungleich schwerer. Ein Bestreben von Bildverarbeitung und Computer Vision ist es, relevante Merkmale eines Bildes zu extrahieren, was meist sehr aufwendige Algorithmen erfordert. Merkmalsextraktion im Allgemeinen beinhaltet die Erkennung und Isolierung von gewünschten Merkmalen eines Bildes, um daraus dann bedeutsame Informationen zu identifizieren bzw. zu interpretieren [Awa16]. Ein Merkmal ist dabei ein Bildmuster (z. B. Ecken oder Kanten), welches sich markant von seiner unmittelbaren Nachbarschaft unterscheidet und sich dadurch stabil in verschiedenen Bildern wiederfinden lässt [Tuy08]. Dabei stellen folgende Eigenschaften die grundsätzlichen Anforderungen dar, die sich bei der Verwendung von Bildmerkmalsextraktionsalgorithmen ergeben [Awa16]:

- *Robustheit*: Der Algorithmus sollte in der Lage sein, die gleichen Merkmalspositionen stabil zu detektieren, unabhängig von Skalierung, Rotation, Verschiebung, photometrische Verformungen, Kompressionsartefakten und Bildrauschen.
- *Reproduzierbarkeit*: Der Algorithmus sollte in der Lage sein, die gleichen Merkmale einer Szene reproduzierbar zu detektieren, auch unter einer Vielzahl von Sichtbedingungen.
- *Genauigkeit*: Der Algorithmus sollte in der Lage sein, die Merkmale mit hoher Genauigkeit an den gleichen Pixelpositionen zu lokalisieren, insbesondere wenn präzise Punktkorrespondenzen notwendig sind, um z. B. die Epipolargeometrie [Sze11] zwischen Bildern zu schätzen.
- *Allgemeingültigkeit*: Der Algorithmus sollte in der Lage sein, Merkmale zu finden, die in unterschiedlichen Anwendungen verwendet werden können.
- *Effizienz*: Der Algorithmus sollte in der Lage sein, Merkmale in neuen Bildern schnell zu erkennen, um auch in Echtzeitsystemen eingesetzt werden zu können.
- *Anzahl*: Der Algorithmus sollte in der Lage sein, alle oder die meisten der Merkmale in einem Bild zu detektieren.

Bildmerkmalsextraktion besteht aus drei algorithmischen Schritten. Als erstes müssen die Pixelpositionen von Bildmerkmalen detektiert werden (*Detektor*). Danach wird die Bildumgebung um das Merkmal beschrieben (*Deskriptor*). In einem dritten Schritt werden die Merkmalsbeschreibungen (Deskriptoren) eines Bildes den Merkmalsbeschreibungen eines anderen Bildes zugeordnet (engl. *Matching*) und es entsteht eine sogenannte *Punktkorrespondenzliste*.

In Abb. 2.3 wird der grundsätzliche Unterschied zwischen einem Eckendetektor (links) und einem skalenraumbasierten Blob-Detektor (rechts) deutlich. Für einen Blob-Detektor

**Abbildung 2.3:** Bildmerkmalsextraktion: FAST-Eckendetektor nach [Ros10] (links), SIFT-Blob-Detektor mit Skalierung und Orientierung nach [Low04b] (rechts). Beispielbild aus der KITTI Vision Benchmark Suite [Gei13].

(englisch für "Klecks") lassen sich einem Merkmal eine Größe (Skalierung) und eine Richtung (Orientierung) zuordnen. Diese sind in Abb. 2.3 über Länge und Position des eingezeichneten Kreisradius dargestellt. Ecken können zwar mit hoher Effizienz unter geringen Rechenleistungsanforderungen detektiert werden, allerdings ist ihre Stabilität beim Matching in vielen Anwendungen schlechter, da sich zwischen zwei Bildern mit ähnlichem Bildinhalt, aber z. B. starken Änderungen im Blickpunkt mitunter auch die Skalierung signifikant ändert. Neben Skalierungsinvarianz und Invarianz gegenüber Bildrotation ist auch affine Invarianz in vielen Änderungen erforderlich, d. h. Merkmale müssen sich auch unter starken perspektivischen Veränderungen, die sich lokal durch eine affine Transformation des Bildinhalts approximieren lassen, wiederfinden lassen [Mik05c]. Außerdem ist Stabilität gegen Bildrauschen und Beleuchtungsänderungen nötig.

Eine Vielzahl von Anwendungen benötigt die Liste von Punktkorrespondenzen zwischen zwei oder mehreren Bildern. Beim Kalibrieren einer Kamera werden Merkmale, z. B. Ecken, deren Echtweltposition in einem 3D-Koordinatensystem bekannt sind, pixelgenau auf der Bildebene für verschiedene Bilder lokalisiert. Dadurch ist es möglich, mithilfe von numerischen Verfahren die spezifischen intrinsischen und extrinsischen Kameraparameter (u. a. Brennweite, Linsenparameter und für Stereo-Kameras die Basislinie) zu bestimmen [Har03]. Auf ähnliche Weise kann die Bewegung einer Kamera berechnet werden (Eigenbewegungsschätzung). Für eine kalibrierte Stereo-Kamera können aus der Punktkorrespondenzliste die rotatorischen und translatorischen Bewegungskomponenten bestimmt werden, wenn man die Pixelpositionen mehrerer Echtweltpunkte zu zwei unterschiedlichen Zeitpunkten findet [Gei11]. Eine Möglichkeit, um Objekte auf Bildern zu erkennen, ist die Extraktion von Merkmalen und ihr Abgleich mit einer großen Datenbank. Da der Blickwinkel auf ein Objekt dabei beliebig sein kann, ist eine hohe Robustheit des Merkmalsalgorithmus gegenüber affinen Transformationen notwendig [Low04b]. Beim Aufnehmen von z. B. Panoramafotos werden mehrere Bilder aneinandergefügt (engl. image stitching). Da die Einzelbilder dabei von der gleichen Echtweltposition aufge-

nommen werden, wird die perspektivische Abbildung zwischen zwei Bildern durch eine Homographie beschrieben, die mithilfe einer Punktkorrespondenzliste bestimmt werden kann [Sze11].

Auf dem Gebiet der Bildmerkmale wurde insbesondere innerhalb der letzten zwei Jahrzehnte eine große Anzahl verschiedener Verfahren vorgestellt, die für sich selbst genommen ein eigenes, umfassendes Forschungsfeld bilden. Die nachfolgenden Abschnitte erläutern den Stand der Forschung zur Bildmerkmalsextraktion, soweit dies zum Verständnis der Auswahl des Referenzalgorithmus für die FPGA-Architektur und dessen algorithmischer Bewertung notwendig ist. Es werden zunächst grundlegende algorithmische Verfahren der Bildmerkmalsextraktion (Abschnitt 2.2.1) sowie Metriken (Abschnitt 2.2.2) erläutert, um diese Verfahren in ihrer Qualität zu bewerten und miteinander zu vergleichen. Darauf aufbauend wird auf den algorithmischen Entwurfsraum eingegangen und die Auswahl des Referenzalgorithmus begründet (Abschnitt 2.2.3). Dieser wird danach in ausführlicher Form dargestellt (Abschnitt 2.2.4).

### 2.2.1 Verfahren

Dieser Abschnitt soll einen Überblick über die am häufigsten verwendeten Verfahren geben, die auf lokalen Bildmerkmalen beruhen. Auf Regionsdetektoren [Mat04; Mik05c] oder hochmoderne CNN-basierte Verfahren [Fan19] soll hier nicht eingegangen werden, da sie algorithmische Ansätze verfolgen, die für die später vorgestellte FPGA-Architektur keine Relevanz haben.

#### Bildmerkmalsdetektoren

Bildmerkmalsdetektoren werden in der Literatur in die Kategorien Ecken-, Blob- und Bereichsdetektoren abhängig davon eingeteilt, worauf sie innerhalb eines Bildes reagieren. Dabei werden verschiedene Ansätze der Detektion verfolgt. Im Folgenden werden einige ausgewählte Detektoren vorgestellt und ihre Prinzipien kurz erläutert. Darüber hinaus sei auf die Literatur verwiesen [Tuy08; Li08; Dah11; Aan12; Awa16; Fan19].

**Moravec's Detektor** (1977) [Mor77] Der Moravec Detektor ist einer der ersten Algorithmen, der zur Detektion von Ecken vorgestellt wurde. Ausgehend von der Annahme, dass ein Bildbereich mit einer Ecke eine geringe Selbstähnlichkeit aufweist, d. h. sich von seiner Umgebung in mindestens zwei signifikant unterschiedlichen Richtungen unterscheidet, wird die Ähnlichkeit zwischen dem Bereich um jeden Pixel und überlappenden Bereichen in der Nachbarschaft bestimmt. Dies geschieht mithilfe von sum-of-squared-differences (SSD) entlang der acht umliegenden Pfade um einen Pixel, von denen das Minimum als Maß für den jeweilige Pixel herangezogen wird. Wenn der SSD-Wert ein lokales Maximum erreicht, liegt eine Ecke vor. Das Prinzip findet zwar eine hohe Anzahl an Ecken im Bild, allerdings liegen diese auch zu großen Teilen auf Kanten. Merkmale auf Kanten haben eine hohe Mehrdeutigkeit beim späteren Matching (*Apertur-Problem* [Bin09]) und können so für falsche oder unpräzise Ergebnisse sorgen.

**Harris-Detektor** (1988) [Har88] Der Harris-Eckendetektor ist ein kombinierter Ecken- und Kantendetektor. Dabei werden um einen Pixel die Ableitungen in x- und y-Richtung und danach die Autokorrelationsmatrix gebildet, welche Informationen über die Struktur um den zu untersuchenden Pixel herum enthält. Sind die Eigenwerte der Matrix beide groß, wurde eine Ecke detektiert. Bei einem großen und einem kleinen Eigenwert liegt eine Kante vor. Anstatt die Eigenwerte direkt zu bestimmen, was die Auswertung einer Wurzelfunktion erfordern würde, verwenden die Autoren ein kombiniertes Maß, das lediglich das Verhältnis der Eigenwerte zueinander widerspiegelt. Dieses Maß dient zur Kantenunterdrückung und macht die Merkmale so stabiler beim Matching. Die Verwendung der Autokorrelationsmatrix zur Detektion ist immer noch in vielen Anwendungen verbreitet. Die detektierten Ecken sind allerdings nicht skalierungsinvariant.

**SIFT-Detektor** (1999) [Low99; Low04b] Der SIFT-Detektor (Scale-Invariant Feature Transform) basiert auf der Skalenraumtheorie [Koe84; Lin94] und detektiert Blobs ("Klecks-se") im Bild. Im Allgemeinen ist der Informationsgehalt und damit auch die Eindeutigkeit von Bildmerkmalen abhängig von der Merkmalsskalierung. Um zusätzlich zur präzisen Merkmalsposition auch noch diejenige Position innerhalb des Skalenraums zu detektieren, die das Merkmal am besten beschreibt, und damit Merkmale höherer Qualität zu erhalten, wird beim SIFT-Detektor der Skalenraum mithilfe von Difference-of-Gaussians (DoG) aufgebaut, was einer Approximation des Laplacian-of-Gaussian (LoG) Operators zum Aufbau des Skalenraums nach [Lin94] entspricht. Dabei wird eine Bildpyramide mit unterschiedlich stark Gauß-gefilterten Bildern (*Gauß-Pyramide*) erzeugt, und nebeneinander liegende Bilder werden voneinander subtrahiert. Die Differenzbilder werden dann nach Extremalpunkten durchsucht. Quadratische Interpolation innerhalb der DoG-Pyramide, eine Kontrastuntersuchung und ein Maß nach Harris zur Kantenunterdrückung (siehe oben [Har88]) sorgen für subpixelgenau lokalisierte, skalierungsinvariante Merkmale mit hoher Matching-Stabilität. Das Verfahren ist allerdings sehr rechen- und speicherintensiv.

**SURF-Detektor** (2006) [Bay06; Bay08] Wie SIFT detektiert auch SURF (Speeded Up Robust Features) Blobs innerhalb des Skalenraums. Anstatt einer auf dem DoG-Prinzip basierenden Bildpyramide zum Aufbau des Skalenraums werden hier Boxfilter zur Approximation des LoG-Operators verwendet (Determinant-of-Hessian, kurz DoH) und auf Integralbildern ausgewertet, wodurch die Rechenanforderungen im Vergleich zu SIFT stark sinken.

**FAST-Detektor** (2006) [Ros06; Ros10] Beim FAST-Eckendetektor (Features from Accelerated Segment Test) wird ein zentrales Pixel mit den 16 sich auf einem umliegenden Kreis befindenden Pixeln verglichen. Existiert auf dem umliegenden Kreis ein geschlossen helleres oder dunkleres Segment von z. B. neun Pixeln, so liegt eine Ecke vor. Eine hohe Detektionsgeschwindigkeit wird dadurch erzielt, dass Pixel, die keine Ecke sind, durch eine zum einen geschickte und zum anderen mit Machine-Learning-Verfahren trainierte Reihenfolge der Pixelvergleiche frühzeitig ausgeschlossen werden können. Für eine Ecke

wird ein Score-Wert bestimmt, auf Basis dessen eine Non-maximal-Suppression die Ecken abschließend filtert.

**ORB-Detektor** (2011) [Rub11] Ein auf FAST basierendes Verfahren ist der ORB (Oriented FAST and rotated BRIEF) Detektor. Neben dem Aufbau einer Bildpyramide zur Bestimmung der Merkmalsskalierung wurde zudem ein Maß zur Kantenunterdrückung nach Harris (siehe oben [Har88]) ergänzt.

**BRISK-Detektor** (2011) [Leu11] Der BRISK-Detektor (Binary Robust Invariant Scalable Keypoints) benutzt den sog. AGAST-Algorithmus, welcher eine Weiterentwicklung von FAST ist, und filtert die erkannten Ecken zusätzlich anhand derselben Score-Metrik wie FAST. Durchgeführt wird die Detektion auf einer Gauß-Pyramide zur Bestimmung der Merkmalsskalierung.

**AKAZE-Detektor** (2012) [Alc12; Alc13] Da skalenraumbasierte Ansätze wie SIFT und SURF auf Gauß-Filtern beruhen, die weder Objektgrenzen noch Unterschiede zwischen Bilddetails und Rauschen berücksichtigen, wurden AKAZE-Merkmale entwickelt. Das Prinzip der anisotropen Diffusion erzeugt hierbei einen nicht-linearen Skalenraum mit bildinhaltsabhängigen Filterkerneln. Die Skalenraumbilder werden dann nach Extremstellen der Hesse-Determinanten (DoH) durchsucht und die Merkmalspositionen im Subpixelbereich lokalisiert. Die Extraktion von AKAZE-Merkmalen ist etwas weniger rechenintensiv als SIFT und liefert ebenfalls hohe algorithmische Stabilität. Dennoch ist zur echtzeitfähigen Verarbeitung bei AKAZE eine dedizierte Implementierung erforderlich (z. B. [Kal17]).

### Bildmerkmalsdeskriptoren

Sobald ein Bildmerkmal in seiner Position mit seiner Skalierung und seiner Orientierung detektiert wurde, muss die Bildstruktur in einer Nachbarschaft um diese Position herum in einen Deskriptor für stabiles Matching und robust gegenüber lokalen Bilddeformationen kodiert werden. Der Deskriptor sollte an der Merkmalsorientierung ausgerichtet und proportional zur Merkmalsskalierung sein [Awa16]. Wie Merkmalsdetektoren weisen auch die Algorithmen zur Merkmalsdeskription Unterschiede auf. Maße wie die Summe quadrierter Differenzen oder normalisierte Kreuzkorrelation haben nur dann Vorteile, wenn die Bewegungen um ein Merkmal hauptsächlich translatorisch sind. Wenn die Merkmalsumgebung sich in Orientierung und Skalierung unterscheidet oder sogar affinen Transformationen unterliegt, sind komplexere Ansätze notwendig [Sze11]. Dabei werden zwei Klassen unterschieden: die histogrammbasierten und die binären Deskriptoren, die im Folgenden anhand ausgewählter Algorithmen vorgestellt werden. Für weitere Deskriptoren sei auf die Literatur verwiesen [Mik05b; Hei09; Awa16; Fan19].

**SIFT-Deskriptor** (2004) [Low04b] Der SIFT-Deskriptor ist ein histogrammbasierter Deskriptor. Um Invarianz gegenüber Rotation und großen perspektivischen Veränderungen zu erzeugen, muss zunächst einem Merkmalspunkt eine Hauptorientierung zugewiesen

werden. Dafür wird in einer Umgebung des Merkmals, deren Größe von der Skalierung abhängig ist, für jedes Pixel der Gradient gebildet (Betrag und Winkel) und in einem Histogramm akkumuliert. Das Histogrammmaximum entspricht der Orientierung. Anschließend werden die Gradienten aus einer 16×16 Pixel großen Umgebung um das Merkmal, die entsprechend der Hauptorientierung gedreht ist, in einem zweiten Histogramm aufakkumuliert. Dabei werden je 4×4 Pixel zusammengefasst und die Winkel auf acht Klassen begrenzt, sodass sich ein 128 Einträge großer Vektor ergibt. Den letzten Schritt bildet eine Normalisierung, ein Clipping und eine erneute Normalisierung, wodurch Robustheit gegenüber unterschiedlichen Lichtsituationen erreicht werden soll.

**SURF-Deskriptor** (2006) [Bay06; Bay08] Beim SURF-Deskriptor werden in einer Merkmalsumgebung Haar-Wavelets verwendet, um sowohl die Hauptorientierung innerhalb einer Kreisumgebung um das Merkmal als auch ein Histogramm zu ermitteln, das mit einer Dimension von 64 Einträgen kleiner ist als bei SIFT, allerdings algorithmische Nachteile bzgl. Stabilität gegenüber Rotation und Blickpunktänderungen zeigt.

**BRIEF-Deskriptor** (2010) [Cal10; Cal12] Der BRIEF-Deskriptor (Binary Robust Independent Elementary Features) besteht aus einem binären String, der durch pixelweise Helligkeitsvergleiche innerhalb einer Umgebung bestimmt wird. Dabei werden nacheinander Abtastpunkte in der Umgebung untersucht, die zuvor zufällig nach einer Gauß-Verteilung generiert wurden. Dabei reicht für viele Anwendungen eine String-Länge von 128 Bit, allerdings ist der Deskriptor nicht invariant gegenüber Rotation.

**BRISK-Deskriptor** (2011) [Leu11] Ähnlich wie bei BRIEF werden auch beim BRISK-Deskriptor pixelweise Vergleiche durchgeführt. Allerdings ist dabei das Abtastmuster nicht mehr zufällig und es wird zuvor eine Orientierung bestimmt.

**ORB-Deskriptor** (2011) [Rub11] Die Merkmalsorientierung wird hier mittels eines Intensitätszentroids an der Merkmalsposition bestimmt. Der Detektor besteht aus einem BRIEF-Testmuster, das um die Orientierung gedreht wird. Zur beschleunigten Ausführung werden die Orientierungen auf 12° diskretisiert und die gedrehten Muster in einer Tabelle abgelegt.

**AKAZE-Deskriptor** (2013) [Alc13] Der AKAZE-Deskriptor ist ein binärer Deskriptor und wird durch Vergleiche der durchschnittlichen Intensität und der Gradientenbilder innerhalb eines Raster erzeugt. Das Raster wird zuvor um eine Hauptorientierung gedreht, um Rotationsinvarianz zu erhalten. Für die Hauptorientierung werden innerhalb einer Kreisumgebung um das Merkmal die Ableitungen in x- und y-Richtung mit einer Gauß-Funktion gewichtet, als Vektoren interpretiert, und der längste Vektor wird ausgewählt.

### Deskriptor-Matching

Um Deskriptoren verschiedener Bilder einander zuzuordnen und so die gewünschte Liste von Punktkorrespondenzen zu erhalten, ist der Schritt des Deskriptor-Matchings notwen-

dig. Dabei wird ein Deskriptor eines ersten Bildes mit allen Deskriptoren eines zweiten Bildes verglichen. Bei einer hohen Anzahl an Merkmalen hat die quadratische Laufzeit dieses Vorgehens einen starken Einfluss auf die Gesamtperformance des Systems. Bei geringen perspektivischen Veränderungen zwischen den Bildern ist die Positionsverschiebung der Merkmale limitiert, sodass der Suchbereich eingeschränkt werden kann, für rektifizierte Stereobilder sogar auf eine Bildzeile. Effiziente Datenstrukturen können dabei helfen, nur die Deskriptoren auszuwählen, die auch wirklich betrachtet werden müssen.

Als Distanzmetriken zum Vergleich zweier Deskriptoren $\mathbf{d}_x$ und $\mathbf{d}_y$ der Länge $l$ dient für histogrammbasierte Deskriptoren meist die L1-Norm

$$\Delta = ||\mathbf{d}_x - \mathbf{d}_y||_1 = \sum_{i=1}^{l} |d_{x,i} - d_{y,i}| \tag{2.1}$$

oder die Euklidische Norm (L2-Norm)

$$\Delta = ||\mathbf{d}_x - \mathbf{d}_y||_2 = \sqrt{\sum_{i=1}^{l} |d_{x,i} - d_{y,i}|^2}. \tag{2.2}$$

Für binäre Deskriptoren eignet sich die weniger aufwendige Hamming-Distanz, welche durch die bitweisen XOR-Verknüpfungen in Implementierungen deutliche Performance-Vorteile gegenüber der Euklidischen Norm hat:

$$\Delta = ||\mathbf{d}_x - \mathbf{d}_y||_{ham} = \sum_{i=1}^{l} d_{x,i} \oplus d_{y,i}. \tag{2.3}$$

Die Matching-Distanz $\Delta$ zwischen zwei Deskriptoren gibt an, ob es sich um einen Match handelt, also die Deskriptoren (und damit die Bildmerkmale und ihr umliegender Bildbereich) zueinander passen oder nicht. Dieser Zusammenhang ist in Abb. 2.4 grafisch dargestellt. Durch Vergleich aller Deskriptoren miteinander ergeben sich die zwei Mengen der *true positives (TP)* und *true negatives (TN)*. Aufgetragen über die Matching-Distanz überlappen die beiden Mengen. Da die jeweilige Zugehörigkeit eines Matches zu einer dieser Mengen im Allgemeinen nicht bekannt ist, ist ein Akzeptanzmaß notwendig, um die Gesamtheit der Merkmals-Matches algorithmisch wieder in diese Mengen zu trennen.

Ein Akzeptanzmaß kann z. B. ein Schwellwert $\theta$ sein, der angibt, ab welcher minimalen Distanz ein Match als korrekt bewertet wird. Eine weitere Methode ist die Auswahl des Nearest-Neighbor (NN), bei dem die zwei Deskriptoren mit der geringsten Distanz als korrekt eingeordnet werden. Eine dritte Methode ist das Nearest-Neighbor-Distance-Ratio (NNDR), bei dem das Verhältnis zwischen der erst- und der zweitkleinsten Distanz gebildet wird. Nur dann, wenn das Verhältnis groß genug ist, also die erstkleinste Distanz einen großen Abstand zur zweitkleinsten besitzt, wird das Match als eindeutig genug eingestuft und als korrekt akzeptiert [Sze11].

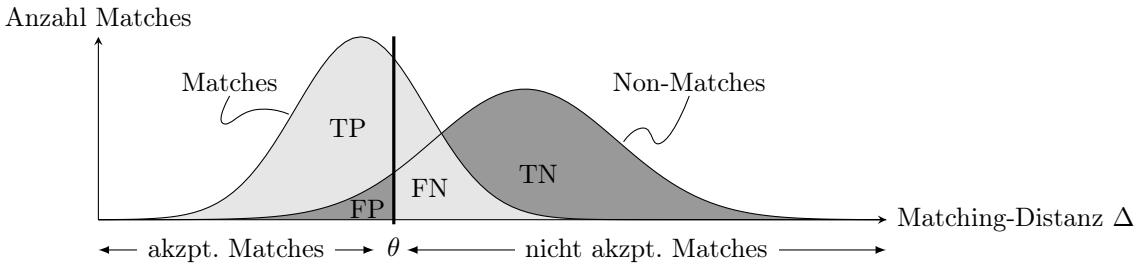



**Abbildung 2.4:** Verteilung von Matches und Non-Matches als Funktion der Matching-Distanz $\Delta$. Eine Erhöhung des Schwellwertes $\theta$ bewirkt eine Vergrößerung der Anzahl an akzeptierten Matches (*true positives (TP)* und *false positives (FP)*). [Sze11]

Mittels der Merkmalsdistanz ist es somit möglich, potentiellen Matches eine Rangfolge zu geben. Das jeweilige Akzeptanzmaß bestimmt dabei die Anzahl der akzeptierten Matches. Ein zu optimistisches Akzeptanzmaß erhöht allerdings die Menge an *false positives (FP)*, ein zu pessimistisches Maß unterdrückt *true positives (TP)*, wodurch die Wahl eines geeigneten Maßes zu einer anwendungsabhängigen und empirischen Aufgabe wird.

### 2.2.2 Bewertungsmetriken

Um die Qualität von Algorithmen der Merkmalsextraktion anwendungsunabhängig zu beurteilen und insbesondere den Merkmalsdetektor und den Merkmalsdeskriptor separat zu untersuchen, werden in der Literatur vor allem die Metriken *Repeatability* und *Precision* verwendet. Zusätzlich wurde in dieser Arbeit als Anwendung die *merkmalsbasierte Eigenbewegungsschätzung (Visual Odometry)* gewählt.

#### Repeatability

Wenn die gefundenen Matches alle auf einer Ebene liegen, die Bilder vom gleichen Kamerazentrum aufgenommen wurden oder die Korrespondenzen sehr weit von der Kamera entfernt sind, dann kann die Beziehung zwischen zwei Kameraposen durch eine perspektivische Transformation (Homographie) beschrieben werden (im letzten Fall nur näherungsweise) [Har03]. Ist diese Homographie z. B. durch einen entsprechenden experimentellen Aufbau bekannt, kann durch Matrixmultiplikation ein Bildmerkmal aus Bild 1 in Bild 2 projiziert werden (siehe Abb. 2.5). Liegt in einem Bereich um das projizierte Bildmerkmal auch ein in Bild 2 gefundenes Bildmerkmal, so ist dies ein Qualitätsmaß des Merkmalsdetektors und wird in [Sch00] als *Repeatability* $r_i$ bezeichnet.

$$r_i(\epsilon) = \frac{|\{(x_1 \in I_1, x_i \in I_i) \mid ||x_i - H_{1i} \cdot x_1|| < \epsilon\}|}{\min(|\{I_1\}|, |\{I_i\}|)} \tag{2.4}$$

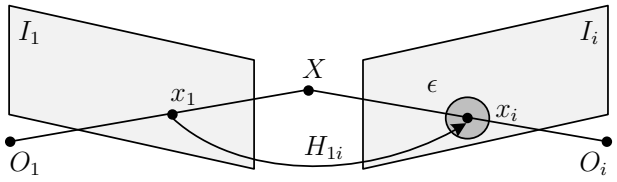



**Abbildung 2.5:** Repeatability nach [Sch00]. Die Punkte $x_1$ und $x_i$ sind die Projektionen vom 3D-Punkt $X$ in die Bilder $I_1$ und $I_i$. Ein detektiertes Merkmal $x_1$ gilt als *wiederholt*, wenn in einer $\epsilon$-Umgebung um $x_i$ ein Merkmal detektiert wird. In planaren Szenen sind $x_1$ und $x_i$ über die Homographie $H_{1i}$ miteinander verbunden.

mit $I_1$ und $I_i$ der Menge an detektierten Merkmalen in den Bildern 1 und $i$, $H_{1i}$ der zugehörige Homographie-Abbildung und $\epsilon$ einem Schwellwert, da sich Merkmalspunkte niemals exakt, sondern nur innerhalb eines Toleranzbereichs lokalisieren lassen, insbesondere wenn keine explizite Subpixel-Lokalisierung durchgeführt wird [Ros06]. Die Repeatability beschreibt das Verhältnis zwischen der Anzahl an sich im zweiten Bild wiederholenden Merkmalen und der Gesamtzahl der Merkmale.

### Precision

Die algorithmische Leistungsfähigkeit eines Deskriptors beim Matching kann anhand der Precision bewertet werden [Gau11]:

$$precision = \frac{\#\text{true positives}}{\#\text{true positives} + \#\text{false positives}} = \frac{\#\text{true positives}}{\#\text{matches}} \tag{2.5}$$

Um true positives von false positives zu unterscheiden, ist für diese Metrik, wie auch schon für die Repeatability, eine Ground Truth Referenz notwendig. Eine Möglichkeit ist eine visuelle, manuelle Kontrolle der Zuordnungen, was aber bei hoher Merkmalsanzahl langwierig und zunehmend fehleranfällig ist. Die Autoren von [Mik05b] stellen dafür den Overlap-Fehler vor. Dieser gibt an, ob sich die Bildregionen, die zum Berechnen des Deskriptors verwendet werden, überlappen. Grundlage hier ist erneut eine Homographie, die die Bildregion in einem ersten Bild in ein zweites Bild transformiert.

### Merkmalsbasierte Eigenbewegungsschätzung

Am Ende der algorithmischen Kette, die mit der Bildmerkmalsextraktion und Korrespondenzsuche beginnt, steht die Anwendung, in der die Merkmale bzw. Merkmalskorrespondenzen verwendet werden. In dieser Arbeit wird neben den beiden obengenannten Metriken auch eine merkmalsbasierte Eigenbewegungsschätzung (Visual Odometry) als Qualitätsmetrik vorgestellt, um verschiedene Implementierungen der Merkmalsextraktion in einer realen Anwendung miteinander zu vergleichen. Dabei wird die Bewegung

einer Kamera entlang einer Referenzstrecke (z. B. bei einem autonom fahrenden Auto oder Roboter) auf Basis der Punktkorrespondenzen der aufgenommenen Bildsequenz berechnet.

Der hier verwendete Algorithmus basiert auf dem in [Gei11] vorgestellten Verfahren. Im Folgenden wird eine im Rahmen dieser Arbeit implementierte und verwendete Variante beschrieben, die in Details von [Gei11] abweicht. Als Eingangsbilddaten werden rektifizierte Stereobildpaare verwendet, auf denen eine Merkmalsextraktion durchgeführt wird. Für das Matching wird die Nearest-Neighbor-Methode als Akzeptanzmaß verwendet, wobei nur diejenigen Korrespondenzen als korrekt akzeptiert werden, die sowohl vom ersten ins zweite Bild als auch vom zweiten ins erste Bild die geringste Deskriptordistanz aufweisen. Danach werden daraus *zirkuläre* Korrespondenzen berechnet, d. h. es werden nur diejenigen Korrespondenzen weiterverwendet, die sowohl in den beiden Bildern zum aktuellen Zeitpunkt (*aktuelles* Bildpaar), als auch in den Bildern zum Zeitpunkt davor (*vorangegangenes* Bildpaar) zu finden sind und sich einander zuordnen lassen. Die sich daraus ergebenden Korrespondenzen aus vier Merkmalen werden als stabil genug angesehen und dienen als Eingangswerte für die Bewegungsrekonstruktion.

Sei $\mathbf{X} = (x, y, z)^T$ ein stationärer 3D-Echtweltpunkt, der an der gleichen Position bleibt, wenn die Kamera sich von einem Bild zum nächsten bewegt, und $\mathbf{x} = (u, v)^T$ seine 2D-Abbildung auf die Bildebene. Dann gilt unter Verwendung des Lochkameramodells und der Annahme quadratischer Pixel folgende Projektionsgleichung eines 3D-Punktes in ein 2D-Bild [Gei11]:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & c_u \\ 0 & f & c_v \\ 0 & 0 & 1 \end{pmatrix} \left[ \begin{pmatrix} \mathbf{R}(\mathbf{r}) & \mathbf{t} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} - \begin{pmatrix} s \\ 0 \\ 0 \end{pmatrix} \right] \tag{2.6}$$

mit $f$ der Brennweite, $(c_u, c_v)$ dem Bildmittelpunkt, $\mathbf{R}(\mathbf{r}) = \mathbf{R}_x(r_x)\mathbf{R}_y(r_y)\mathbf{R}_z(r_z)$ der Rotationsmatrix, $\mathbf{t} = \begin{pmatrix} t_x & t_y & t_z \end{pmatrix}^T$ dem Translationsvektor und $s$ der Verschiebung entlang der Basislinie mit $s = 0$ (Projektion in linkes Bild) bzw. $s = B$ (Projektion in rechtes Bild). $\begin{pmatrix} \mathbf{R} & \mathbf{t} \end{pmatrix}$ repräsentieren dabei die Euklidische Transformation, die durch die Kamerabewegung erzeugt wird und deren sechs Parameter es zu schätzen gilt. Die Projektion nach Gl. 2.6 eines 3D-Punktes $\mathbf{X}$ auf ein Pixel im linken Bild $\mathbf{x}_i^{(l)} \in \mathbb{R}^2$ sei nun als $\pi^{(l)}(\mathbf{X}; \mathbf{r}, \mathbf{t}) : \mathbb{R}^3 \to \mathbb{R}^2$ bezeichnet. In gleicher Weise beschreibe $\pi^{(r)}(\mathbf{X}; \mathbf{r}, \mathbf{t})$ die Projektion auf ein Pixel im rechten Bild.

Aufgrund des Stereoaufbaus und der zugrunde liegenden Epipolargeometrie gilt für alle gefundenen Merkmale im linken Bild die Gl. [Har03; Sze11]

$$Z = \frac{f \cdot B}{d} = \frac{f \cdot B}{x_l - x_r} \tag{2.7}$$

mit dem Abstand $Z$ des Bildpunktes vom linken Kamerazentrum, der Brennweite $f$, der Basislinie $B$ und der Disparität $d$, welche der Differenz der x-Positionen der Merkmale

aus dem linken und rechten Bild entspricht. Mithilfe der Gl. 2.7 kann der zu einer 2D-Punktkorrespondenz gehörende 3D-Echtweltpunkt errechnet werden.

Zur Schätzung der Bewegungsparameter werden nun zunächst aus den zirkulären Korrespondenzen die 2D-Positionen des vorangegangenen Bildpaares verwendet, um mit Gl. 2.7 ihre 3D-Echtweltposition zu bestimmen. Danach wird folgender Rückprojektionsfehler in Abhängigkeit der Bewegungsparameter $(\mathbf{r}, \mathbf{t})$ mittels Gauß-Newton-Optimierung minimiert:

$$\sum_{i=1}^{N} ||\mathbf{x}_i^{(l)} - \pi^{(l)}(\mathbf{X}_i; \mathbf{r}, \mathbf{t})||^2 + ||\mathbf{x}_i^{(r)} - \pi^{(r)}(\mathbf{X}_i; \mathbf{r}, \mathbf{t})||^2. \tag{2.8}$$

Dabei bezeichnen $\mathbf{x}_i^{(l)}$ und $\mathbf{x}_i^{(r)}$ die Merkmalspositionen im aktuellen Bildpaar. Um robust gegenüber Ausreißern bzw. zirkulären Korrespondenzen auf nicht-stationären Objekten zu sein, wird die Fehlerschätzung zunächst in einem RANSAC-Schema [Fis81] mit drei zufälligen zirkulären Korrespondenzen und 20 Iterationen durchgeführt. Alle validen Korrespondenzen der Gewinneriteration werden dann für eine finale Schätzung verwendet. Im Ergebnis resultieren daraus dann die Bewegungsparameter von Bild zu Bild, die zu einer Trajektorie zusammengefügt werden können. Die translatorische Geschwindigkeit der Bewegung ergibt sich dabei aus dem Betrag des Vektors $\mathbf{t}$.

Um die Qualität der geschätzten Kameratrajektorie anhand einer Bildsequenz zu bewerten, stellt ein reiner Endpunkt-Fehler zwischen Referenz und Schätzung keine faire Metrik dar, da der Zeitpunkt, zu dem ein Fehler in der Schätzung auftritt, einen starken Einfluss auf eine solche Metrik besitzt. Schätzfehler in der Rotation zu einem frühen Zeitpunkt der Referenzfahrt führen zu großen Endpunkt-Fehlern. Daher werden in [Gei12] die Abweichung zwischen Referenz und Schätzung in Rotation und Translation separat voneinander als Funktion der Geschwindigkeit und Trajektorienlänge ausgewertet. Dadurch können leichter Rückschlüsse auf die Qualität und Schwächen des Algorithmus gezogen werden. Formal lassen sich der rotatorische und der translatorische Fehler wie folgt formulieren:

$$E_{rot}(\mathcal{F}) = \frac{1}{|\mathcal{F}|} \sum_{(i,j)\in\mathcal{F}} \angle[(\hat{\mathbf{p}}_j \ominus \hat{\mathbf{p}}_i) \ominus (\mathbf{p}_j \ominus \mathbf{p}_i)] \tag{2.9}$$

$$E_{trans}(\mathcal{F}) = \frac{1}{|\mathcal{F}|} \sum_{(i,j)\in\mathcal{F}} ||(\hat{\mathbf{p}}_j \ominus \hat{\mathbf{p}}_i) \ominus (\mathbf{p}_j \ominus \mathbf{p}_i)|| \tag{2.10}$$

wobei $\mathcal{F}$ ein Bildpaar $(i, j)$ aus der Bildsequenz, $\hat{\mathbf{p}}$ und $\mathbf{p}$ die geschätzte und wahre Kamerapose und $\mathbf{p}_j \ominus \mathbf{p}_i$ [Küm09] die relative Transformation von Kamerapose $\mathbf{p}_i$ nach $\mathbf{p}_j$ unter Verknüpfung der Einzelposen bezeichnen.

### 2.2.3 Algorithmischer Entwurfsraum

In Abschnitt 2.2.1 wurde eine Reihe von Verfahren zur Merkmalsextraktion vorgestellt. Tab. 2.1 soll einige von ihnen noch einmal übersichtlich in ihren wesentlichen Eigenschaf-

**Tabelle 2.1:** Gegenüberstellung der wesentlichen algorithmischen Eigenschaften verschiedener Algorithmen zur Merkmalsextraktion

| Name | Detektor | Deskriptor | Skalierungs-invarianz | Skalen-raum | Rotations-invarianz |
|---|---|---|---|---|---|
| Harris [Har88] | Ecken | - | nein | - | nein |
| SIFT [Low99; Low04b] | Blob (DoG) | Histogramm | ja | linear | ja |
| SURF [Bay06; Bay08] | Blob (DoH) | Histogramm | ja | linear | ja |
| FAST [Ros06; Ros10] | Ecken | - | nein | - | nein |
| BRIEF [Cal10; Cal12] | - | Binärstring | nein | - | nein |
| BRISK [Leu11] | Ecken | Binärstring | ja | linear | ja |
| ORB [Rub11] | Ecken | Binärstring | ja | linear | ja |
| AKAZE [Alc13] | Blob (DoH) | Binärstring | ja | nicht-linear | ja |

ten gegenüberstellen. Nicht abgebildet sind hier regionsbasierte Verfahren und spezielle Verfahren zur verbesserten Invarianz gegen affine Transformationen, die in der Literatur ebenfalls aufgeführt werden (z. B. [Tuy08; Aan12]). Das Matching von histogrammbasierten Deskriptoren ist durch die L2-Norm komplexer als die Hamming-Distanz im Fall von binären Deskriptoren, da diese eine wesentlich kompaktere Darstellungsform bieten, allerdings im Gegenzug auch weniger Information beinhalten und daher weniger robust sind. Die Betrachtung des Skalenraums, sowie die Berechnung einer Orientierung und Ausrichtung des Deskriptors für Rotationsinvarianz erhöht die Komplexität des Algorithmus deutlich zugunsten erhöhter Robustheit.

Deutlich wird dies auch in den erforderlichen Ausführungszeiten pro Extraktion eines Merkmals, welche in Tab. 2.2 exemplarisch für verschiedene Algorithmen aufgelistet sind. Zwar unterscheiden sich die Hardware, die verwendeten Bildauflösungen und auch die Software-Implementierungen, allerdings lassen sich trotzdem Tendenzen erkennen. Deutliche Unterschiede aufgrund der verschiedenen algorithmischen Komponenten sind erkennbar, wobei SIFT stets die höchste Ausführungszeit hat, gefolgt von SURF. FAST-BRIEF sind im Vergleich um Größenordnungen schneller. BRISK und ORB stellen weitere schnelle Alternativen mit erhöhter Robustheit dar. AKAZE ist das modernste der Verfahren bei im Vergleich mittleren Ausführungszeiten.

Die Leistungsfähigkeit eines Algorithmus lässt sich allerdings aus diesen beiden Gegenüberstellungen nicht ableiten. Die Entscheidung für oder gegen einen Algorithmus ist immer in Abhängigkeit von der zu Grunde liegenden Anwendung zu treffen [Aan12; Men18]. Es ist nicht eindeutig, was ein Bildmerkmal *interessant* macht, da es dafür keine klare Definition gibt und dies stets im Kontext der Anwendung bewertet werden muss. Die Detektion von Bildmerkmalen ist lediglich ein Zwischenschritt in einer Anwendung [Gau11]. Es lässt sich in der Literatur eine hohe Anzahl an Veröffentlichungen finden, die verschiedene Detektor-Deskriptor Kombinationen für unterschiedlichste Benchmarks und Computer-Vision-Anwendungen evaluieren.

In [Mik05b] vergleichen die Autoren Deskriptoren basierend auf skalierungs- und affininvarianten Detektoren. Neben ihrem eigenen Deskriptor GLOH schneidet SIFT bei

**Tabelle 2.2:** Vergleich der Rechenzeiten bei der Extraktion von Bildmerkmalen

| Ref. | Alg. | $\mu s$ pro Merkmal | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | SIFT | SURF | AKAZE | ORB | FAST | BRIEF | BRISK |
| [Chi16][2] | dt+dc[1] | 128 | 44 | 72 | 9 | | | |
| [Tar18][3] | dt+dc | 88 | 43 | 59 | 14 | | | 21 |
| [Kha15][4] | dt | 121 | 104 | | 8 | 2 | | |
| [Kha15][4] | dc | 581 | 448 | | 12 | | 2 | 5 |
| [Hei12][5] | dt | 119 | 100 | | 2 | 0,3 | | |
| [Hei12][5] | dc | 314 | 143 | | 5 | | 4 | 13 |

[1] dt: Detektor, dc: Deskriptor
[2] Intel Core i7 @ 3,2 GHz, $1242 \times 376$ px
[3] Intel Core i5 @ 3,4 GHz, $800 \times 640$ px
[4] Intel Core i5 @ 2,7 GHz, $640 \times 480$ px
[5] Intel Xeon @ 2,67 GHz, $800 \times 640$ px

geometrischen und fotometrischen Transformationen als zweitbester ab. Dabei wird der Oxford-Datensatz [Mik05a] verwendet, der im Anschluss großen Anklang gefunden hat und als weit verbreitetes Benchmark zum Untersuchen von Bildmerkmalen eingesetzt wird. Zwischen zwei Bildern sind jeweils die Homographien angegeben, sodass es leicht möglich ist, die oben genannte Repeatability (siehe Gl. 2.4) und Precision (siehe Gl. 2.5) zu ermitteln. Die Störfaktoren, die der Datensatz beinhaltet, sind verschiedene Grade von Bildweichzeichnung, perspektivische Änderungen, Belichtung, Beleuchtung, Rotation und Skalierung. Die Autoren von [Li08] bestätigen die Ergebnisse von [Mik05b]. In [Gau11] werden Bildmerkmale speziell für bildbasiertes Tracking untersucht, jedoch kann kein klarer Sieger festgestellt werden, da die algorithmische Performance abhängig von der untersuchten Bewegung und Bewegungsunschärfe ist. In [Pro15] werden u. a. SIFT, SURF und BRIEF verglichen, wobei SURF auf verrauschten Bildern und SIFT auf hochqualitativen Bildern bessere Ergebnisse liefern. BRIEF schneidet am schlechtesten ab. Ein simples Ranking wollen die Autoren nicht zulassen, sondern jeder Algorithmus liefert für sich genommen das beste Ergebnis in bestimmten Bereichen des algorithmischen Entwurfsraums. In [Kha15] sind Detektor-Deskriptor-Kombinationen aus SIFT, SURF und BRISK unter einer Vielzahl von anderen Varianten diejenigen mit der höchsten Qualität in Hinsicht auf Präzision, Recall und statistische Tests. Die Anwendungen hier sind Objekterkennung und Merkmalsmatching. Die Autoren von [Tar18] vergleichen SIFT, SURF, AKAZE, ORB und BRISK für Image Stitching und werten dafür die Repeatability und die Rekonstruktion der Homographie aus. Aus ihrer Sicht sind hierfür SIFT und BRISK die besten Algorithmen. Image Stitching ist auch die Anwendung in [Dha18], in der SIFT, SURF und ORB verglichen werden. In der Anwendungsklasse der Visual Odometry, also der Rekonstruktion von Bewegungsparametern anhand von Bilddaten, erzielen in [Chi16] SIFT und SURF die besten und in [Xie17] mit anderen Verfahren vergleichbare Ergebnisse. Auch die Untersuchung in [Don20] vergleicht für diese Anwendung SIFT mit moderneren Verfahren. In [Fan19] werden für 3D-Rekonstruktion

klassische Verfahren wie SIFT mit Verfahren verglichen, die auf maschinellem Lernen basieren. Für die Autoren ist in diesem Zusammenhang SIFT noch immer die erste Wahl, allerdings zeigen neuere Verfahren großes Potential.

Insgesamt machen die wenigsten Studien eine klare Aussage zu einem algorithmischen Gewinner. Allen Vergleichen gemein ist allerdings eine durchgehend positive Bewertung von DoG-basierten Detektoren und histogrammbasierten Deskriptoren, die auf der Verteilung der lokalen Orientierungen beruhen, wie der SIFT-Deskriptor. Aufgrund seiner hohen algorithmischen Qualität in einer Vielzahl von Anwendungen hat dieser Algorithmus einen signifikanten, grundlegenden Einfluss auf das Forschungsgebiet der Bildmerkmalsextraktion genommen. Obwohl er vor fast zwei Jahrzehnten vorgestellt wurde, wird er selbst in aktuellen Untersuchungen immer noch betrachtet und liefert dort gute Ergebnisse. Deutlich wird dies auch an der Anzahl an Zitierungen der beiden Arbeiten [Low99] (20k[1]) und [Low04b] (61k[1]). Darüber hinaus unterlag der Algorithmus in den USA bis zum 06.03.2020 einem Patent [Low04a]. Die hohen Rechenleistungsanforderungen schließen SIFT allerdings für Echtzeitanwendungen mit begrenztem Verlustleistungsbudget oft aus, und es muss ein weniger komplexes und zugleich allerdings auch weniger robustes Verfahren gewählt werden. Aus diesen Gründen wird der SIFT-Algorithmus in dieser Arbeit als Referenzalgorithmus gewählt.

### 2.2.4 Scale-Invariant Feature Transform (SIFT)

Der Algorithmus *Scale-Invariant Feature Transform (SIFT)* wurde erstmals in [Low99] als Blob-Detektor vorgestellt und in [Low04b] um einen histogrammbasierten Deskriptor ergänzt. Da der Algorithmus, wie im Abschnitt zuvor erläutert, in dieser Arbeit als Referenzalgorithmus dient, soll er in diesem Abschnitt im Detail vorgestellt werden. Der Algorithmus ist in vier Stufen eingeteilt:

- Extremadetektion im Skalenraum
- Lokalisierung des Merkmals
- Berechnung der Merkmalsorientierung
- Beschreibung des Merkmals (Deskriptor)

#### Extremadetektion im Skalenraum

Um die Position von Bildmerkmalen, die invariant gegenüber Skalierungsänderungen des Bildes sind, zu ermitteln, muss das Bild nach stabilen Merkmalen auf *allen möglichen* Skalierungen durchsucht werden. Daher besteht der erste Schritt des SIFT-Algorithmus aus dem Aufbau einer Bildreihe, die den Skalenraum des Eingangsbildes repräsentiert. Der Skalenraum eines Bildes ist definiert als eine Funktion $L(x, y, \sigma)$, die man durch die

[1] Google Scholar - 18.02.2021

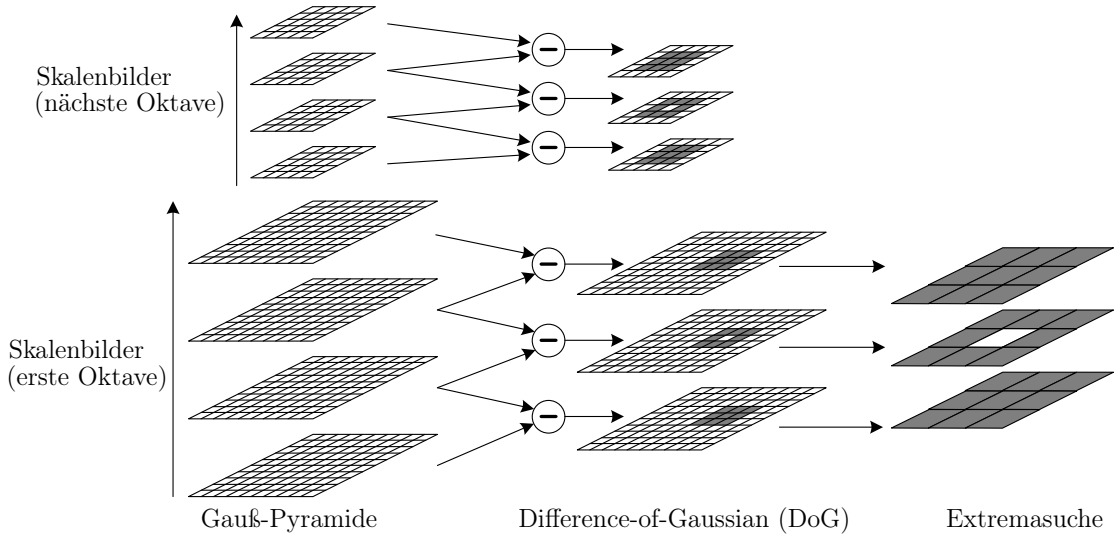



**Abbildung 2.6:** Darstellung der Extremadetektion im Skalenraum nach [Low04b]. Abgebildet sind zwei Oktaven mit je vier Skalenbildern. Dies entspricht einer Verdopplung der Standardabweichung nach drei Intervallen ($s = 3$, $k = 2^{1/3}$).

Faltung eines variablen Gauß-Kernels $G(x, y, \sigma)$ mit einem Eingangsbild $I(x, y)$ erhält:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{2.11}$$

wobei $\sigma$ die Standardabweichung eines Gauß-Kernels ist, der folgendermaßen beschrieben wird:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}. \tag{2.12}$$

Eine Verdopplung von $\sigma$ wird als eine Oktave bezeichnet. Anstatt fortwährender Vergrößerung von $\sigma$ (und dadurch auch der Filtergröße) wird die Eigenschaft ausgenutzt, dass eine Verdoppelung von $\sigma$ innerhalb eines Bildes den gleichen Effekt hat wie ein Downsampling des Bildes um den Faktor zwei, d. h. eine Halbierung der Bildlänge und -breite. Das erste Skalenbild der darauffolgenden Oktave wird durch Faltung dieses kleineren Bildes mit den gleichen Filtern wie zuvor erhalten, weshalb dieser Schritt Gauß-Pyramide genannt wird. Der Skalenraum ist in $\sigma$ kontinuierlich, allerdings wird aus Realisierungsgründen eine Oktave in $s$ Zwischenbilder (Intervalle) eingeteilt, welche dann eine Approximation des Skalenraums darstellen. Um eine Verdoppelung von $\sigma$ innerhalb einer Oktave mit $s$ Intervallen zu erreichen, wird eine Schrittweite von $\sigma$ zwischen zwei Bildern von $k = 2^{1/s}$ verwendet (siehe Abb. 2.6). Die einzelnen Intervallbilder innerhalb einer Oktave entsprechen einer Folge von Bildern mit abnehmenden Details.

Nach Berechnung der Gauß-Pyramide werden die Bilder der einzelnen Intervalle, die sich durch den konstanten Faktor $k$ in ihrer Skalierung voneinander unterschieden,

subtrahiert, wodurch eine Difference-of-Gaussian (DoG) Pyramide entsteht:

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned} \tag{2.13}$$

Unter Zuhilfenahme der Wärmeleitungsgleichung kann gezeigt werden, dass die Differenz der Gauß-Bilder dabei eine Approximation des skalen-normierten Laplace-Operators darstellen, dessen Extrema sich als sehr stabile Bildmerkmale erwiesen haben [Low04b; Lin94].

Die Bildmerkmalsdetektion geschieht nun so, dass die DoG-Pyramide nach lokalen Extrema durchsucht wird. Ein Bildpunkt wird innerhalb seiner $3 \times 3 \times 3$ Nachbarschaft mit seinen acht Nachbarpixeln im gleichen Bild und den neun Pixeln in der Skalierung darüber und darunter verglichen. Wenn das zentrale Pixel ein lokales Minimum oder Maximum darstellt, wird dieses Pixel als möglicher Merkmalskandidat detektiert, der im nachfolgenden Schritt in Position und Skalierung verfeinert und anhand von Stabilitätskriterien final ausgewählt wird.

### Lokalisierung des Merkmals

Um die Matching-Qualität und die Stabilität der gefundenen Merkmalskandidaten zu verbessern, wird eine dreidimensionale quadratische Funktion um die Position des Merkmals gefittet, um eine interpolierte Lokalisierung innerhalb der Position und Skalierung zu erhalten. Zu diesem Zweck wird eine Taylor-Reihenentwicklung bis zum quadratischen Term auf den DoG-Bildern $D(x, y, \sigma)$ durchgeführt, welche verschoben wird, sodass sichergestellt ist, dass der Entwicklungspunkt auf der Kandidatenposition liegt:

$$D(\mathbf{x}) = D + \frac{\delta D^T}{\delta \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\delta^2 D}{\delta \mathbf{x}^2}\mathbf{x}, \tag{2.14}$$

wobei die Hesse-Matrix und die Ableitungen von $D$ durch Differenzen der Nachbarpixel des Merkmalskandidaten approximiert werden und $\mathbf{x} = (x, y, \sigma)^T$ dem Offset von der ursprünglichen Merkmalskandidatenposition entspricht. Die Subpixelposition des Offset-Extremums $\hat{\mathbf{x}}$ wird anhand des Nulldurchgangs der Ableitung von Gl. 2.14 nach $\mathbf{x}$ bestimmt:

$$\hat{\mathbf{x}} = -\frac{\delta^2 D^{-1}}{\delta \mathbf{x}^2}\frac{\delta D}{\delta \mathbf{x}} \tag{2.15}$$

Solange das Offset-Extremum in eine beliebige Richtung größer als 0,5 ist, liegt der interpolierte Merkmalspunkt näher an einer anderen Pixel- oder Skalenposition. Daher wird der Offset zur aktuellen Kandidatenposition hinzuaddiert und Gl. 2.14 wird erneut berechnet.

Den DoG-Wert der verfeinerten Merkmalsposition ergibt die Gleichung

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\delta D^T}{\delta \mathbf{x}}\hat{\mathbf{x}}, \tag{2.16}$$

und er wird zum Ablehnen von Merkmalen mit geringem Kontrast durch Vergleich mit einem variablen Schwellwert verwendet. Die Autoren von [Low04b] wählen dabei einen Wert von 0,03 bei Pixelwerten im Intervall [0,1].

Darüber hinaus hat der Algorithmus eine starke Sensitivität für Merkmale, die auf Kanten liegen und dadurch hohes Maß an Mehrdeutigkeit besitzen, was das Matching erschwert (Aperturproblem [Bin09]). Diese Art von Merkmalen hat im DoG-Bild eine große Hauptkrümmung entlang der Kante und eine kleine in der dazu senkrechten Richtung. Um diese Kanten-Merkmale zu unterdrücken, wird die 2×2 Hesse-Matrix $\mathbf{H}$ an der Merkmalskandidatenposition berechnet, welche schon aus der Subpixel-Lokalisierung zur Verfügung steht. Dabei ist nur das Verhältnis ihrer Eigenwerte $r = \alpha/\beta$ im Vergleich zu einem empirischen Schwellwert von Bedeutung, was erstmals schon beim Harris-Eckendetektor Anwendung gefunden hatte [Har88]. Da die Eigenwerte nicht explizit benötigt werden, sondern lediglich ihr Verhältnis, kann der Test zur Kantenunterdrückung folgendermaßen formuliert werden:

$$\frac{Tr(\mathbf{H})^2}{Det(\mathbf{H})} < \frac{(r+1)^2}{r}, \tag{2.17}$$

wobei die Spur und die Determinante der 2×2 Hesse-Matrix benötigt werden, welche mit wenig Aufwand berechnet werden können. Die Autoren von [Low04b] wählen hier $r = 10$ als Schwellwert.

Wenn ein Merkmalskandidat alle Tests besteht, wird er als stabiles SIFT-Merkmal angesehen. Zur Verdeutlichung der Tests siehe Abb. A.1 im Anhang.

### Berechnung der Merkmalsorientierung

Indem einem Bildmerkmal auf Basis der umliegenden Bildeigenschaften eine Orientierung zugewiesen wird, kann der Deskriptor relativ zu dieser Orientierung dargestellt werden, wodurch eine Invarianz gegenüber Rotation erzielt werden kann. Zur Berechnung der Orientierung wird dasjenige Gauß-Pyramidenbild $L(x, y, \sigma)$ gewählt, das am dichtesten an der detektierten Skalierung des Merkmals liegt. Dann werden in der Umgebung um das Merkmal die Gradienten anhand von Pixeldifferenzen berechnet, aufgeteilt in Betrag und Winkel:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{2.18}$$

$$\theta(x,y) = \arctan\left(\frac{(L(x+1,y) - L(x-1,y)}{(L(x,y+1) - L(x,y-1)}\right) \tag{2.19}$$

Es wird dann ein Winkelhistogramm mit 36 Klassen von je 10° gebildet, und die Gradientenbeträge der um das Merkmal liegenden Pixel werden mit einer Gauß-Funktion gewichtet (1,5 $\sigma$) und aufakkumuliert. Diejenige Winkelklasse mit dem höchsten Betrag wird als Hauptorientierung ausgewählt. Sollte eine zweite Winkelklasse einen Betrag von

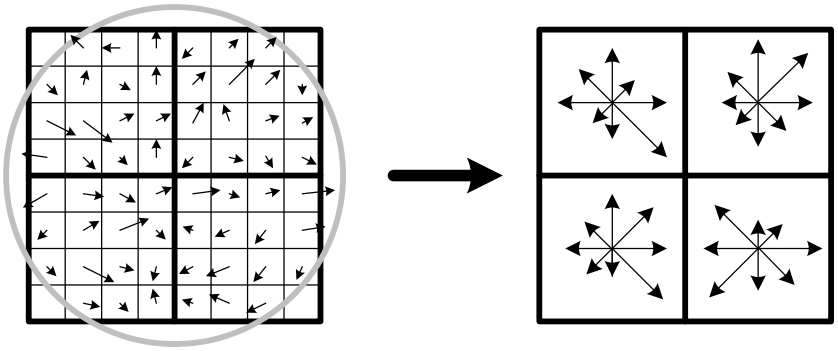


**Abbildung 2.7:** Ein Merkmalsdeskriptor wird berechnet, indem die Bildgradienten in einem Bereich um das Merkmal herum in Orientierungshistogrammen für Subregionen aufakkumuliert werden (hier 2×2). Vor der Akkumulation werden die Gradientenbeträge mit einer Gauß-Funktion gewichtet, welche hier durch einen Kreis dargestellt ist [Low04b].

mehr als 80% des Maximums aufweisen, dann wird das Merkmal mit einer zweiten Orientierung dupliziert. Zum Schluss wird die genaue Position der Orientierung noch durch Interpolation der 3 Winkelklassen, die am dichtesten am Maximum liegen, bestimmt, wodurch eine bessere Genauigkeit erzielt werden kann. Die Umgebung um das Merkmal, die in die Orientierungsberechnung mit einbezogen wird, ist in der Veröffentlichung des Algorithmus [Low04b] nicht genau spezifiziert, kann aber aus öffentlich zugänglichen Referenzimplementierungen (z. B. [Hes10]) zu 4,5 $\sigma$ bestimmt werden.

### Beschreibung des Merkmals (Deskriptor)

Das Skalenbild, das am dichtesten an der detektierten Skalierung des Merkmals liegt, wird zur Bestimmung des Deskriptors gewählt. Eine 16×16 Pixel Umgebung um das Bildmerkmal wird anhand der Orientierung dieses Merkmals gedreht und die Gradientenbeträge in diesem Bereich werden mit einer Gauß-Funktion gewichtet, um den Einfluss der weit vom Merkmalszentrum entfernten Gradienten zu reduzieren, da diese stärker von Fehldetektionen betroffen sind. Die 16×16 Umgebung wird in 4×4 Unterregionen unterteilt und jede der Unterregionen besitzt acht Winkelklassen, sodass ein Vektor mit 128 Einträgen entsteht. Innerhalb einer Unterregion werden die Gradientenbeträge entsprechend ihrer Orientierung aufakkumuliert (siehe Abb. 2.7). Dabei wird trilineare Interpolation verwendet, um die Beträge auf umliegende Histogrammeinträge zu verteilen, sodass sich der Deskriptor nicht abrupt ändert, falls ein Merkmal sich leicht in seiner Position oder Orientierung ändert. Der gesamte Deskriptor wird anschließend auf einen Betrag von eins normiert, um Änderungen im Bildkontrast zu vermeiden. Der Betrag der Histogrammeinträge wird auf einen Maximalwert (hier 0,2) begrenzt, um beim Matching den Schwerpunkt auf die Orientierungsverteilung zu legen und nicht auf hohe Gradientenbeträge, wie z. B. Beleuchtungsänderungen auf dreidimensionalen Flächen es bewirken können. Zum Schluss wird der Deskriptor erneut auf einen Betrag von eins normiert.

# 3 Heterogene FPGA-basierte Architektur zur Bildmerkmalsextraktion

Für Bildverarbeitung mit hoher Datenrate sind dedizierte Hardware-Architekturen unerlässlich. Durch inhärente Parallelität und einen auf den Ziel-Algorithmus zugeschnittenen Datenpfad können FPGAs und ASICs Rechenleistung mit hohem Durchsatz bei großer Energieeffizienz bereitstellen. Ausschließlich durch Software programmierbare General-Purpose-Architekturen wie z. B. CPUs, DSPs oder GPUs liegen in der Energie- und Flächen-Effizienz um Größenordnungen darunter und können dabei gleichzeitig die erforderliche Rechenleistung nur schwer erfüllen. FPGAs sind zudem rekonfigurierbar, sodass sie als Plattform für kommerzielle Produkte anders als ASICs durch ihre Flexibilität auch schon bei kleinen Stückzahlen wirtschaftlich sind [Ban13].

Im Rahmen dieser Arbeit wurde eine heterogene FPGA-basierte Architektur bestehend aus einem VLIW-Prozessor und einem dedizierten Beschleuniger zur Bildmerkmalsextraktion nach dem SIFT-Algorithmus entwickelt, welche in diesem Kapitel im Detail vorgestellt wird. Anwendung und Algorithmus wurden dabei bereits in Abschnitt 2.2 beschrieben. Zunächst werden in Abschnitt 3.1 bestehende dedizierte Architekturkonzepte zur Abbildung von SIFT auf FPGAs und ASICs betrachtet. Darüber hinaus wird die Notwendigkeit einer tiefergehenden Untersuchung hinsichtlich der Systemoptimierung begründet. Die Abschnitte 3.2 bis 3.6 erläutern dann die entwickelte FPGA-Architektur und gehen auf Implementierungsdetails ein. Abschließend wird in Abschnitt 3.7 eine algorithmische Charakterisierung als Vergleich gegenüber einer Software-Referenz präsentiert.

## 3.1 Bestehende Architekturen

Seit seiner Veröffentlichung im Jahr 2004 [Low04b] wurde der SIFT-Algorithmus in der Literatur vielfach auf verschiedenen Hardware-Plattformen implementiert. Seine hohe Komplexität und die damit verbundenen Rechenleistungsanforderungen stellen dabei eine Herausforderung für Echtzeit-Bildverarbeitung in eingebetteten Systemen mit limitiertem Verlustleistungsbudget dar. Der Begriff *Echtzeit* soll an dieser Stelle zu 30 fps bei VGA-Bildauflösung (640×480 px) definiert werden.

Ein Überblick über die CPU-Ausführungszeiten aus der Literatur für die verschiedenen algorithmischen Schritte innerhalb des SIFT-Algorithmus ist in Tab. 3.1 gegeben. Es wer-

**Tabelle 3.1:** Vergleich CPU-Ausführungszeiten einzelner Teile im SIFT-Algorithmus

| Algorithmischer Abschnitt | [Men16][1] | [Hua12][2] | [Hua12][3] |
|---|---|---|---|
| Aufbau des Skalenraums | 77,34 s (80,67 %) | 2,1 s (72,99 %) | 2281,89 s (85,74 %) |
| Merkmalsdetektion | 1,60 s (1,67 %) | 0,16 s (5,71 %) | 9,08 s (0,33 %) |
| Orientierungsberechnung | 2,44 s (2,55 %) | 0,08 s (2,85 %) | 57,56 s (2,16 %) |
| Deskriptorberechnung | 14,49 s (15,11 %) | 0,52 s (18,57 %) | 311,80 s (11,69 %) |
| Total | 95,87 s (100,00 %) | 2,87 s (100,00 %) | 2660,66 s (100,00 %) |

[1] Tensilica Xtensa LX5 (Basisprozessor - 45nm TSMC) @ 625 MHz, 800 × 640 px
[2] Intel Core 2 Duo @ 2,09 GHz, 640 × 480 px
[3] Altera NIOS II (DE-70 FPGA Board) @ 100 MHz, 640 × 480 px

den Softwareimplementierungen auf zwei Prozessoren für eingebettete Systeme (Tensilica Xtensa LX5 [Men16] und Altera NIOS II [Hua12]) mit einer Softwareimplementierung auf einer General-Purpose-CPU (Intel Core 2 Duo [Hua12]) verglichen. Der Aufbau des Skalenraums ist für alle drei Prozessoren mit Abstand der aufwändigste Schritt, da hier neben der massiven Anzahl an Operationen zur Berechnung der Gauß-Pyramide auch die erforderliche Menge an Speicher für die Zwischenbilder am größten ist. Die Deskriptorberechnung ist mit einigem Abstand der zweitaufwändigste Schritt, da eine Vielzahl an trigonometrischen Funktionen erforderlich ist. Darüber hinaus ist die Gesamtausführungszeit für alle drei Prozessoren weit von der Echtzeitanforderung entfernt. Insbesondere für die beiden Embedded-Prozessoren ist eine signifikante Beschleunigung unumgänglich.

Ein Architekturansatz ist die Erweiterung eines Prozessors um applikationsspezifische Instruktionen (A*pplication* S*pecific* I*nstruction-Set* P*rocessor*), was von den Autoren von [Men16] untersucht wurde. Dort wurde ein Tensilica Xtensa LX5-Prozessor verwendet und mit Instruktionssatzerweiterungen (u. a. einer FIR-Filtereinheit, Wurzel- und Sinus-Einheiten, Beschleuniger zur Histogrammerstellung) zur Verbesserung der Ausführungsgeschwindigkeit ausgestattet. Es konnte ein Speed-up von 125× bei einer Erhöhung der Gatteranzahl (45 nm TSMC) um den Faktor 5× im Vergleich zur Basiskonfiguration des Prozessors erzielt werden. Trotz des hohen Hardwareaufwands betrug der Durchsatz nur 0,8 fps bei einer Bildauflösung von 800×640 px. Dies zeigt, dass ein Ansatz mit reiner Instruktionssatzerweiterung nicht ausreichend ist, um den SIFT-Algorithmus echtzeitfähig zu implementieren. Lediglich durch einem Einsatz von komplexeren dedizierten Beschleunigern kann die erforderliche Performance-Steigerung erreicht werden.

Da der SIFT-Algorithmus strukturell aus mehreren aufeinander aufbauenden, aber in sich gut parallelisierbaren Teilschritten besteht, sind FPGAs als parallele, energieeffiziente Plattformen für die Implementierung gut geeignet. Daher lässt sich in der Literatur eine Vielzahl von dedizierten Lösungen zum Einsatz von SIFT auf FPGAs finden. Im Allgemeinen ist der Durchsatz von Bildmerkmalsextraktoren zu einem gewissen Grad abhängig von hohen Taktfrequenzen und Speicherbandbreiten, da die Bilder in dedizierten Architekturen oft pixelweise durch Streamprocessing verarbeitet werden und bei skalenraumbasierten Algorithmen viele temporäre Zwischenergebnisse entstehen. Dies sind zwei Aspekte, in

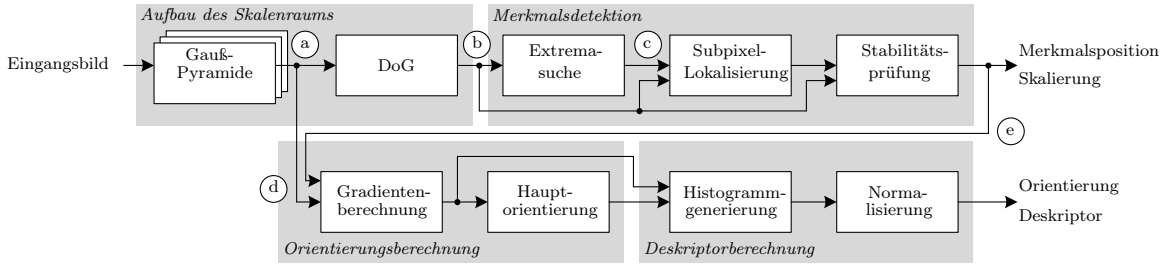



**Abbildung 3.1:** Blockschaltbild des SIFT-Algorithmus

denen FPGAs durch die vorgegebenen Strukturen und daraus resultierenden Routingverzögerungen grundsätzlich eher limitiert sind [Awa16]. Daher müssen FPGA-Architekturen versuchen, diese Schwierigkeiten durch gezielte algorithmische Vereinfachungen, effiziente Pipeline-Strukturen und eine geschickte Nutzung von Datenlokalitäten bzw. Mehrfachnutzung von Daten zu überwinden. Im Folgenden soll in Abschnitt 3.1.1 zunächst ein struktureller Überblick über mögliche Architekturkonzepte für den SIFT-Algorithmus gegeben werden. Dabei werden der Datenfluss und Abtauschmöglichkeiten beschrieben, die sich an verschiedenen Positionen in der Architektur ergeben. Diesem Überblick liegen die Ergebnisse einer umfangreichen Literaturrecherche zugrunde. Im Anschluss werden in Abschnitt 3.1.2 wesentliche in der Literatur diskutierte Architekturen vorgestellt. Abschließend wird in Abschnitt 3.1.3 die Notwendigkeit tiefergehender Untersuchungen einer FPGA-Architektur für den SIFT-Algorithmus dargelegt.

## 3.1.1 Struktureller Überblick

### Gesamtsystem

In Abb. 3.1 sind die algorithmischen Blöcke des SIFT-Algorithmus grafisch dargestellt. Für die Hardware-Abbildung lassen sich neben sequentiellen auch parallele Strukturen finden.

Das Eingangsbild wird in dedizierten Architekturen zur Bildverarbeitung zumeist im Ganzen oder in Blöcken entlang der Bildzeilen pixelweise in die Verarbeitungskette geleitet (*Streaming*-Verarbeitung). Am Anfang steht hier die Gauß-Pyramide zur Erstellung des approximierten Skalenraums. Diese besteht aus einer Reihe von FIR-Filtern mit unterschiedlich vielen Taps und liefert mehrere Einzelbilder, die jeweils unterschiedlich stark Gauß-gefilterte Versionen des Eingangsbildes sind. Aufgrund seiner Komplexität wird dieser Block in einem eigenen Abschnitt weiter unten beschrieben.

Zur Merkmalsdetektion werden danach an Position Abb. 3.1(a) die Einzelbilder der Pyramide an den DoG-Block geleitet, der aus mehreren Subtrahierern besteht. Die Einzelbilder müssen dabei pixelweise parallel und zeitlich synchron anliegen. Da die FIR-Filter in der Pyramide unterschiedliche Latenzen aufweisen, sind hier Schieberegister bzw. ganze Zeilenspeicher notwendig. Die Extremasuche durchsucht alle 3×3×3 großen

Bildausschnitte innerhalb der DoG-Bilder, also sind an Position Abb. 3.1(b) wieder Zeilenspeicher notwendig, um einen Bildausschnitt gleichzeitig anliegen zu haben. Insgesamt kann die Extremasuche auf verschiedenen Skalierungen parallel erfolgen. Die gefundenen Extremstellen an Position Abb. 3.1(c) werden *Merkmalskandidaten* genannt.

Die Subpixel-Lokalisierung der Kandidaten benötigt im Anschluss einen noch größeren Ausschnitt aus der DoG-Pyramide, um die Extremstelle bei Bedarf um Pixelpositionen zu verschieben. Der letzte Schritt der Detektion ist die Stabilitätsprüfung, in der entschieden wird, ob ein Merkmalskandidat als *stabil* akzeptiert oder verworfen wird. Sowohl Subpixel-Lokalisierung als auch Stabilitätsprüfung erfordern Divisionsoperationen. Als Ergebnis stehen subpixelgenaue Merkmalspositionen und Skalierungen der SIFT-Merkmale zur Verfügung.

Zur Berechnung der Orientierung und zur Deskriptorberechnung werden die Bildgradienten in einem Bildbereich um das Bildmerkmal benötigt. Laut Referenzalgorithmus werden diese auf demjenigen Skalenbild berechnet, das der ermittelten Merkmalsskalierung am nächsten liegt (Abb. 3.1(d)). Die Berechnung von Betrag und Winkel der Bildgradienten erfordern eine Auswertung der Quadratwurzelfunktion sowie Divisions- und Arcustangens-Operationen. Danach folgt die Berechnung der Hauptorientierung. Der Bildbereich zur Histogrammgenerierung wird an dieser Orientierung ausgerichtet, um die Rotationsinvarianz des Deskriptors zu erreichen. Mittels trilinearer Interpolation werden die Gradienten auf mehrere Bins im Histogramm verteilt und dort aufakkumuliert. Die Normalisierung des Histogramms zur Verringerung von Effekten, die durch Gradientenspitzen auftreten, ist der letzte Schritt der Deskriptorberechnung.

In der Darstellung wird der kritische Pfad innerhalb der Datenabhängigkeiten deutlich. Die Orientierungs- und Deskriptorberechnungen sind nach dem Referenzalgorithmus auf die Merkmalsextraktion angewiesen. Zum einen wird die Position benötigt, um den Deskriptor im richtigen Bildausschnitt zu bestimmen. Zum anderen wird das zu verwendende Gradientenbild anhand der Skalierung gewählt. Diese Verbindung ist durch Abb. 3.1(e) gekennzeichnet. In einer realen Hardware-Implementierung auf einer parallelen Plattform wie einem FPGA ist dies unvorteilhaft, da die Daten der Gauß-Pyramide während der Dauer der Merkmalsdetektion in einem Speicher vorgehalten werden müssen. Währenddessen befindet sich die Deskriptorhardware im Leerlauf. Nach der Detektion wird der entsprechende Teil der Pyramide zur Orientierungs- und Deskriptorberechnung weitergeleitet und die Detektorhardware pausiert. Dieses Vorgehen ist ineffizient und erhöht zudem die Gesamtlatenz, was aufgrund der sequentiellen Abhängigkeit einen direkten Einfluss auf den Durchsatz der Architektur hat. Darüber hinaus ergibt sich ein *merkmalsabhängiger Durchsatz* abhängig von der Anzahl der stabilen Merkmale, da nur für diejenigen Merkmale ein Deskriptor berechnet werden muss, die der Stabilitätsprüfung standhalten. Es können nun verschiedene Designentscheidungen getroffen werden, um die genannten Probleme zu lösen. Dabei sind bei Umgestaltungen im Datenpfad, die eine Verringerung der Komplexität oder eine Veränderung zum Referenzalgorithmus darstellen, immer auch Abschläge in der algorithmischen Qualität zu berücksichtigen.

In einer Variante werden die Gradienten auf dem kompletten Bild berechnet und

im Speicher abgelegt. Dies geschieht entweder direkt auf dem Eingangsbild oder auf einem Skalenbild mit einer zuvor gewählten, konstanten Skalierung. Dadurch muss zur Bestimmung der Gradienten nicht auf die exakte Merkmalsskalierung gewartet und die Pyramide im Ganzen gespeichert werden. Die Problematik bleibt, dass sich Teile der Architektur zeitweise im Leerlauf befinden. Es bietet sich daher an, die Pfade für Detektion und Deskription parallel arbeiten zu lassen und für jedes Pixel einen Deskriptor zu berechnen, der im Zweifel am Ende verworfen wird. Da beide Pfade im Allgemeinen unterschiedliche Latenzen aufweisen, sind Synchronisationsregister erforderlich. Auch hier stellt sich die Frage nach der Effizienz, wenn Rechenergebnisse verworfen werden müssen. Auch eine verschränkte Berechnung (Task-Level Pipelining), bei der Detektor- und Deskriptorhardware gleichzeitig auf verschiedenen Teilbildern arbeiten, bzw. der Deskriptor immer zum Merkmal aus dem Zeitschritt davor berechnet wird, ist möglich, jedoch auch hier nur im Fall einer positiven Stabilitätsprüfung.

Viele dedizierte Implementierungen verzichten im Detektorpfad auf algorithmische Schritte wie z. B. die Subpixel-Lokalisierung, da der Schritt algorithmisch viele Kontrollflussoperationen erfordert, die sehr aufwändig in Hardware abzubilden sind. Auch Vereinfachungen des Orientierungs- und Deskriptor-Pfades können vorgenommen werden, wie z. B. das Weglassen der Orientierungsberechnung unter Aufgabe der Rotationsinvarianz oder eine vereinfachte Normalisierung, um die Komplexität der Hardware zu reduzieren. Die Verschiebung von Teilen der Berechnung auf einen Prozessor (z. B. DSP) ist in der Literatur ebenfalls zu finden. Dabei handelt es sich meist um eine Auslagerung des Deskriptors. Dadurch sinkt zwar die Größe der dedizierten Hardware, aber die Komplexität in einem solchen heterogenen System steigt, und es müssen Konzepte für Synchronisation und Datenübergabe gefunden werden.

### Gauß-Pyramide

Der Aufbau der Gauß-Pyramide wurde bereits in Abschnitt 2.2.4 kurz beschrieben und soll hier um Implementierungsdetails ergänzt werden, da die Gauß-Pyramide in dedizierter Hardware auf verschiedene Weise implementiert werden kann [Nik14]. Zum Aufbau des Skalenraums mit den Skalenbildern $L(x, y, \sigma)$ für ein Eingangsbild $I(x, y)$ wird eine Reihe von Faltungen mit verschiedenen Gauß-Kerneln mit anwachsender Standardabweichung $\sigma$ durchgeführt (Gauß-Filter).

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{3.1}$$

mit

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}. \tag{3.2}$$

Die Standardabweichung berechnet sich dabei zu

$$\sigma_i = \sqrt{(k^i\sigma_0)^2 - \sigma_a^2} \qquad \forall\, i > 0 \tag{3.3}$$

mit der Schrittweite $k = 2^{1/s}$, der Aufnahmeunschärfe $\sigma_a = 0{,}5$ und der Standardabweichung nach der ersten Filterung von $\sigma_0 = 1{,}6$ [Low04b]. Durch die derartige Festlegung von $k$ verdoppelt sich die Unschärfe innerhalb von $s$ Skalenbildern, was in Anlehnung an die Verdopplung der Schwingfrequenz in der musiktheoretischen Intervalllehre als Oktave bezeichnet wird. Die Faltung zweier Gauß-Funktionen ist wieder eine Gauß-Funktion und die resultierende Standardabweichung lässt sich durch

$$\sigma_{f*g} = \sqrt{\sigma_f^2 + \sigma_g^2} \tag{3.4}$$

berechnen. Im Anschluss werden nebeneinander liegende Skalenbilder voneinander subtrahiert (DoG-Bilder):

$$D(x, y, \sigma_i) = L(x, y, \sigma_i) - L(x, y, \sigma_{i-1}) \tag{3.5}$$

Heutige FPGAs sind darauf optimiert, eine 2D-Faltung eines Bildes mit einer Gauß-Funktion mittels der FIR-Filterstruktur [Pir98] auf den DSP-Slices zu implementieren [Xil11a]. Dabei weisen Gauß-Filter die Eigenschaft der Separierbarkeit auf [Mit06]. Anstatt einer 2D-Faltung werden dabei zwei 1D-Faltungen (vertikal und horizontal) durchgeführt, wodurch bei einem $N \times N$ großen Filterkernel die Anzahl der Multiplikationen von $N^2$ auf $2N$ und die Anzahl der Additionen von $N^2 - 1$ auf $2(N - 1)$ reduziert werden können. Es ist dabei möglich, zuerst eine vertikale und dann eine horizontale Filterung durchzuführen oder umgekehrt. Die Eingangsbilder liegen meist zeilenweise im Speicher und werden daher auch zeilenweise in einem Pixelstrom angeliefert. Wird mit der horizontalen Filterung begonnen, sind vor der vertikalen Filterung Zeilenspeicher notwendig, die durch Fixpunkt-Arithmetik eine erhöhte Wortbreite je Pixel erfordern. Wird dagegen mit der vertikalen Filterung begonnen, so sind diese Zeilenspeicher aufgrund der zeilenweisen Bereitstellung der Pixel bereits vor dem Filter mit lediglich der Eingangswortbreite der Pixel notwendig. Zusätzlich lassen sich in dieser Organisation die internen Register der DSP-Slices im horizontalen Filter besser nutzen, wodurch eine Abbildung auf dem FPGA günstiger[1] [Xil11a] und somit der transponierten Filterung mit beginnendem horizontalen Filter vorzuziehen ist.

Zur Abschätzung der Filterordnung eines Gauß-Kernels lässt sich folgende Faustformel für den Kerneldurchmesser in Pixeln heranziehen [Men18]:

$$N_{Taps} = 2\lceil 4\sigma \rceil + 1 \tag{3.6}$$

Zur Realisierung einer Filterung mit einem Gauß-Kernel mit höherer Standardabweichung muss also ein größeres Filter mit mehr Taps auf das FPGA abgebildet werden.

Die Folge von Skalenbildern kann grundsätzlich mittels dreier verschiedener Ansätze berechnet werden (siehe Abb. 3.2) [Nik14]. Der erste Ansatz ist eine *sequentielle* Berechnung

[1] *Günstiger* heißt in diesem Zusammenhang mit weniger Ressourcen, höherem maximalen Takt und größerer Energieeffizienz aufgrund der Nutzung von dedizierten Strukturen innerhalb der DSP-Slices

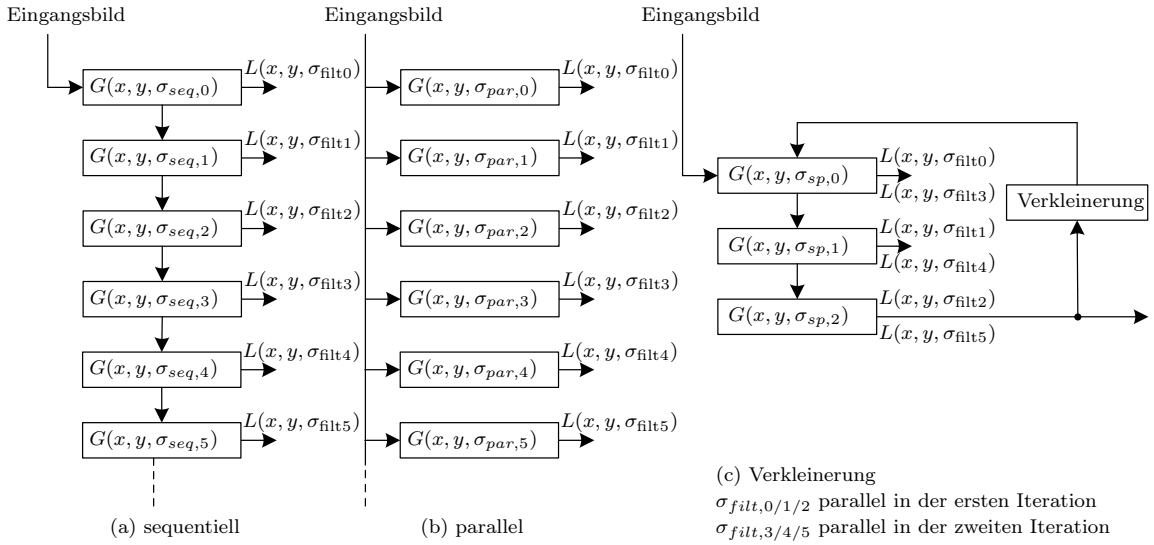



**Abbildung 3.2:** Verschiedene Architektur-Ansätze zur Berechnung der Skalenbilder [Nik14]

(Abb. 3.2(a)), d. h. das jeweils nächste Skalenbild wird auf Basis des vorangegangenen berechnet. Da das Eingangsbild pixelweise die Gauß-Filter durchläuft und die Skalenbilder zur Berechnung der DoG-Bilder parallel anliegen sollten, muss in diesem Ansatz auf das Ergebnis einer vorangegangenen Gauß-Filterung gewartet werden und die Daten müssen währenddessen mit Zeilenspeichern zwischengespeichert werden. Dieses Vorgehen hat eine größere Latenz als die alternative *parallele* Berechnung (Abb. 3.2(b)). Dabei werden alle Gauß-Filter parallel auf dem Eingangsbild ausgeführt. Es entfallen die Zeilenspeicher, lediglich Synchronisationsregister zum Ausgleich unterschiedlicher Filtergrößen sind notwendig. Allerdings werden hier die Filter aufgrund insgesamt höherer, notwendiger Standardabweichungen größer.

Die dritte Alternative beinhaltet eine *Verkleinerung* des gefilterten Bildes nach einer (oder mehrerer) Oktave(n), also Verdoppelung der Standardabweichung (Abb. 3.2(c)). Dies bietet den Vorteil der parallelen Filterung innerhalb einer Oktave, einer Limitierung der Filtergrößen und einer Wiederverwendung der Hardware, da die Filterkoeffizienten gleichbleibend sind. Bezüglich der Latenz liegt dieser Ansatz zwischen der sequentiellen und parallelen Berechnungsweise.

Ein exemplarischer Aufbau der Gauß-Pyramide ist in Abb. 3.3 dargestellt. Es sind die Struktur und die quantitativen Standardabweichungen für sechs Gauß-Filter und drei Oktaven mit $\sigma_a = 0{,}5$ und $k = 2^{1/3}$ gezeigt (vgl. Gl. 3.3). Die Standardabweichungen der Filter ergeben sich aus Gl. 3.4, wobei die gepunktete Linie einem sequentiellen ($\sigma_{seq}$ in Klammern) und die durchgezogene Linie einem parallelen ($\sigma_{par}$ ohne Klammern) Filterdatenpfad nach Abb. 3.2 entsprechen. $\sigma_{\text{filter}}$ verdoppelt sich alle drei Filter. Durch Verkleinerung des Bildes nach der ersten Oktave bleiben die Filterkoeffizienten gleich

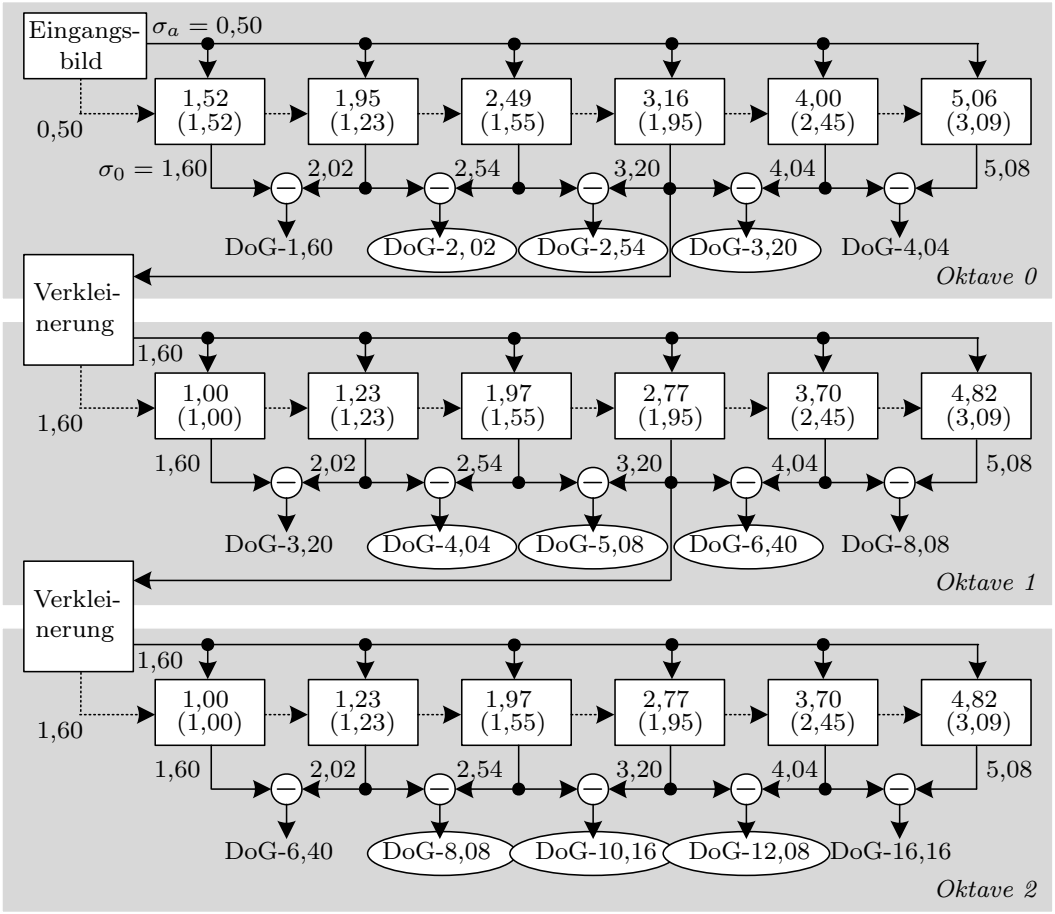



**Abbildung 3.3:** Exemplarische Struktur einer Gauß-Pyramide. Die Zahlen entsprechen den Standardabweichungen der Filter bzw. den angenommenen Unschärfen im jeweiligen Bild. Weitere Erläuterungen im Text.

und erlauben eine Wiederverwendung der Filter für alle weiteren Oktaven. Nach der Faustformel Gl. 3.6 ergibt sich für den größten Filter eine Ordnung von 43. Da die äußeren Filterkoeffizienten nahe 0 sind, kann hier in Wechselwirkung mit der resultierenden algorithmischen Qualität optimiert werden. In den Ellipsen sind diejenigen DoG-Bilder angegeben, die zusammen mit den jeweiligen Bildern links und rechts davon nach Extrema, d. h. Merkmalskandidaten, durchsucht werden (z.B. die Bilder DoG-1,60/2,02/2,54).

Die Anzahl der Oktaven ist abhängig von der Größe des Eingangsbildes und lässt sich nach folgender Formel bestimmen:

$$N_{oct} = \lfloor \log_2 (\min(W_{img}, H_{img})) - (\log_2 (L_{img,min}) - 1) \rfloor \tag{3.7}$$

Dabei sind $W_{img}$ und $H_{img}$ Bildbreite bzw. Bildhöhe des Eingangsbildes und $L_{img,min}$ die minimale Seitenlänge eines Skalenbildes. Für ein Bild mit einer Auflösung von 640×480 px und minimal 32 px Seitenlänge pro Skalenbild ergeben sich vier Oktaven.

Der Speicherbedarf der gesamten Gauß-Pyramide kann mittels der unendlichen geometrischen Reihe nach oben hin abgeschätzt werden, da der Beitrag der kleineren Bilder auf den höheren Oktaven minimal ist. Aufgrund der sukzessiven Bildverkleinerung, bei der jede zweite Reihe und jedes zweite Pixel verworfen wird, ergibt sich

$$MEM_{bit} = \alpha \sum_{k=0}^{\infty} \left(\frac{1}{4}\right)^k = \alpha \cdot \frac{4}{3} \tag{3.8}$$

$$\text{mit} \qquad \alpha = N_{scalesPerOct} \cdot N_{bitPerFilteredPixel} \cdot W_{img} \cdot H_{img}. \tag{3.9}$$

Für ein Bild mit einer Auflösung von 640×480 px, sechs Skalenbildern pro Oktave und 16 bit pro gefiltertem Pixel (Fixpunkt-Darstellung bei 8 bit breiten Eingangspixeln) resultiert daraus ein Speicherbedarf von etwa 4,7 MB. Für 1920×1080 px wächst dieser auf etwa 31,6 MB an. Das entspricht einem Vergrößerungsfaktor von 16× im Vergleich zum Eingangsbild. Werden die DoG-Bilder noch hinzugezählt, erhöht sich der Bedarf entsprechend auf einen Gesamtfaktor von mehr als 29×. Zum Vergleich steht auf einem kompletten Xilinx Virtex-6 XC6VLX240T FPGA, wie er auf einem Xilinx ML605 Entwicklungsboard verbaut ist, nur eine Gesamtmenge von 3,6 MB distributed und 14,6 MB Block RAM zur Verfügung [Xil15]. Für eine FPGA-Implementierung wird dadurch deutlich, dass eine Architektur gefunden werden muss, bei der es während der Prozessierung nicht erforderlich ist, die gesamte Pyramide vorzuhalten.

### 3.1.2 Literaturrecherche

Im Folgenden wird ein Überblick über in der Literatur erschienene dedizierte Architekturen für den SIFT-Algorithmus gegeben. Ein quantitativer Vergleich ist in Abschnitt 3.1.3 in Form einer Tabelle aufgeführt (Tab. 3.2).

Eine der ersten Implementierungen von SIFT auf einem FPGA in der Literatur ist [Se04], worin ein Stereosystem für Roboternavigation beschrieben wird. Eine angepasste Version des Algorithmus wird dort mittels High-Level-Synthese auf einen Xilinx Virtex-II FPGA abgebildet. Details zur Architektur werden allerdings nicht gegeben, da eher das Gesamtsystem im Vordergrund steht.

[Pet05] lagern die Gauß-Pyramide und einen Sobelfilter zur Gradientenberechnung auf einen Xilinx Virtex-II FPGA aus. Der restliche Algorithmus wird auf einem Host-PC berechnet. Die Pixel werden dabei im Stream in eine Kaskade von Gauß-Filtern geleitet. Mithilfe von Zeilenspeichern sind die verschiedenen Skalenbilder gleichzeitig zur Berechnung der DoG-Bilder sowie zur Extremasuche verfügbar. Die 2D-Gauß-Filter werden als separierte Filter implementiert, wobei zuerst horizontal und danach vertikal gefiltert wird. Die Filterkernel haben dabei lediglich eine Breite von fünf Taps, was im Vergleich zur verwendeten Standardabweichung gering erscheint.

[Kim07] präsentieren eine NoC-basierte ASIC Prozessorarchitektur zur Objekterkennung mit SIFT als Merkmalsextraktor, allerdings ohne Stabilitätsprüfung. Die Berechnung

erfolgt blockweise auf Subbildern und die Gauß-Pyramide wird mit Spezialinstruktionen beschleunigt.

In [Cha07] wird ein Konzept für ein Hardware-Software Co-Design vorgestellt. Dabei operieren ein Prozessor (Deskriptor) und ein dedizierter Beschleuniger (Detektor) in verschiedenen Clock-Domains jeweils auf Teilen des Algorithmus. Der Instruktionssatz des Prozessors wird dabei applikationsspezifisch erweitert. Welche Instruktionen dies sind und wie der Datenaustausch zwischen den Komponenten bzw. die Art und Weise der Berechnungen abläuft, bleibt jedoch unklar.

Eine der ersten vollständigen und viel zitierten Implementierungen des SIFT-Algorithmus auf FPGAs wurde in [Bon08] detailliert präsentiert. Die Autoren haben den Detektor als parallelen Hardwarebeschleuniger implementiert, während der Deskriptor in einem über einen FIFO angeschlossen Softcore-Prozessor berechnet wird, da die erforderlichen Berechnungen leichter in Software umzusetzen sind. Um ihren Beschleuniger im Durchsatz zu balancieren und einen Gesamtdurchsatz von 30 fps auf QVGA-Bildern zu erreichen, führen sie eine Analyse der notwendigen Parallelitätsgrade durch, worin der erforderliche Durchsatz- und Parallelitätsfaktor der Einzelmodule bestimmt wird. Der Bottleneck ist der Softcore-Prozessor, welcher 11,7 ms zur Deskriptorberechnung benötigt, was drei Deskriptoren bei 30 fps entspricht. Die Berechnung von Betrag und Phase der Gradienten erfolgt parallel zur Detektion unter Verwendung des CORDIC-Algorithmus.

In der Arbeit von [Yao09] wird neben einer dedizierten Architektur zur Merkmalsdetektion ein eigener, in der Komplexität reduzierter Deskriptor beschrieben, welcher auf einem Softcore berechnet wird. Die Gauß-Pyramide wird dabei blockweise und mit parallelen Filtern berechnet. Zur Bestimmung der Gradientenbeträge wird die Manhattan-Distanz (L1-Norm) als Alternative zur euklidischen Norm verwendet. Die abschließende Evaluation bezieht sich nur auf den Detektor. Eine Folgearbeit ist [Den12], die eine Hardwareimplementierung des Deskriptors im Fokus hat.

Um den Hardwareaufwand der parallelen Gauß-Filter zu verringern, nutzen [Qiu10] die Assoziativität der Faltung aus und approximieren ein großes Gauß-Filter durch Hintereinanderausführung von zwei kleineren Filtern. Da manche Filterkernel dabei identisch sind, können Zwischenergebnisse wiederverwendet werden.

[Lin10] präsentiert eine dedizierte Implementierung des SIFT-Deskriptors für ASIC, die alle algorithmischen Schritte enthält.

Eine weitere Architektur für FPGA und ASIC ist in [Miz10; Miz11] beschrieben. Die Verarbeitung erfolgt blockweise und der Datenpfad ist stark gepipelinet, wodurch zwei Ausführungsmodi ermöglicht werden: ein Präzisionsmodus und ein High-Speed Modus. Die höhere Präzision wird dadurch erreicht, dass auf größeren Blöcken gerechnet wird, was aber einen höheren Speicherbedarf und einen niedrigeren Durchsatz zur Folge hat.

[Hua12] konzentrieren sich in ihrer ASIC-Architektur auf eine Reduktion des benötigten Speichers durch segmentweise Berechnungen und spezielle SRAM-Strukturen zur Wiederverwendung von Daten. Die Komponenten zur Detektion und Deskription arbeiten im Wechsel. Das Deskriptormodul wartet zunächst auf ein Merkmal. Wurde ein Merkmal detektiert, wartet das Detektormodul während der Bestimmung des Deskriptors. Die

Berechnung der Gauß-Filter erfolgt parallel mit Kernelgrößen von 9 bis 25. Zur Berechnung von mehreren Oktaven werden die Filter dupliziert und nacheinander mit Daten versorgt. Die Bestimmung der Gradientenbeträge und der -orientierung erfolgt parallel zur Merkmalslokalisierung im Detektormodul.

Die Autoren von [Chi13] stellen eine veränderte Berechnung der Gauß-Pyramide vor, bei denen die Skalierungen parallel berechnet werden. Die Gauß-Filter werden durch vorherige Berechnung eines Integralbilds und Box-Filter approximiert [Sze11], was zur Reduzierung des Hardwareaufwands aufgrund der anwachsenden Filtergrößen gegenüber der parallelen Berechnung führt. Außerdem wird ein alternativer Test zur Berechnung der Merkmalsstabilität basierend auf Helligkeitswerten vorgestellt, sodass keine Taylor-Reihenentwicklung durchgeführt werden muss.

In [Cha13] wird eine Architektur für den Detektor vorgestellt, welche dieselben Hardwareressourcen zur Gauß-Filterberechnung für mehrere Oktaven wiederverwendet, um die Anzahl an berechneten Oktaven zu erhöhen und gleichzeitig Ressourcen zu sparen. Dabei werden pro Skalenfilter die Einzelbilder für die verschiedenen Oktaven in einer verschränkten Art und Weise mittels Time-Division-Multiplexing berechnet, sodass alle Oktavenbilder parallel vorliegen. Dabei wird der Takt halbiert und die Berechnung der höheren Oktaven auf jeden zweiten Takt aufgeteilt. Die Berechnung der Skalierungen selbst erfolgt sequentiell in einer Filterkaskade. Durch den beschriebenen Ansatz kann die SIFT-Detektion auf vier Oktaven mit jeweils fünf Skalenbildern in nur 1,1 ms für ein QVGA-Bild (320x240 Pixel) durchgeführt werden. Werte zu höheren Auflösungen werden leider nicht gegeben. Subpixel-Lokalisierung, Stabilitätsprüfung, sowie Orientierungs- und Deskriptorberechnung erfolgen in diesem System in Software auf einem Host-PC.

In [Zho13] kombinieren die Autoren einen FPGA-basierten Beschleuniger zur Merkmalsdetektion mit einem TI DSP zur Deskriptorberechnung. Obwohl der Detektor eine Bildrate von 100 fps für QVGA-Bilder erreichen würde, ist der Durchsatz durch die Deskriptorberechnung, die 80 $\mu$s pro Merkmal benötigt, beschränkt, sodass das Gesamtsystem mit 50 fps für 250 Deskriptoren arbeitet.

Auch in [Kim13] steht die Speicherreduktion im Fokus. Die Berechnungen erfolgen blockweise, und die Gauß-Filterbank ist parallel aufgebaut, wobei zuerst vertikal und dann horizontal gefiltert wird. Die Filterkernel haben dabei Größen von sieben bis 19 Taps. Die Filter selbst werden für alle Oktaven wiederverwendet, wobei aber die Oktaven nacheinander berechnet werden. Zur Berechnung der Gradienten wird ein konstantes Bild aus der Pyramide zwischengespeichert, und sobald aus der Detektion die Merkmalsskalierung bekannt ist, sorgt ein nachgeschaltetes Gauß-Filter für eine nachträgliche Anpassung der Skalierung.

Unter Verwendung von Double-Buffering haben die Autoren von [Jia14] eine auf Tasklevel parallele Hardwarestruktur veröffentlicht, worin sowohl Detektor als auch Deskriptor parallel auf verschiedenen Speicherblöcken arbeiten können. Darüber hinaus stellen sie einen neuartigen Ansatz zur Deskriptorrotation vor, ohne eine explizite Rotation des Berechnungsfensters wie in der Referenz durchzuführen.

In [Suz12; Len16] wird der SIFT-Detektor durch einen Harris-Eckendetektor ersetzt,

während trotzdem der SIFT-Deskriptor verwendet wird. Dabei wird die Skalierungsinvarianz durch den skalenraumbasierten Ansatz aufgegeben, was aber für die jeweilige Anwendung ausreichend sein kann. In [Wan14] wird der SIFT-Detektor mit einem BRIEF-Deskriptor unter Aufgabe der Rotationsinvarianz kombiniert.

Im Kontext der reversiblen Logik wird in [Pal15] SIFT als exemplarische Anwendung verwendet, um neuartige Addiererstrukturen auf einem FPGA zu entwickeln.

[Qas15] implementieren die Gauß-Pyramide mit parallelen Filtern und vollständig ohne Multiplizierer, um Hardwareressourcen zu sparen. Dabei wird der Multiplierless Multiple Constant Multiplication (MCM) Algorithmus verwendet. Darüber hinaus wird zur Berechnung der trilinearen Interpolation im Deskriptor ein Konzept vorgestellt, um die Anzahl an notwendigen Lese- und Schreib-Ports zum Histogrammspeicher zu reduzieren.

Die Autoren von [Vou16] präsentieren ein neuartiges Architekturkonzept, bei dem in einer komplett gepipelineten Architektur in jedem Takt für jedes Pixel ein Deskriptor berechnet wird, wodurch der Gesamtdurchsatz unabhängig von der Anzahl an Deskriptoren ist. Da nur ein Bruchteil der Pixel tatsächlich stabile Merkmale darstellt, werden die meisten Deskriptoren verworfen. Aus Sicht der Autoren wird auf eine separierte Implementierung der Gauß-Filter verzichtet, weil bei der Berechnung erheblicher Hardwareaufwand getrieben werden müsste, um keine Genauigkeit zu verlieren. Sie optimieren daher die 2D-Faltung, indem kleine Koeffizienten gestrichen werden, durch Ausnutzung von Symmetrien und durch Ersetzen von großen Multiplikationen durch eine Kombination aus kleinen Multiplikationen und Shift-Add Operationen. Der Deskriptor wird im Vergleich zur Referenz stark vereinfacht. Auf eine Drehung und die trilineare Interpolation wird verzichtet, und die ursprüngliche Histogramm-Normalisierung wird durch eine einfache L1-Norm ersetzt.

[Yum16] reduzieren den benötigten internen Speicher durch Auslagerung in externen Speicher, geschickte Wiederverwendung von Bildbereichen unter Verwendung von lokalen Buffern und Subsampling der Gauß-Bilder vor dem Speichern, da durch die Gauß-Filterung keine Aliasing-Artefakte zu erwarten sind. Zur Reduktion der externen Speicherbandbreite werden Nachkommabits abgeschnitten und der algorithmische Einfluss bewertet.

[Pen16] stellen ein Gesamtsystem auf einem SoC inklusive Kameraanbindung vor, auf dem der SIFT-Algorithmus als IP-Core eingesetzt wird. Der CORDIC-Algorithmus wird in ausgerollter Form zur Orientierungsberechnung verwendet.

In [Zho16] wird eine Architektur bestehend aus einem Detektor- und einem Deskriptormodul beschrieben, welche parallel und mit unterschiedlichen Takten arbeiten können. Der Detektor beinhaltet parallele Gauß-Filter, in denen zwei Oktavenbilder einer Skalierung in verschränkter Art und Weise durch die gleiche Filterhardware geschoben werden. Die Verbindung der beiden Module erfolgt durch einen FIFO und ein Speicherarray. In diesem Konzept können Merkmale verloren gehen, wenn die Berechnungen der Orientierung und des Deskriptors nicht ausreichend schnell erfolgt. Zur Verringerung der Merkmalsverluste wird das Deskriptormodul schneller getaktet als das Detektormodul. Orientierungs- und Deskriptorberechnung können parallel arbeiten, da unterschiedliche

Bildbereiche gleichzeitig in Buffern vorgehalten werden. Der Durchsatz der Architektur wird durch die Laufzeit des Detektormoduls bestimmt.

[Dom20] arbeitet mit einem bildübergreifenden Double-Buffering-Konzept für die Merkmalsdetektion. Damit der Deskriptor parallel zur Detektion berechnet werden kann, müssen ein ganzes Einzelbild und dessen zugehörige Merkmale zwischengespeichert werden. Zur Berechnung der Gradientenbeträge wird die L1-Norm verwendet. Die Winkel werden mit einem LUT-basierten Ansatz bestimmt. Die Autoren spezifizieren, dass das System für 1% der Pixel des Eingangsbildes bei VGA-Auflösung den Deskriptor in Echtzeit berechnen können soll. In ihrem Ansatz liegt damit der Bottleneck in der Deskriptorberechnung, und es ist möglich, im Detektor durch sequentielle Berechnung der Gauß-Filter Hardwareressourcen zu sparen.

Eine spezialisierte Architektur ausschließlich für den Schritt der Subpixel-Lokalisierung ist in [Rub18] dargestellt. Die Lokalisierung erfolgt in der SIFT-Referenz in einer Schleife und kann Merkmalskandidaten um mehrere Pixel verschieben. Diese Schleife wird hier abgerollt und nach einer Untersuchung auf zwei Iterationen begrenzt, da nur eine verschwindend kleine Anzahl der Merkmalskandidaten weitere Iterationen benötigt.

In [Fej19] wird eine Architektur für den Detektor ohne Subpixel-Lokalisierung oder Stabilitätsprüfung mittels High-Level-Synthese generiert und auf einem FPGA als Teil eines SoCs implementiert. Der FPGA dient dabei als Beschleuniger für einen ARM Cortex-A9-Prozessor, der die restlichen Teile des Algorithmus berechnet.

### 3.1.3 Diskussion

Durch den vorangegangenen Abschnitt wird die große Anzahl der bisherigen in der Literatur erschienenen, dedizierten Architekturen für den SIFT-Algorithmus deutlich. Zum einen durch individuelle Beiträge zu den verschiedenen algorithmischen Komponenten und zum anderen durch die Komplexität des Algorithmus mit vielen kleinen Teilschritten ist eine Klassifizierung der Architekturen schwierig. Es lassen sich ASIC- und FPGA-Architekturen unterscheiden, wobei im Fall von ASICs größere Flexibilität hinsichtlich Speicherressourcen besteht und höhere Taktfrequenzen erreicht werden können, was aber im Vergleich nicht unmittelbar zu erkennbar höheren Bildraten führt (siehe Tab. 3.2). Außerdem lassen sich homogene und heterogene Systeme unterscheiden, in denen ein dedizierter Beschleuniger in Kombination mit einem eingebetteten Prozessor arbeitet.

Tab. 3.2 stellt die genannten Architekturen quantitativ gegenüber, ähnlich der Darstellung in [Vou16]. Dabei unterscheiden diese sich stark in Bildauflösung, Durchsatz und Anzahl an unterstützten Merkmalen, was zwar einen direkten Vergleich erschwert, dabei aber die Grenzen des Entwurfsraums deutlich abbildet. Es wird im Allgemeinen erwartet, dass etwa 1% der Pixel im Bild Merkmale sind [Bon08]. Als echtzeitfähige Bildverarbeitung gilt oft ein Durchsatz von 30 fps bei 640×480 px (VGA-Auflösung). Außerdem sind in der Tabelle Detektionszeit pro Bild $t_{detect}$ und Deskriptionszeit pro Merkmal $t_{execPerDesc}$ angegeben. Abhängig von der Architektur lässt sich daraus der Gesamtdurchsatz und die Anzahl an zu verarbeitenden Merkmalen ableiten. Mit Ausnahme

von in [Vou16], bei denen sich der Durchsatz durch Multiplikation der Taktperiode mit der Anzahl an Pixeln im Bild ergibt, kann der Durchsatz (Bildrate) vereinfacht durch folgende Formel berechnet werden:

$$R_{img} = \frac{1}{t_{img}} \qquad \text{mit} \qquad t_{img} = \max(t_{detect}, t_{execPerDesc} \cdot N_{Merkmale}) \tag{3.10}$$

Entweder wurde dabei von den Autoren ein Zieldurchsatz definiert und die Anzahl an Merkmalen pro Bild leitet sich daraus ab oder es gibt eine erwartete Anzahl an Merkmalen und die Bildrate ergibt sich daraus. Die Detektion läuft dabei im besten Fall parallel ab, d. h. dass sie entweder den Gesamtdurchsatz bestimmt oder von der Ausführungszeit für die Deskriptorberechnung überdeckt wird.

Ein effizientes System muss zu den fortlaufend generierten Merkmalen schritthaltend einen Deskriptor berechnen können. Ist dies nicht der Fall, müssen Merkmale verworfen (Beeinträchtigung der auf SIFT aufbauenden Algorithmen), die Detektion pausiert (sinkender Durchsatz, Hardware im Leerlauf), oder größere Pufferspeicher vorgesehen werden. Ein zu schneller Detektionsteil ist nicht von Vorteil, wenn dieser auf die Bereitschaft des Deskriptionsteils warten muss. Daher können Ressourcen und Durchsatz für den Detektionsteil abgetauscht werden, z. B. durch einen geringeren Parallelitätsgrad bei der Berechnung der Gauß-Pyramide. Der Detektionsteil sollte immer nur so schnell wie nötig arbeiten, damit der Deskriptionsteil voll ausgelastet ist. Gleiches gilt auch umgekehrt für eine zu schnelle Berechnung des Deskriptors. Unterschiedliche Taktfrequenzen im System sind dabei möglich.

Um die Architekturen in Tab. 3.2 besser vergleichen zu können, wird hier eine Normierung der Detektionszeit auf VGA-Auflösung und eine einzige Oktave durchgeführt:

$$t_{detect,norm} = \left\lceil t_{detect} \cdot \frac{640 \cdot 480}{W_{img} \cdot H_{img}} \cdot \kappa_{oct} \right\rceil \tag{3.11}$$

$$\text{mit} \qquad \kappa_{oct} = \begin{cases} 1 & \text{wenn } N_{oct} = 1 \\ \frac{1}{1+\frac{1}{4}} = 0,8 & \text{wenn } N_{oct} = 2 \\ \frac{1}{1+\frac{1}{4}+\frac{1}{16}+\ldots} = 0,75 & \text{wenn } N_{oct} \geq 2 \end{cases} \tag{3.12}$$

$\kappa$ ist hier ein Korrekturfaktor abhängig von der Anzahl Oktaven. Der Wert für $N_{oct} \geq 2$ ergibt sich aus dem Grenzwert der geometrischen Reihe (vgl. Gl. 3.8). Zur Normierung wird ein linearer Zusammenhang zwischen Oktavenbildgröße und Rechenzeit unterstellt. Die Werte sind in Tab. 3.2 in Klammern angegeben. Verschiedene Aspekte sind bei der Normierung zu berücksichtigen. Den Feinheiten einzelner Architekturen (z. B. verschränkte Oktavenberechnung) wird durch die Normierung nicht Rechnung getragen. Zudem skalieren Architekturen im Allgemeinen nicht linear und nicht identisch entlang der Bildbreite und -höhe. Es ergeben sich aufgrund der zeilenweisen Verarbeitung Wartezyklen beim Zeilenwechsel, z. B. in Filtern, sodass die Normierung Architekturen bevorzugt, die für wenig Bildzeilen ausgelegt sind. Außerdem sind die Seitenverhältnisse der verschiedenen Auflösungen unterschiedlich. Auch der Durchsatz wurde zur besseren Vergleichbarkeit

auf VGA-Bildauflösung normiert. Da den Durchsatz meist der Deskriptionsteil bestimmt, wird auf die Normierung auf Oktaven an dieser Stelle verzichtet. Der normierte Durchsatz ergibt sich zu:

$$R_{img,norm} = \left\lfloor R_{img} \cdot \frac{W_{img} \cdot H_{img}}{640 \cdot 480} \right\rfloor . \tag{3.13}$$

### Fazit

Die präsentierten Architekturen erweitern den Stand der Forschung jeweils um individuelle Beiträge, weisen dabei aber alle die gleiche Charakteristik auf, die sich aus dem SIFT-Algorithmus ergibt. Dieser besteht aus zwei vom Berechnungsmuster her signifikant unterschiedlichen Teilen, welche im Laufe dieser Diskussion weiter oben vereinfacht als Detektionsteil und Deskriptionsteil beschrieben wurden (vgl. auch Abb. 3.1) und hier noch einmal verallgemeinert betrachtet werden sollen.

Zum einen gibt es den *Pixelteil*, der ein Eingangsbild auf Pixelebene verarbeitet und Merkmalskandidaten mit einer *deterministischen*, nur von der Bildauflösung abhängigen Laufzeit hervorbringt. Zu diesem Teil zählen der Aufbau des Skalenraums, die Extremasuche und die Gradientenberechnung, sofern sie für jedes Pixel durchgeführt wird. Zum anderen gibt es den *Objektteil*, der die Merkmalskandidaten im Subpixelbereich lokalisiert bzw. verschiebt, sie auf Stabilität hin prüft und im Fall einer positiven Prüfung den zugehörigen Deskriptor bestimmt. Dieser Teil ist *datenabhängig*, d. h. abhängig vom Bildinhalt. Die Merkmalskandidaten sind dabei im Bild nicht gleichverteilt, und ihre Anzahl verändert sich, sodass die Rechenlast für den Objektteil variiert.

Der Pixelteil ist dabei von der Struktur her voll parallelisierbar und somit gut mit dem höchsten Durchsatz in dedizierter Hardware zu implementieren. Der Objektteil hingegen bietet zwar in Teilen bzw. innerhalb seiner Einzelschritte Datenparallelitäten, da auch hier weiterhin Berechnungen auf Pixelebene durchzuführen sind, ist aber naturgemäß auf weiter zu verarbeitende Objektdaten angewiesen und daher inhärent sequentiell zum Pixelteil. Dies limitiert den Gesamtdurchsatz.

Insgesamt müssen die Berechnungen des Pixelteils und des Objektteils als Gesamtsystem in Balance stehen, damit das System effizient ist. Dazu sollten die beiden Rechenzeiten in Gl. 3.10 gleich sein, was bei veränderlicher, eingangsdatenabhängiger Anzahl an Merkmalskandidaten nicht mehr gegeben ist. Wartezeiten der Komponenten aufeinander müssen minimiert werden, um maximalen Durchsatz und Effizienz zu erreichen. Insbesondere die Energieeffizienz profitiert von einem balancierten System. Es müssen Konzepte zum Datenaustausch, zur Synchronisation und zur variierenden Rechenlast im Objektteil gefunden werden, um ein umfassend optimiertes System zu erhalten.

Eine Untersuchung zum Zusammenhang zwischen Systembalance, Durchsatz, Merkmalsanzahl und Energieeffizienz einer dedizierten Architektur für den SIFT-Algorithmus ist in der Literatur nicht zu finden. Die Ergebnisse einer tiefergehenden Analyse dazu wären auch übertragbar auf eine Vielzahl anderer Algorithmen, die eine gleiche Charakteristik von Pixel- und Objektteil aufweisen.

**Tabelle 3.2:** Überblick und quantitativer Vergleich dedizierter Architekturen für den SIFT-Algorithmus. Werte in Klammern geben die normierte Detektionszeit nach Gl. 3.11 bzw. die normierte Bildrate nach Gl. 3.13 an.

| | Plattform | Bild-auflösung (Pixel) | Detektor/ Deskriptor | Oktaven/ Skalen-bilder | Detektions-zeit (norm.) (ms) | Deskriptions-zeit pro Merkmal ($\mu$s) | Anzahl Merkmale | System-frequenz (MHz) | Bildrate (norm.) (fps) | Verlust-leistung (W) |
|---|---|---|---|---|---|---|---|---|---|---|
| [Se04] | Virtex-II | 640×480 | SIFT / SIFT | - | - | - | - | - | 17 | - |
| [Pet05] | Virtex-II | - | SIFT / SIFT | 4/4 | - | - | - | 54 | 60 (-) | - |
| [Kim07] | ASIC (180 nm) | 320×240 | SIFT / SIFT | - | - | - | - | 200-400 | 10-16 (2-4) | 1,4 |
| [Cha07] | Virtex-II+Proz. | 320×240 | SIFT / SIFT | - | - | - | - | 100 | 1250 (312) | - |
| [Bon08] | Stratix-II+Nios-II | 320×240 | SIFT / SIFT | 3/6 | 33 (99) | 11.700 | $< 3$ | 50/100 | 30 (7) | - |
| [Yao09] | Virtex-5+MicroBlaze | 640×480 | SIFT / - | 2/4 | 31 (25) | - | - | 100 | 32 | - |
| [Miz10] | Cyclone-II | 640×480 | SIFT / - | 2/4 | 17,8 (15) | - | - | 50 | 32/56 | - |
| [Lin10] | ASIC (130 nm) | - | - / SIFT | - | - | 15,3 | - | 200 | - | - |
| [Miz11] | ASIC (65 nm) | 1920×1080 | SIFT / SIFT | 2/4 | - | - | - | 78 | 30 (202) | 0,038 |
| [Hua12] | ASIC (180 nm) | 640×480 | SIFT / SIFT | 3/6 | 3,4 (3) | 33,1 | <890 | 100 | 30 | - |
| [Den12] | Virtex-6 | 640×480 | - / SIFT | - | - | 7,57 | <2.200 | 100 | 60 | - |
| [Suz12] | Virtex-5 | 1920×1080 | Harris / SIFT | - | - | 16 | <1.024 | 168 | 60 (405) | - |
| [Chi13] | Virtex-6 | 640×480 | SIFT / SIFT | 2/4 | - | 5,5 | $< 6.000$ | 100 | 30 | - |
| [Cha13] | Virtex-II | 320×240 | SIFT / - | 4/5 | 1,1 (4) | - | - | 145 | 900 (225) | - |
| [Zho13] | Virtex-4+TI DSP | 320×256 | SIFT / SIFT | 3/6 | 10 (29) | 80 | <250 | 106 | 50 (13) | 8,35 |
| [Kim13] | Stratix-II | 320×240 | SIFT / SIFT | 3/6 | - | 60 | <550 | 50 | 30 (7) | |
| [Jia14] | Virtex-5 | 512×512 | SIFT / SIFT | 2/4 | 6,55 (7) | 2,23 | <2.900 | 100 | 150 (128) | 3,9 |
| [Wan14] | Virtex-5 | 1280×720 | SIFT / BRIEF | 2/6 | 8 (3) | 8,3 | <2.000 | 75 | 60 (180) | 4,5 |
| [Pal15] | Virtex-5 | 640×480 | SIFT / SIFT | 2/5 | 15 (12) | - | - | 100 | 64 | - |
| [Men16] | Xtensa LX5 (45 nm) | 800×640 | SIFT / SIFT | 7/5 | 643 (290) | 2.200 | <256 | 400 | 0,8 (1,3) | - |
| [Qas15] | Virtex-5 | 640×480 | SIFT / SIFT | 3/6 | 6,14 (5) | 25,9 | <1.270 | 50 | 30 | - |
| [Vou16] | Cyclone-IV | 640×480 | SIFT / SIFT | 1/4 | 6,8 (7) | 0,046 | $\infty$ | 21,7 | 70 | $< 1$ |
| [Yum16] | ASIC (130 nm) | 1280×720 | SIFT / SIFT | 3/6 | - | - | <2.816 | 170 | 37 (111) | 0,128 |
| [Pen16] | Xilinx ZC702 | 1920×1080 | SIFT / SIFT | 9/6 | - | - | <1.102 | - | 30 (202) | - |
| [Zho16] | Virtex-7 | 1280×720 | SIFT / SIFT | 2/6 | 23,8 (7) | 20 | <1.148 | 50/100 | 42 (126) | - |
| [Dom20] | Virtex-5 | 640×480 | SIFT / SIFT | 1/6 | 3,3 (4) | 3,3 | <3.072 | 100 | 99 | - |
| [Fej19] | TUL PYNQ-Z2 | 1920×1080 | SIFT / - | 1/3 | - | - | - | 200 | 96 (648) | 0,274 |

## 3.2 Partitionierung und Anpassungen des Algorithmus

Auf Basis der Literaturrecherche wurde für die im Rahmen dieser Arbeit entwickelte FPGA-Architektur ein heterogener Ansatz ausgewählt. Die dahingehende Partitionierung des Algorithmus ist in Abb. 3.4 dargestellt (vgl. Abb. 3.1). Die Designkriterien, die zur Auswahl geführt haben, sind zum einen eine Maximierung des Durchsatzes und zum anderen die Möglichkeit einer differenzierten Untersuchung zur Systembalance zwischen Pixelteil und Objektteil bzw. des Abtauschs zwischen Durchsatz, Merkmalsanzahl und Energieeffizienz, wie am Ende des vorangegangenen Abschnitts beschrieben. Die Berechnung der Gauß-Pyramide, der DoG-Bilder und die Extremasuche (Pixelteil) erfolgt auf einem dedizierten Detektionsbeschleuniger (engl. *SIFT Detection Engine*, welcher im Folgenden kurz *SDE* genannt wird), der die Eingangsbilder pixelweise verarbeitet und Merkmalskandidaten sowie ein Pixelfenster zur Deskriptorberechnung bereitstellt. Subpixel-Lokalisierung, Stabilitätsprüfung und die Berechnung des Deskriptors (Objektteil) erfolgt dabei auf einem *applikationsspezifischen VLIW-Prozessor* [Pay12]. Die Schnittstelle zwischen den beiden Komponenten ist in Abb. 3.4(a) gekennzeichnet.

Wie auch in der Literatur wurden zur Implementierung des Algorithmus auf dem FPGA Anpassungen im Vergleich zur SIFT-Referenz durchgeführt. Diese sind im Folgenden aufgelistet. Der Einfluss dieser Anpassungen auf die algorithmische Qualität wird in Abschnitt 3.7 charakterisiert.

- Eine Untersuchung auf einem Referenzdatensatz [Hou13] hat gezeigt, dass etwa 86% der Merkmalskandidaten während der Subpixel-Lokalisierung ihre Position nicht verändern, d. h. nicht verschoben werden. 12% werden um einen Pixel und nur 2% werden um mehr als einen Pixel verschoben. Die Skalierung bleibt bei 97% der Kandidaten auf dem gleichen Skalenbild [Nik14]. Daher wird nur eine Iteration der Subpixel-Lokalisierung durchgeführt, in der die Pixelposition um maximal einen Pixel in x- und y-Richtung plus Subpixel-Offset und in der Skalierung gar nicht verschoben werden kann. In der Literatur ist in [Rub18] eine ähnliche Untersuchung zu finden und die Iterationen werden auf maximal zwei begrenzt. Diese Anpassung verringert die Komplexität der Berechnung auf dem VLIW-Prozessor und die Menge an DoG-Bilddaten, die vom SDE zum VLIW-Prozessor übertragen werden müssen.

- Mit Blick auf die Literatur und Anwendungen aus dem Fahrerassistenzbereich, in denen bei ausreichend hoher Bildrate nur geringe perspektivische Änderungen zwischen den Einzelbildern zu erwarten sind, wurde auf die Berechnung einer Merkmalsorientierung verzichtet. Dadurch entfällt die Notwendigkeit einer präzisen Winkelbestimmung der Gradienten und einer Drehung des Pixelfensters, in dem der Deskriptor bestimmt wird.

- Gemäß der Referenz wird der Deskriptor auf demjenigen Skalenbild berechnet, das der detektierten Merkmalsskalierung am nächsten liegt. Um Speicherplatz zu sparen,

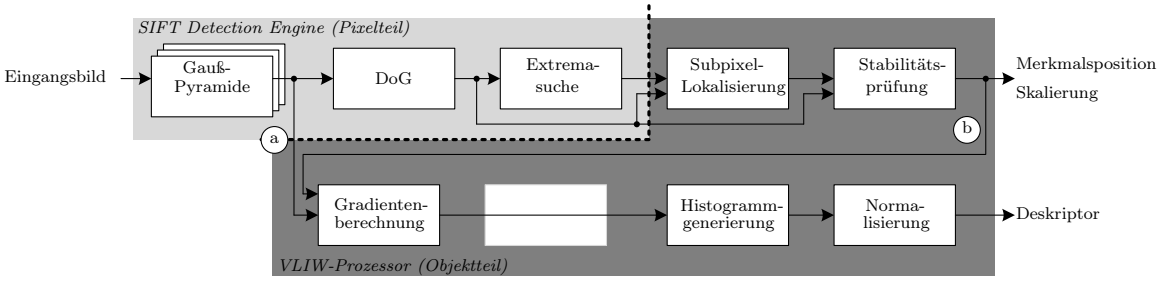



**Abbildung 3.4:** Heterogene Partitionierung des SIFT-Algorithmus

wird ein 20×20 px großes Fenster von einem festen Skalenbild zum VLIW-Prozessor weitergegeben, welches auf den Merkmalskandidaten zentriert ist. Dieses erlaubt eine Verschiebung der Merkmalsposition aufgrund der Subpixel-Lokalisierung um einen Pixel und wird zur Gradientenberechnung verwendet. Die Datenabhängigkeit in Abb. 3.4(b) liegt somit komplett in der Software. Erst nach der Stabilitätsprüfung wird das Pixelfenster weiterverarbeitet oder verworfen.

- Der Deskriptor wurde in der Komplexität stark reduziert und orientiert sich an der Implementierung in [Vou16]. Ohne Merkmalsorientierung verliert der Deskriptor seine Rotationsinvarianz. Durch diesen Ansatz ergeben sich im Deskriptor nur acht Winkelbins, in die die Gradienten einsortiert werden müssen. Diese lassen sich aus Verhältnis und Vorzeichen der x- und y-Komponenten der Gradienten bestimmen. Trigonometrische Funktionen sind nicht erforderlich. Zur Berechnung der Gradientenbeträge wird die L1-Norm verwendet. Die Gauß-Gewichtung der Gradienten abhängig von ihrem Abstand zur Merkmalsposition wird nicht durchgeführt, ebenso wie die trilineare Interpolation bei der Histogrammgenerierung. Zur Normalisierung werden die Einzeleinträge im Histogramm durch die Summe aller Einträge geteilt. Ein Clipping und eine zweite Normalisierung finden danach nicht mehr statt.

Ein Objekt, das zwischen dem SDE und dem VLIW-Prozessor an Position Abb. 3.4(a) ausgetauscht wird, besteht aus:

- *x- und y-Position* des Merkmalskandidaten
- *Oktave* des Merkmalskandidaten
- *Skalierung* des Merkmalskandidaten
- *5×5×3 px große Pixelumgebung* aus der DoG-Pyramide um den Merkmalskandidaten herum (16 bit pro Pixel). Anhand dieser Daten kann im Schritt der Subpixel-Lokalisierung eine Verschiebung des Merkmals um jeweils einen Pixel in x- und y-Richtung erfolgen, sowie die Stabilitätsprüfung durchgeführt werden. Der Merkmalskandidat wurde zuvor im SDE durch Vergleich der inneren 3×3×3 px dieser Umgebung detektiert, die ein lokales Extremum darstellen.

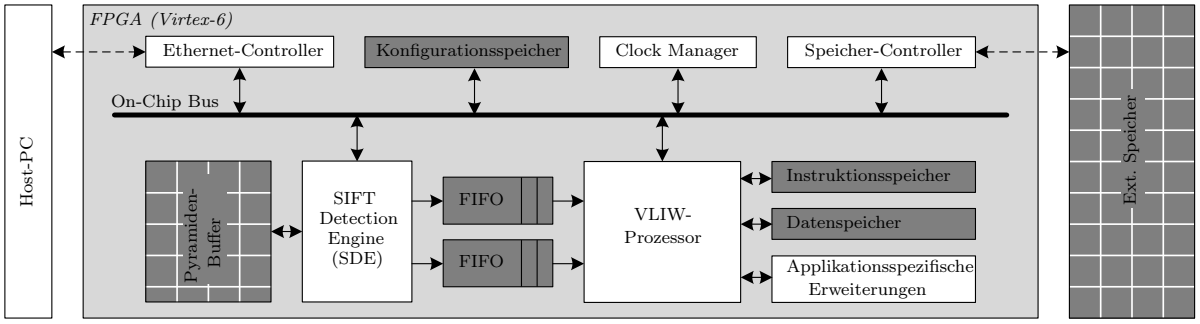



**Abbildung 3.5:** Überblick über das FPGA-basierte heterogene Gesamtsystem zur SIFT-Bildmerkmalsextraktion

- *20×20 px großes Pixelfenster* aus der Gauß-Pyramide eines festen Skalenbildes (8 bit pro Pixel). Dieses Fenster dient im VLIW-Prozessor zur Berechnung der Gradienten und Bildung des Deskriptorhistogramms.

## 3.3 Systemüberblick

Ein Überblick über das Gesamtsystem ist in Abb. 3.5 dargestellt.

Die Zielplattform ist das ML605 Entwicklungsboard von Xilinx [Xil11b], welches einen Xilinx Virtex-6 FPGA (XC6VLX240T-1 / 40 nm) enthält. Das hier verwendete SoC-Emulationsframework [Koc14] beinhaltet einen Ethernet-Controller zur Kommunikation mit einem Host-PC, einen Speicher-Controller zum externen SDRAM auf dem Board und einen Systembus (OCP). Der Host-PC ist verantwortlich für die globale Steuerung und hat Zugriff auf den internen Konfigurationsspeicher. Da das System keine eigene Kamera besitzt, werden die Bilddaten vom Host-PC aus in den externen Speicher auf dem Board geschrieben. Durch Verwendung von Double-Buffering können transferbedingte Wartezeiten zwischen zwei Einzelbildern innerhalb einer Bildsequenz vermieden werden. Ein konfigurierbarer Clock-Manager stellt auf dem FPGA den Takt für die einzelnen Systemkomponenten bereit. Xilinx ermöglicht dabei auch eine dynamische Rekonfiguration des Clock-Managers zur Laufzeit des Systems [Xil10].

Ein dedizierter Detektionsbeschleuniger (engl. *SIFT Detection Engine*, im Folgenden kurz *SDE* genannt) streamt das Eingangsbild pixelweise durch seinen gepipelineten Datenpfad. Die Daten werden dazu mittels eines eigenen Bus-Masters aus dem externen Speicher angefordert. Die Aufgaben des SDE sind die rechenintensiven Schritte des Algorithmus auf Pixelebene, d. h. die Berechnung der Gauß-Pyramide, die Erstellung der DoG-Bilder und die Extremasuche. Ein angeschlossener Pyramiden-Buffer ist in der Lage, ein Bild mit einem Viertel der Größe des Eingangsbildes zu speichern, wodurch ab der zweiten Oktave auf lokale Daten zurückgegriffen werden kann.

Über FIFO-Speicher werden Daten zu einem *VLIW-Prozessor* [Pay12] übertragen. Dieser führt Subpixel-Lokalisierung, Stabilitätsprüfung und die Berechnung des De-

skriptors durch. Zur beschleunigten Berechnung dieser Schritte wurde der Prozessor um applikationsspezifische Instruktionen und Koprozessoren erweitert. Der Prozessor verfügt über separate Instruktions- und Datenspeicher und einen eigenen DMA. Da der VLIW-Prozessor für Multimedia-Anwendungen konzipiert ist, für die zumeist bekannt ist, welche Daten geladen werden müssen, wurde ein DMA einem Cache-System vorgezogen.

Das System in seinen Grundzügen mit einer Untersuchung zur Balancierung für die SIFT-Merkmalsdetektion wurde in [Har17b] publiziert.

## 3.4 SIFT Detection Engine (SDE)

Der SIFT Detection Engine (SDE) ist ein dedizierter Detektionsbeschleuniger zur Extraktion von Merkmalskandidaten. Eine detaillierte Übersicht über den internen Aufbau des Datenpfades ist in Abb. 3.9 dargestellt.

Die Steuerung des SDE geschieht vom VLIW-Prozessor aus. Bildgröße, Anzahl zu berechnender Oktaven, Speicheradressen und Performance-Counter (z. B. Laufzeit der SDE, Wartezeit usw.) werden in das Konfigurations-Registerfile geschrieben und der SDE Pipeline-Controller wird gestartet. Der Input-Controller liest die Bilddaten entweder in der ersten Oktave über den Bus-Master aus dem externen Speicher oder ab der zweiten Oktave direkt aus dem Pyramiden-Buffer und füllt pixelweise, d. h. mit einem Pixel pro Takt, die Verarbeitungspipeline. Dabei wird von 8 bit breiten Eingangspixeln ausgegangen. Der Pyramiden-Buffer hat eine Kapazität von einem Viertel des spezifizierten Eingangsbildes. Die Anbindung an den VLIW erfolgt über FIFO-Speicher, wobei der VLIW dabei entweder über ein 64-bit oder ein 128-bit Interface verfügt. Sollten die FIFOs vollständig mit Merkmalskandidaten gefüllt sein, wird die Pipeline angehalten, sodass keine Merkmale verloren gehen können. Außerdem wird die Pipeline gestoppt, wenn am Eingang keine gültigen Pixel anliegen oder die Merkmalsextraktion auf dem aktuellen Bild abgeschlossen ist. Der Stop der Pipeline kann dabei entweder durch Nutzung von Enable-Logik an den Flip-Flops erfolgen oder direkt durch Deaktivieren des Clockbuffers (Clock Gating).

Der SDE ist für Taktfrequenzen bis zu 200 MHz ausgelegt. Zum Bus hin liegt die Grenze der Clock-Domain im Bus-Master, da der Bus mit 100 MHz arbeitet. Die FIFOs bzw. das Konfigurations-Registerfile stellen das Clock-Domain-Crossing zum VLIW bereit. Dabei kann der SDE sowohl schneller als auch langsamer als der VLIW getaktet werden. Die unterstützte maximale Bildgröße ist ein generischer Syntheseparameter, welcher einen großen Einfluss auf die genutzten Block-RAM-Ressourcen hat, da die Zeilenspeicher und der Pyramiden-Buffer mit den Bildabmessungen skalieren.

### 3.4.1 Filterstruktur der Gauß-Pyramide

Die erste Verarbeitungsstufe innerhalb der Pipeline ist die Berechnung der Gauß-Pyramide. Eine Diskussion zu Implementierungsalternativen wurde bereits in Abschnitt 3.1.1 ab Seite 33 geführt.

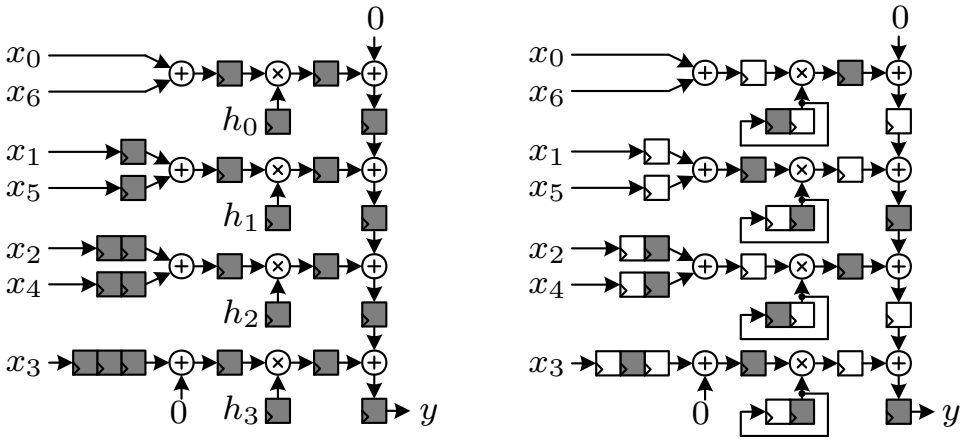
Die im SDE implementierte Pyramide besteht aus sechs parallelen 2D-Gauß-Filtern zur Erzeugung von fünf DoG-Bildern pro Oktave (vgl. Abb. 3.3). Durch den parallelen Aufbau sind alle Skalenbilder einer Oktave gleichzeitig in der DoG-Stufe zur Subtraktion verfügbar und es können Merkmalskandidaten für drei Skalierungen parallel ermittelt werden. Zur Bildverkleinerung wird jeder zweite Pixel jeder zweiten Reihe am Filterausgang des vierten Bildes im Pyramiden-Buffer gespeichert und vor der Berechnung der nächsten Oktaven wieder von dort gelesen. Alle Oktaven verwenden nacheinander dieselbe Filterhardware. Eine Oktavenbegrenzung gibt es in dieser Implementierung daher nicht. Die sich ergebenden Standardabweichungen der Filter sind in Abb. 3.3 aufgelistet. Die notwendigen Filtergrößen bei der Verwendung von Gauß-verteilten Filterfunktionen lassen sich mit Gl. 3.6 abschätzen und liegen in dieser Implementierung bei 13×13, 17×17, 21×21, 25×25, 33×33 und 41×41. Kleinere Filter würden die Filterfunktion an den Rändern abschneiden, was auf der einen Seite Hardwarekosten reduzieren, allerdings auf der anderen Seite eine algorithmische Approximation darstellen würde.

Die 2D-Gauß-Filter sind symmetrisch und separierbar, sodass hier zuerst eine vertikale (Abb. 3.6a) und anschließend eine horizontale (Abb. 3.7a) 1D-FIR-Filterung durchgeführt wird. 40 Zeilenspeicher puffern die Bilddaten für die sechs parallelen vertikalen Filter, wobei das größte Filter hier die Größe des bereitzustellenden vertikalen Bildstreifens bestimmt. Am Bildrand wird das äußerste Pixel repliziert. Das horizontale Filter kann ohne separate Pufferung direkt mit dem vertikalen verbunden werden. Aufgrund der Symmetrie der Filterfunktion lassen sich mithilfe von vorgeschalteten Addierern Multiplikationsoperationen einsparen. Die Filter sind zum Erreichen der maximalen Taktfrequenz stark gepipelinet. Die Implementierung erfolgt in DSP-Slices auf dem FPGA [Xil11a], die die Register in dedizierter Hardware bereitstellen. Die gestaffelten Register am Eingang des vertikalen Filters werden in Logic Slices implementiert. Durch die tiefe Pipeline in den Filtern und die unterschiedliche Anzahl an Taps ergeben sich auch unterschiedliche Latenzen an den Filterausgängen. Damit die Subtraktion der Skalenbilder parallel erfolgen kann, sind zur Synchronisation Schieberegister notwendig.

Die Filter-Implementierung erfordert den Einsatz von Fixpunkt-Arithmetik. Die 8 bit Eingangspixel (8 bit Ganzzahl, keine Nachkommabits, d. h. 8.0) werden mit auf 16 bit (0.16) quantisierten Filterkoeffizienten multipliziert. Nach dem vertikalen Filter (8.16) erfolgt ein Shift (8.8) und die horizontale Filterung. An deren Ausgang (8.24) wird erneut geshiftet (8.8). Ein Überlauf des maximalen Wertebereichs kann nicht auftreten, da die Filterkoeffizienten in Summe 1 ergeben.

### Time-Division-Multiplexing der Filterhardware (TDM-Modus)

Die DSP-Slices des FPGAs können laut Datenblatt [Xil11a] für den hier verwendeten FPGA mit Speed Grade -1 mit bis zu 450 MHz getaktet werden. Dies entspricht einem Vielfachen der maximalen Taktfrequenz des restlichen Teils des Beschleunigers, welche bei 200 MHz liegt. Aufgrund dessen werden hier die Vor- und Nachteile eines Time-Division-Multiplexing (TDM-Modus) der Filterhardware untersucht.

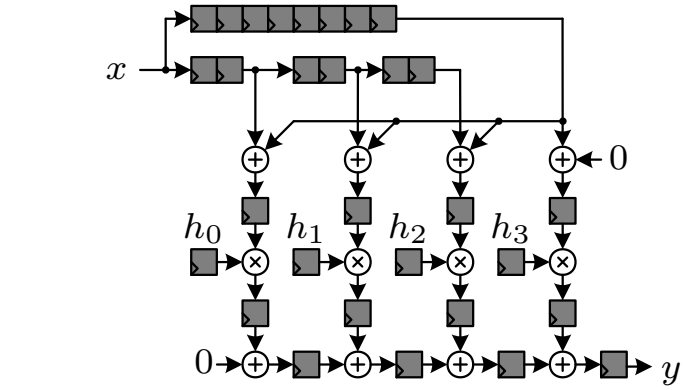



**(a)** kein TDM **(b)** TDM×2

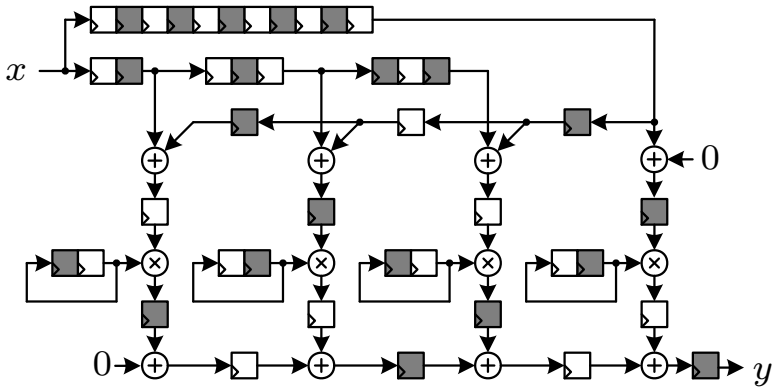
**Abbildung 3.6:** (a) 1D-FIR-Filter zur vertikalen Bildfilterung mit sieben Taps und vorgeschalteten Addierern aufgrund der symmetrischen Filterfunktion. Die Eingangspixel $x_0$ bis $x_6$ liegen gleichzeitig an und ändern gleichzeitig ihren Wert. (b) Durch Verdoppelung des internen Filtertakts aller dargestellten Register und zyklisches Schieben der Koeffizienten kann das Filter im Time-Division-Multiplexing (TDM) betrieben werden. Da die Filterhardware geteilt wird, muss für gleichen Durchsatz die Filtertaktfrequenz im Vergleich zur umliegenden Hardware um den TDM-Faktor ansteigen. Die zwei Farben der Register sollen die zwei Phasen des Filters andeuten.

Insgesamt werden drei TDM-Modi betrachtet: TDM×2, TDM×3 und TDM×6. Der sich dadurch ergebende reduzierte Hardwareaufwand ist schematisch in Abb. 3.8 dargestellt. Da die Filterhardware geteilt wird, muss für gleichen Durchsatz die Filtertaktfrequenz im Vergleich zur umliegenden Hardware um den TDM-Faktor ansteigen. Indem die Filter entweder doppelt, dreifach oder sechsfach so schnell wie der restliche Beschleuniger getaktet werden, sind nur drei, zwei oder ein Filter notwendig, im Vergleich zu sechs ohne TDM. Die Hardwareersparnis skaliert dabei nicht linear, da beim Zusammenführen mehrerer Filter jeweils der größte Filter erhalten bleiben muss.

Abb. 3.6b zeigt TDM×2 für die vertikale Filterung. Der interne Filtertakt aller dargestellten Register wird verdoppelt. Die Eingangssignale $x_0$ bis $x_6$ variieren immer noch mit demselben, langsameren Takt der umliegenden Schaltungsteile. In Schieberegistern werden die Koeffizienten zyklisch verschoben. Auf diese Weise werden dieselben Eingangsdaten mit zwei unterschiedlichen Koeffizienten multipliziert. Es ergeben sich somit zwei Phasen, die durch unterschiedliche Farben der Register in der Abbildung angedeutet sind. Abgesehen von den zyklischen Schieberegistern für die Koeffizienten ergibt sich bei der vertikalen Filterung kein zusätzlicher Hardwareaufwand durch den TDM-Modus. Anders ist dies bei der horizontalen Filterung, welche in Abb. 3.7b für TDM×2 dargestellt ist. Es sind zusätzliche Register notwendig, um die Datensynchronität in den zwei Phasen




**(a)** kein TDM




**(b)** TDM×2

**Abbildung 3.7:** (a) 1D-FIR-Filter zur horizontalen Bildfilterung mit sieben Taps und vorgeschalteten Addierern aufgrund der symmetrischen Filterfunktion. (b) Durch Verdoppelung des internen Filtertakts aller dargestellten Register und zyklisches Schieben der Koeffizienten kann das Filter im Time-Division-Multiplexing (TDM) betrieben werden. Die zwei Farben der Register sollen die zwei Phasen des Filters andeuten.

zu halten. Aufgrund dessen lässt sich diese Struktur nicht mehr so effizient auf einem DSP-Slice abbilden, da diese auf die in 3.7a gezeigte Struktur mit zwei Registern am Eingang optimiert sind. Blockschaltbilder zum Aufbau der vertikalen und horizontalen Filter für TDM×3 und TDM×6 sind im Anhang in Abb. A.2 und Abb. A.3 dargestellt.

Die maximale Taktfrequenz der Filter wird hier auf 400 MHz (2,5 ns) begrenzt, wodurch sich eine Reduktion des Maximaltakts des gesamten SDE für TDM×3 (400 MHz/3 = 133 MHz < 200 MHz) und für TDM×6 (400 MHz/6 = 66 MHz < 200 MHz) ergibt. Daher sinkt in diesen Fällen der maximal erreichbare Durchsatz des SDE.

Durch Mehrfachverwendung der DSP-Slices beim TDM-Modus werden insgesamt weniger Slices benötigt. Allerdings sind teilweise zusätzliche Schieberegister notwendig. Die dynamische Verlustleistung der Filter aufgrund des lokal höheren Takts steigt,

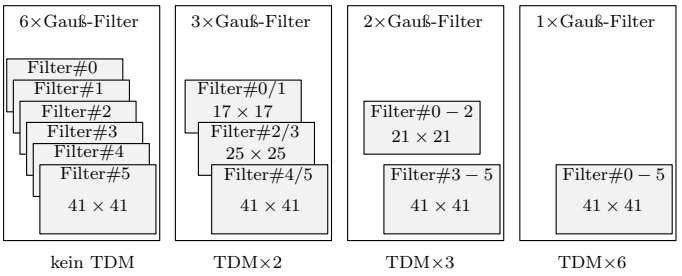



**Abbildung 3.8:** Schematische Darstellung des Time-Division-Multiplexing (TDM) der Gauß-Filter

während der SDE zum Teil langsamer getaktet werden muss. Eine spätere Untersuchung bewertet den TDM-Modus hinsichtlich der sich ergebenden Trade-offs.

### 3.4.2 DoG-Bilder und Extremasuche

Am Ausgang der sechs Gauß-Filter stehen Pixelstreams aller Skalenbilder synchronisiert zur Verfügung. Die jeweiligen Pixel entsprechen dabei der gleichen Position im Bild, sodass diese zur Berechnung der DoG-Bilder direkt subtrahiert werden können. Um zur Deskriptorberechnung den einem Merkmal zugehörigen Bildausschnitt vorzuhalten, welcher 20×20 px um einen Merkmalskandidaten herum umfasst, sind für den vertikalen Bildstreifen 19 Zeilenspeicher je Skalenbild notwendig. Die mittleren 5 px jedes Skalenbildstreifens im 8.8 Fixpunktformat werden vom benachbarten Streifen subtrahiert und auf 16-bit abgeschnitten, sodass die DoG-Pixel im 9.7 Format vorliegen.

Ein Array aus Schieberegistern puffert anschließend für jedes DoG-Bild ein 5×5 px großes Fenster. Dadurch stehen zur Extremasuche alle Pixel parallel zur Verfügung. Bei der Extremasuche wird das zentrale Pixel mit den inneren 3×3 px dieses Fensters auf derselben Ebene und den in der Pyramide benachbarten DoG-Bildausschnitten verglichen. Dabei werden drei unterschiedliche Skalierungen parallel betrachtet (vgl. Abb. 3.3). Im Fall eines lokalen Extremums (Minimum oder Maximum) wird das Pixel als Merkmalskandidat erkannt und die Position zusammen mit der Skalierung sowie der kompletten 5×5×3 px Umgebung um den Kandidaten zur Subpixel-Lokalisierung an den VLIW-Prozessor gegeben. Die Übergabe erfolgt parallel in 22 64-bit breiten FIFOs. Die FIFOs sind mit Block-RAMs implementiert und haben jeweils eine Tiefe von 512 Einträgen. Das Interface zum VLIW-Prozessor hat je nach Konfiguration 64-bit bzw. 128-bit. Es wurde ein rotierender Zeiger auf einen bzw. zwei FIFOs implementiert, der sich bei jedem Lesezugriff entsprechend zum nächsten FIFO weiterbewegt.

### 3.4.3 Bereitstellung des Deskriptorfensters

Zur Deskriptorberechnung wird dem VLIW-Prozessor die 20×20 px große Pixelumgebung um die Merkmalsposition bereitgestellt. Dazu wird das mittlere Skalenbild ausgewählt (vgl. Abb. 3.3), von dem ein 20 px großer vertikaler Bildstreifen in einem Pixelbuffer zwischengespeichert wird. Dieser Buffer dient zum Ausgleichen der Pipeline-Latenz bis zum Ende der Extremasuche. Steht dort ein Merkmalskandidat fest, so wird bei sehr dicht benachbarten Kandidaten, bei denen sich die Deskriptorfenster überlappen würden, zusätzlich ein Overlap-Signal generiert und zusammen mit dem vertikalen Bildstreifen im Fensterbuffer abgespeichert. Dieser Buffer kann bis zu 77 Fenster vorhalten.

Der VLIW-Prozessor liest die Daten der Merkmalskandidaten und evaluiert deren Stabilität. Im Hintergrund wird dabei das Deskriptorfenster in Streifen von 20 px in den Fenster-FIFO übertragen. Der Overlap-Controller wertet dabei aus, ob sich einer der Steifen in der Überlappung mehrerer Deskriptorfenster befindet. Ist dies der Fall, wird der Streifen in einem separaten Speicher zwischengespeichert, sodass er für den nächsten Kandidaten wieder zur Verfügung steht. Der Datentransfer zum Fenster-FIFO erfolgt dann beim nächsten Kandidaten zunächst direkt von dort, bis dieser Speicher leer ist.

Wird ein Kandidat akzeptiert, kann der VLIW-Prozessor das Fenster in 64-bit oder 128-bit Worten aus dem Fenster-FIFO auslesen. Wird der Kandidat verworfen, so kann der Fenster-FIFO durch ein Flush-Signal direkt geleert werden. Dadurch ist sichergestellt, dass das Deskriptorfenster nur dann zum VLIW-Prozessor transferiert wird, wenn tatsächlich ein Deskriptor berechnet werden muss.

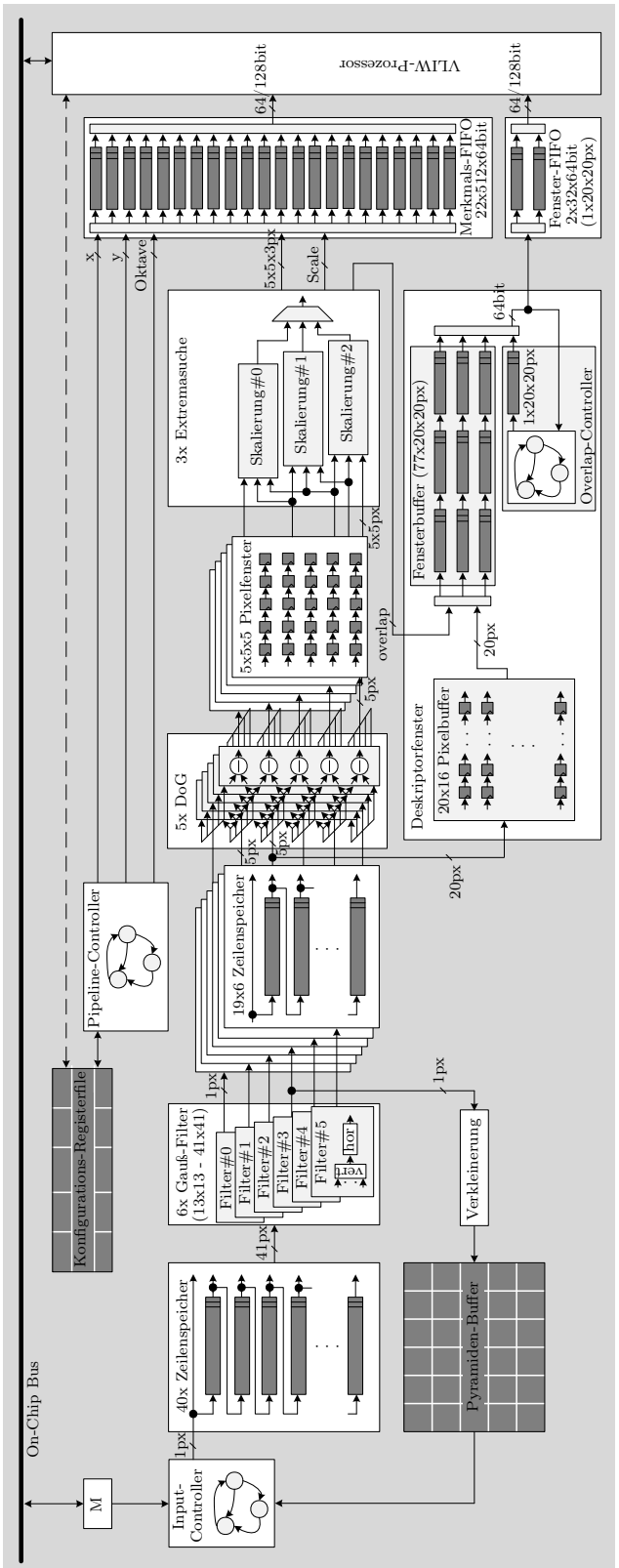



**Abbildung 3.9:** Datenpfad des SIFT Detection Engine (SDE) zur Extraktion von Merkmalskandidaten

## 3.5 Applikationsspezifischer VLIW Softcore-Prozessor

### 3.5.1 Basisarchitektur

Zur Weiterverarbeitung der Daten aus dem SIFT Detection Engine (SDE) wird in dieser Arbeit ein konfigurierbarer VLIW-Prozessor [Pay11; Pay12] verwendet. Dieser folgt dem Paradigma eines *Application-Specific Instruction-Set Processors (ASIP)* und ist in seiner Grundstruktur in Abb. 3.10 dargestellt. Durch Anpassung der Pipeline, Veränderung der Rechenressourcen und anwendungsspezifische Erweiterungen bietet die Prozessor-Architektur die Möglichkeit einer gezielten Optimierung für eine Zielanwendung.

Der Prozessor besteht aus einem 64-bit breiten Datenpfad mit zwei parallelen Issue-Slots (VLIW), sodass zwei Instruktionen gleichzeitig dekodiert und verarbeitet werden können. Das Registerfile ist in zwei Partitionen mit je 32 Registern aufgeteilt und jede der Partitionen besitzt vier Lese- und zwei Schreib-Ports, welche gleichzeitig benutzt werden können. Die Nomenklatur der Register folgt dem Schema V$a$R$b$, wobei $a$ der Partition und $b$ der Registernummer entspricht. Jedes Register kann von beiden Issue-Slots aus erreicht werden.

In der Basiskonfiguration hat der Prozessor eine fünfstufige Pipeline (*IF*, *DE*, *RA*, *EX*, *WB*). Die funktionalen Einheiten (engl. functional units, kurz: FUs) in der EX-Stage (Arithmetic-Unit, Permute-Unit, Clip-Max-Min-Unit, Shift-Round-Unit, Bit-Logic-Unit, Mul/MAC-Unit) haben dabei standardmäßig einen Takt Latenz. Durch eine Forwarding-Unit können die Ergebnisse hintereinander ausgeführter Instruktionen, die zueinander Datenabhängigkeiten aufweisen, direkt im Folgetakt wieder in die EX-Stage zurückgeführt werden, ohne zuvor ins Registerfile geschrieben worden zu sein.

Der Prozessor unterstützt horizontale Vektorisierung der Operanden (SIMD), d. h. die 64-bit breiten Register können in Subwords von 8, 16, oder 32-bit aufgeteilt und parallel verarbeitet werden. Zudem ermöglicht der Mechanismus der *predicated execution* auf Subword-Ebene, dass bestimmte Instruktionen in Abhängigkeit eines Flag-Registers nur auf manchen und nicht auf allen Subwords ausgeführt werden. Dadurch werden Kontrollfluss-Abhängigkeiten (if-else-Konstrukte) in Datenabhängigkeiten überführt, was die Anzahl der Sprünge reduziert und die Ausführung beschleunigt.

Die Ausführungseffizienz kann durch die *IPC*-Metrik (Instructions per Cycle) angegeben werden. Durch die zwei Issue-Slots liegt das Maximum hier bei zwei. Durch den Mechanismus des *X2 Instruction Merging (X2-Modus)* [Pay09] können zwei Instruktionen, die die gleiche Operation auf konsekutiven Registern ausführen, in nur einer einzigen Instruktion kodiert werden. Der IPC kann so bei ausreichenden Datenparallelitäten durch die zwei zusätzlichen virtuellen Issue Slots auf vier angehoben werden. Die Prozessorarchitektur muss dafür die entsprechende FU zweimal besitzen. Ein Code-Beispiel zum X2-Modus ist in Abb. 3.11 gegeben.

Für Sprünge ohne Bedingung stellt der Prozessor einen besonderen Mechanismus zur Verfügung. Die Sprungadressen können auf einem Stack abgelegt werden, welcher bereits in der IF-Stage ausgewertet wird. Im Vergleich zu normalen Sprüngen, bei denen das

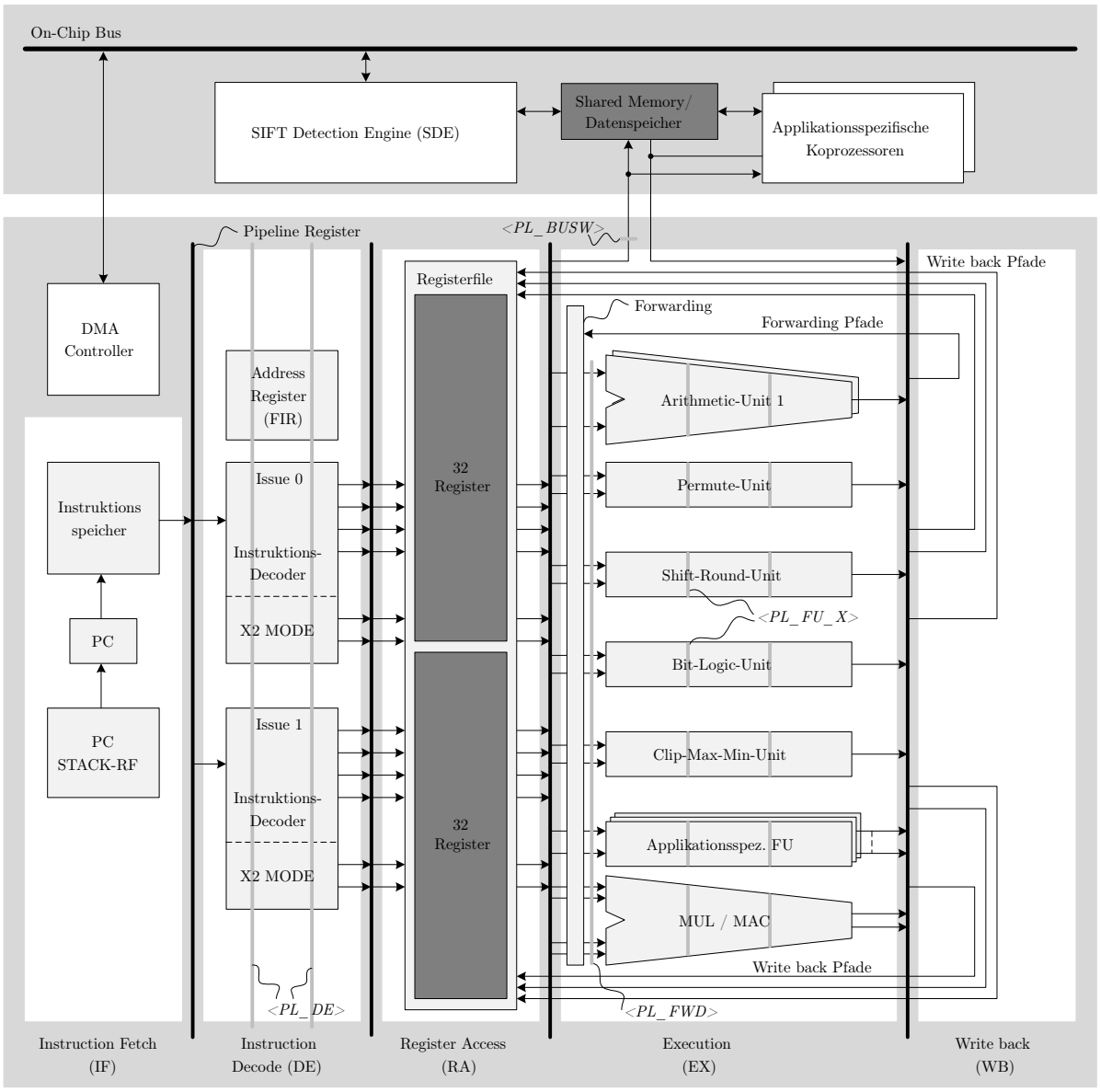



**Abbildung 3.10:** Architektur des verwendeten VLIW-Prozessors [Pay12]

Sprungziel erst nach der EX-Stage feststeht und die drei Takte Latenz besitzen, ergibt sich so stattdessen für Sprunginstruktionen mit Stack nur ein Takt Latenz.

Der Prozessor ist als Harvard-Architektur mit separatem Instruktions- und Datenspeicher aufgebaut. Ein DMA stellt Daten aus dem externen Speicher bereit. Die Daten-Transfers können dabei parallel zur Programmabarbeitung ablaufen.

```
1 // ohne X2-Modus
2 ADD_8    V0R0, V1R1, V0R10
3 ADD_8    V0R1, V1R2, V0R10
4
5 // mit X2-Modus (funktional äquivalent)
6 ADD_8_X2 V0R0+V0R1, V1R1+V1R2, V0R10
```

**Abbildung 3.11:** Code-Beispiel zum X2-Modus. Die zwei oberen ADD-Instruktionen (8-bit SIMD) können in nur einer Instruktion kodiert werden, da sie die gleiche Operation auf konsekutiven Registern ausführen.

### 3.5.2 Instruction Scheduler

In VLIW-Architekturen wird das Sortieren und das bei zwei Issue-Slots paarweise Anordnen von Einzelinstruktionen als *Instruction Scheduling* bezeichnet. Ziel ist es, eine möglichst hohe Kompaktheit des Codes zu erhalten, um möglichst wenig Leerzyklen (NOPs) in der Pipeline zu bekommen. Abhängigkeiten zwischen den Instruktionen sowie Konflikte um Hardwareressourcen sind zu berücksichtigen. Da dieses Problem NP-vollständig ist [Hen83], kann eine optimale Lösung vermutlich nicht auf effizientem Wege erreicht werden. Für die meisten Programme ist die Anzahl an verschiedenen möglichen Anordnungen zu hoch für eine vollständige Suche. Es müssen daher Heuristiken verwendet werden, um die Anzahl an alternativen Anordnungen, die betrachtet werden, zu reduzieren und in überschaubarer Zeit eine ausreichend gute Lösung zu generieren. Das Instruction Scheduling ist statisch, d. h. die gefundene Anordnung der Instruktionen wird offline vor der Ausführung des Codes durchgeführt und zur Laufzeit nicht mehr verändert. Der hier verwendete Instruction Scheduler für die VLIW-Architektur wurde in [Pay07] vorgestellt und in [Gie17; Gie20] weiterentwickelt.

Der Scheduling Algorithmus basiert auf dem *List-Scheduling*-Verfahren [Fis79]. Dabei wird der Code in Basisblöcke unterteilt, welche nur einen Eintrittspunkt am Anfang haben und keine weiteren Abzweigungen außer am Ende des Blocks. Diese Blöcke werden auch als *Straight Line Microcode (SLM)* bezeichnet. Das Scheduling wird danach lokal für jedes SLM einzeln durchgeführt, um die Komplexität zu reduzieren. Für jedes SLM wird dann ein Abhängigkeitsgraph generiert, bei dem die Knoten mit Gewichten versehen werden. Die Gewichte eines SLMs werden vom Scheduler als Heuristik verwendet und mit einem genetischen Algorithmus über mehrere Iterationen (Generationen) hinweg erzeugt und optimiert. Dabei entspricht ein SLM einem Individuum und die gewichteten Instruktionen den Genen im Chromosom, welche durch Crossover und Mutationen verändert werden. Als Fitnessfunktion wird die resultierende Anzahl an Takten eines fertig geschedulten SLMs verwendet. Die Populationsgröße (Anzahl an Individuen pro Generation) stellt einen wichtigen Optimierungsparameter dar, da dieser maßgeblich die Laufzeit des Schedulings bestimmt und gegen einen Gewinn in Taktzyklen abgetauscht werden kann.

Um die Komplexität für den Programmierer zu verringern, ist es möglich, virtuelle Register zu verwenden (Nomenklatur VxR$i$), die vom Scheduler auf reale Register

abgebildet werden (Registerallokation). Die Registerallokation wird immer nach einem erfolgreichen Scheduling-Schritt durchgeführt. Kann aufgrund der begrenzten Lese- und Schreib-Ports am Registerfile keine gültige Allokation gefunden werden, wird eine hohe Penalty dem Fitness-Wert hinzuaddiert.

Abb. 3.12 zeigt ein exemplarisches Instruction Scheduling mit virtuellen Registern, bei dem die Instruktionsgewichte zum einen mit einem einfachen Ansatz (Position im Abhängigkeitsgraph; Abb. 3.12b) und zum anderen mit dem genetischen Algorithmus (Abb. 3.12c) generiert wurden. Zwei *ADD*-Instruktionen können nicht parallel ausgeführt werden, da hier lediglich eine einzelne Arithmetic-Unit zur Verfügung steht. Aufgrund der Limitierung in der Registeranzahl und den Registerfile-Ports (jeweils vier Lese- und zwei Schreibports pro Registerfilepartition V0R$i$ bzw. V1R$i$) ergeben sich weitere Bedingungen für den Scheduler. Mit dem generischen Algorithmus kann ein Scheduling gefunden werden, das zwar einen Takt länger ist, dafür allerdings ein Register weniger verwendet. Der genetische Algorithmus ist in der Lage, auch dann ein gültiges Scheduling für komplexe SLMs zu finden, für die aufgrund der limitierten Registerressourcen ansonsten nur schwer eine Lösung gefunden werden kann.

Zudem ist es möglich, mit einem weiteren genetischen Algorithmus ein automatisches X2-Merging durchzuführen. Der Optimierungsalgorithmus kann im Vergleich zu einem manuellen X2-Merging in vielen Fällen kompaktere Schedulings finden. Auch hierbei kann durch die Populationsgröße Kompaktheit einer SLM gegen Laufzeit des Merging-Schrittes abgetauscht werden.

### 3.5.3 Applikationsspezifische Architekturanpassungen

Die Prozessor-Architektur bietet Konfigurationsmöglichkeiten, um sie an eine Zielanwendung anzupassen. Dabei kann zwischen *Parallelisierung* und *Spezialisierung* unterschieden werden. Auf Basis dieser beiden Mechanismen kann darüber hinaus noch eine *Pipeline-Optimierung* durchgeführt werden.

#### Parallelisierung

Idealerweise enthält der geschedulte VLIW-Code keine Leerzyklen (NOPs), da der eine der beiden Issue-Slots Daten lädt bzw. in den Speicher zurückschreibt und der andere die geladenen Daten kontinuierlich verarbeitet. Aufgrund fehlender Datenparallelität in der Anwendung, limitierter Rechenressourcen oder einem ungleichen Verhältnis von Daten- und Speicherinstruktionen ist dies in realen Anwendungen selten der Fall, sodass der IPC immer unter zwei liegt.

Um die Auslastung der Pipeline zu erhöhen, können die jeweils zur Verfügung stehenden FUs *dupliziert* werden. Dies ermöglicht die parallele Ausführung von z. B. zwei arithmetischen oder zwei Speicher-Instruktionen. Sofern genug Datenparallelitäten zur Verfügung stehen, können so unter zusätzlichem Hardwareaufwand die NOPs gefüllt werden. Das Scheduling-Ergebnis ist somit kompakter, und der Prozessor kann die gleiche

```
 1 ADD      VxR0, V0R0, V0R1
 2 SRI      VxR1, VxR0, #3
 3 MAX      VxR2, VxR1, V0R2
 4 PERMREG  VxR2, VxR0, VxR2
 5
 6 ADD      VxR0, V0R0, V0R3
 7 MAX      VxR2, VxR0, V0R2
 8 PERMREG  VxR2, VxR0, VxR2
 9
10 ADD      VxR0, V0R0, V1R0
11 MAX      VxR2, VxR0, V0R2
```

**(a)** Code-Beispiel mit ungeschedultem Assemblercode. Die Register VxR$i$ sind virtuelle Register und müssen vom Scheduler noch allokiert werden.

```
1 (1) ADD      V1R1, V0R0, V0R1 ; (0)  NOP
2 (2) SRI      V1R2, V1R1, #3   ; (6)  ADD V1R3, V0R0, V0R3
3 (3) MAX      V1R2, V1R2, V0R2 ; (10) ADD V0R4, V0R0, V1R0
4 (4) PERMREG  V1R2, V1R1, V1R2 ; (7)  MAX V1R1, V1R3, V0R2
5 (8) PERMREG  V1R1, V1R3, V1R1 ; (11) MAX V1R3, V0R4, V0R2
```

**(b)** Mit einer einfachen Heuristik geschedulter Assemblercode (zwei Issue-Slots). Das Programm benötigt fünf Takte und neun Register.

```
1 (1)  ADD      V1R1, V0R0, V0R1 ; (0)  NOP
2 (2)  SRI      V1R3, V1R1, #3   ; (6)  ADD V1R2, V0R0, V0R3
3 (3)  MAX      V1R3, V1R3 V0R2  ; (0)  NOP
4 (4)  PERMREG  V1R1, V1R1, V1R3 ; (7)  MAX V1R3, V1R2, V0R2
5 (8)  PERMREG  V1R2, V1R2, V1R3 ; (10) ADD V1R3, V0R0, V1R0
6 (11) MAX      V1R3, V1R3, V0R2 ; (0)  NOP
```

**(c)** Mit dem genetischen Algorithmus geschedulter Assemblercode. Das Programm benötigt sechs Takte, allerdings nur acht Register, da ein zusätzliches NOP eingefügt wurde.

**Abbildung 3.12:** Exemplarisches Instruction Scheduling aus [Gie17]

Anzahl an Instruktionen in weniger Taktzyklen ausführen, was den IPC erhöht und die Effizienz steigert.

Darüber hinaus kann bei duplizierten FUs der *X2-Modus* genutzt werden, um gleiche Instruktionen zu verschmelzen und so zwei weitere virtuelle Issue-Slots mit wenig Hardwareoverhead zu erzeugen (vgl. Abb. 3.11).

### Spezialisierung

Im vorangegangenen Abschnitt wurde erläutert, wie eine Durchsatzsteigerung durch Parallelisierung erreicht werden kann. Dabei bleibt die Anzahl der auszuführenden Instruktionen unverändert. Alle Maßnahmen erhöhen lediglich den IPC. Sollte die so erzielte Verringerung der Ausführungszeit eines Programms noch nicht ausreichen, kann die Anzahl an Instruktionen durch applikationsspezifische Instruktionen und Koprozessoren, d. h.

dedizierte Zusatzmodule reduziert werden. Beide Varianten sind in Abb. 3.10 dargestellt.

Applikationsspezifische Instruktionen fassen mehrere Einzelinstruktionen des Programms zusammen oder führen Berechnungen durch, die mit dem Standardinstruktionssatz nicht zu realisieren wären. Die dafür entwickelte *applikationsspezifische FU* befindet sich direkt in der EX-Stage der Pipeline. Dadurch kann zwar direkt auf das Registerfile zugegriffen werden, allerdings ergeben sich durch die zusätzlichen Schaltungsteile Wechselwirkungen mit der umliegenden Schaltung. Durch die zusätzliche FU vergrößern sich die Forwarding-Strukturen, was einen negativen Einfluss auf das Timing des Prozessors hat. Zudem muss der kritische Pfad der FU selbst kurz genug sein, um die Verzögerungszeit der Pipeline-Stufe nicht zu verlängern. Daher eignen sich applikationsspezifische Instruktionen vor allem für weniger komplexe Aufgaben.

Für größere Aufgaben können *applikationsspezifische Koprozessoren* angekoppelt werden. Der Datenaustausch erfolgt durch Speicheroperationen über einen geteilten Speicher. Die Latenz des Koprozessors für die spezielle Aufgabe kann durch die geschedulte Latenz zwischen Schreiben der Eingangsdaten und Lesen der Ergebnisse gesteuert werden. Somit ist der Timing-Einfluss auf den Prozessor gering. Darüber hinaus kann der Speicher zum Datenaustausch in verschiedenen Taktdomänen liegen, wie hier für die Anbindung des SIFT Detection Engine (SDE) geschehen.

### Pipeline-Optimierung

Durch erhöhte Parallelisierung und Spezialisierung mit dedizierten Modulen ist die Struktur des Prozessors klar definiert. Weiteres Potential zur Erhöhung der Prozessorleistung liegt jetzt noch in der Optimierung der Pipeline-Stufen, da die einzelnen Stufen und insbesondere die verschiedenen FUs sich in der Länge ihres jeweiligen kritischen Pfades unterscheiden. Einige der konfigurierbaren Pipeline-Register sind in Abb. 3.10 dargestellt und mit den Kürzeln $<PL\_XX>$ gekennzeichnet. Tab. 3.3 listet diese zusätzlich zur Übersicht auf. Es existieren weitere Register, welche in Abschnitt A.4.2 im Anhang gelistet sind. Durch die Erhöhung der Pipeline-Stufen ergeben sich zusätzliche Latenzzyklen der Instruktionen. Wenn zwei Instruktionen eine Datenabhängigkeit aufweisen, müssen sie vom Scheduler einen Takt weiter voneinander entfernt platziert werden. Dazu zeigt Abb. 3.13 ein Scheduling-Beispiel. Mit einer Standardpipeline benötigt der Beispielcode vier Takte zur Ausführung (Abb. 3.13a). Durch eine zusätzliche Pipeline-Stufe in der Arithmetic-Unit zur Verringerung des dortigen kritischen Pfades erhöht sich die Latenz der ADD- und SUB-Instruktion um einen Takt (Abb. 3.13b). Im Beispiel muss so die MUL-Instruktion trotz Forwarding zwei Takte auf das Ergebnis der SUB-Instruktion warten. Wird außerdem eine Pipeline-Stufe im Forwarding eingefügt (Abb. 3.13c) erhält jede FU eine zusätzliche Latenz, sodass im Beispiel jetzt sieben Takte benötigt werden.

Die durch die Pipeline-Erweiterung entstandenen zusätzlich notwendigen Takte müssen beim Scheduling mit Instruktionen gefüllt werden, damit der IPC erhalten bleibt. Stehen keine Instruktionen zur Verfügung, werden NOPs eingefügt. Der IPC und damit die Ausführungseffizienz sinkt. Um eine echte Verringerung der Ausführungszeit einer

**Tabelle 3.3:** Konfigurationsmöglichkeiten für zusätzliche Pipeline-Stufen (vgl. Abb. 3.10)

| Kürzel | Beschreibung | Latenz-Penalty |
|---|---|---|
| <PL_DE> | Decode-Stage | +1 auf Branches |
| <PL_BUSW> | Bus Write | +1 auf alle STOREs |
| <PL_FU_x> | individuell für jede FU | +x auf die jeweilige FU |
| <PL_FWD> | Forwarding Multiplexer | +1 auf alle FUs |

```
1 ADD   VxR2,   VxR0, VxR1 ; SRI VxR6, VxR5, #8
2 SUB   VxR3,   VxR0, VxR1 ; SLI VxR8, VxR7, #8
3 MUL   VxR4,   VxR2, VxR3 ; MAX VxR9, VxR6, VxR8
4 STORE 0x402a, VxR4       ; NOP
```

**(a)** Standardpipeline. IPC: 1,75

```
1 ADD   VxR2,   VxR0, VxR1 ; SRI VxR6, VxR5, #8
2 SUB   VxR3,   VxR0, VxR1 ; SLI VxR8, VxR7, #8
3 MAX   VxR9,   VxR6, VxR8 ; NOP
4 MUL   VxR4,   VxR2, VxR3 ; NOP
5 STORE 0x402a, VxR4       ; NOP
```

**(b)** Zusätzliche Pipeline-Stufe in der Arithmetic-Unit (<PL_AU_1>). IPC: 1,4

```
1 ADD   VxR2,   VxR0, VxR1 ; SRI VxR6, VxR5, #8
2 SUB   VxR3,   VxR0, VxR1 ; SLI VxR8, VxR7, #8
3 NOP                      ; NOP
4 MAX   VxR9,   VxR6, VxR8 ; NOP
5 MUL   VxR4,   VxR2, VxR3 ; NOP
6 NOP                      ; NOP
7 STORE 0x402a, VxR4       ; NOP
```

**(c)** Zusätzliche Pipeline-Stufe in der Arithmetic-Unit (<PL_AU_1>) und im Forwarding Multiplexer (<PL_FWD>). IPC: 1,0

**Abbildung 3.13:** Exemplarisches Instruction Scheduling bei Veränderung der Pipeline.

Anwendung zu erreichen, müssen die zusätzlichen Leerzyklen durch einen Zugewinn an Geschwindigkeit aufgrund der nun kürzeren Periodenlänge kompensiert werden. Im Beispiel wird nur dann ein Speedup erzielt, wenn sich durch die zusätzlichen Pipeline-Stufen die Taktfrequenz um mehr als $\frac{5}{4}\times$ bzw. $\frac{7}{4}\times$ erhöht. Dies ist in hohem Maße abhängig von der Anwendung und der jeweiligen Prozessorstruktur und muss im Einzelfall betrachtet werden. Der Einfluss dieser Methodik auf den Energieverbrauch aufgrund der geringeren Ausführungseffizienz und des höheren Takts soll später untersucht werden.

### 3.5.4 SIFT-Implementierung

Der VLIW-Prozessor kommuniziert mit dem Host-PC über Semaphor-Signale (siehe dazu Abb. A.5 im Anhang) und steuert den Kontrollfluss des Gesamtsystems. Der SIFT

Detection Engine (SDE) wird ebenfalls vom VLIW aus konfiguriert und gesteuert. Der VLIW bekommt zunächst vom Host-PC ein Signal, dass ein Bild im externen Speicher bereit liegt. Danach sendet der VLIW seinerseits dem SDE ein Startsignal und wartet in einer Schleife auf Merkmalskandidaten aus dem Merkmals-FIFO. Diese werden ausgelesen und weiterverarbeitet, bevor wieder in die Warteschleife gewechselt wird. Meldet der SDE das Ende der Berechnung und ist der Merkmals-FIFO leer, schreibt der VLIW noch Kontroll- und Profiling-Informationen in den externen Speicher und setzt das Semaphor für den Host-PC, um einen neuen Bildzyklus einzuleiten.

Nachfolgend wird die Implementierung der verschiedenen SIFT-Teilschritte auf dem VLIW erläutert. Die Implementierung erfolgt dabei direkt mit Assemblerbefehlen.

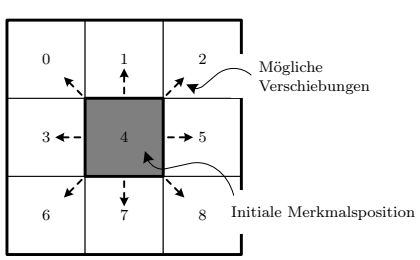
### Subpixel-Lokalisierung

Zunächst werden die Daten der Kandidaten aus dem Merkmals-FIFO gelesen. Dazu erfolgen 22 64-bit Zugriffe auf den Speicher im SDE. Durch die Verwendung des X2-Modus können zwei 64-bit LOAD-Instruktionen in einer LOAD_X2-Instruktion kodiert werden, was 128-bit breite Zugriffe ermöglicht.

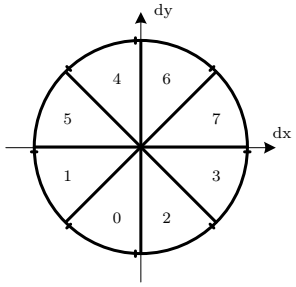
Danach werden die ersten und zweiten Ableitungen in der 3×3×3 DoG-Pixelumgebung (x- und y-Richtung, Skalierung) um das Merkmal anhand von Pixeldifferenzen berechnet und die Subpixelposition nach Gl. 2.15 bestimmt. Anstatt der Invertierung der 3×3-Hesse-Matrix wird das Gleichungssystem mittels der Cramerschen Regel gelöst. Dies erfordert die Berechnung von vier Determinanten und drei Divisionen. Während die Pixelberechnungen mit 16-bit SIMD-Instruktionen durchgeführt werden, erfolgen die Division in 32-bit Operationen. Es wird dazu ein dedizierter Koprozessor verwendet, der die Division nach dem *non-restoring Algorithmus* durchführt [Pir98]. Der Koprozessor hat eine Latenz von $1 + 32/x + 2$ Takten, wobei $x$ die Werte 1, 2, 4 und 8 annehmen kann und den Leveln an Logikstufen im Koprozessor entspricht. Dieses bestimmt die Anzahl an berechneten Bits pro Takt, wodurch Latenz gegen Hardwareaufwand abgetauscht werden kann. Die Operanden werden in den Koprozessor per STORE-Instruktion geschrieben, und das Ergebnis kann nach Ablauf der Latenztakte mit einer LOAD-Instruktion ins Registerfile zurückgelesen werden. Es ist möglich, den Koprozessor zweimal zu instantiieren, sodass die zwei Divisionen zur beschleunigten Ausführung parallel ablaufen können.

Ist die berechnete Subpixelposition größer gleich einem Pixel in x- oder y-Richtung, so erfolgt eine Verschiebung der Merkmalsposition, weshalb dem VLIW eine 5×5×3 DoG-Pixelumgebung vorliegt. Die Verschiebung wird in einen Index zum entsprechend verschobenen Lesen der Daten umgerechnet, wobei ein Index von 4 keiner Verschiebung entspricht (siehe Abb. 3.14a). Zur beschleunigten Ausführung wird der Prozessor dafür um die 16-bit SIMD-Spezialinstruktion *getIdxPatch* erweitert, welche den Index in nur einem Takt berechnet:

$$\begin{aligned} z_{index} &= getIdxPatch\,(\,[\,0,0,x_{\text{alt}}\,,\,y_{\text{alt}}\,],[\,0,0,x_{\text{neu}},y_{\text{neu}}\,]\,) \\ &= 3\,(y_{\text{neu}} - y_{\text{alt}}) + x_{\text{neu}} - x_{\text{alt}} + 4 \end{aligned} \tag{3.14}$$




**(a)** Indizierung bei der Berechnung der Verschiebung der Merkmalsposition




**(b)** Indizierung der Orientierungen bei der Berechnung des Deskriptors

**Abbildung 3.14:** Skizzierung der Index-Reihenfolge während der Berechnungen

Nach der ersten Verschiebung werden die Ableitungen erneut bestimmt und das Gleichungssystem wird ein zweites Mal gelöst. Eine zweite Verschiebung findet nicht statt (vgl. Abschnitt 3.2). Um keinen Code zu replizieren, erfolgt die Berechnung in einer Schleife. Die Schleife enthält drei bedingte Sprünge, die den Kandidaten verwerfen können, und einen bedingten Sprung aus der Schleife heraus. Zudem werden viele Kandidaten um weniger als eine ganze Pixelposition verschoben, was dazu führt, dass die Schleife kein zweites Mal ausgeführt werden muss. Aufgrund dieser Kontrollstrukturen ist der Programmablauf stark datenabhängig, d. h. die Anzahl an Instruktionen und damit die Laufzeit pro Merkmalskandidat ist nicht deterministisch. Der Kontrollflussgraph zum Programm ist in Abb. A.4 im Anhang dargestellt.

### Stabilitätsprüfung

In diesem Schritt werden die bereits in der Position mit Subpixelgenauigkeit lokalisierten Merkmalskandidaten auf algorithmische Stabilität überprüft. Zur Überprüfung des Bildkontrastes wird zunächst der DoG-Wert an der neuen Position nach Gl. 2.16 berechnet und mit einem Schwellwert verglichen. Für den Kantentest nach Gl. 2.17 können die Werte der Hesse-Matrix aus der Subpixel-Lokalisierung wiederverwendet werden. Schlagen beide Tests fehl, werden der Kandidat und das zugehörige Pixelfenster für den Deskriptor verworfen (vgl. Abschnitt 3.4.3). Andernfalls wird das nun als stabil eingestufte SIFT-Merkmal per DMA-Transfer in den externen Speicher übertragen, und der Deskriptor kann berechnet werden. Der DMA-Transfer läuft dabei im Hintergrund ab, sodass direkt mit der Berechnung des Deskriptors begonnen werden kann.

Einzig in diesem Teilschritt der kompletten Applikation werden zwei 64-bit Shift-Instruktionen benötigt, in welchen der Shift-Wert sogar konstant ist. Daher wurden diese Instruktionen durch Spezialinstruktionen ersetzt, welche während der Hardwaresynthese lediglich durch Verdrahtung implementiert werden können. Dadurch kann der Barrel-Shifter in der Shift-Round-Unit des Prozessors auf 32-bit reduziert werden, was den kritischen Pfad dort verringert.

### Deskriptorberechnung

Zu Beginn der Deskriptorberechnung wird eine 20×20 px große Umgebung um das Merkmal herum aus dem Speicher im SDE in den lokalen Datenspeicher des VLIW transferiert. Der Datentransfer erfolgt entweder mit 64-bit oder mit 128-bit (X2-Modus) Datenwortbreite. Die Pixelumgebung ist in Abb. 3.15(a) dargestellt. Insgesamt müssen 60 64-bit Transfers durchgeführt werden, wobei ein Register genau acht Pixel enthält. Die Reihenfolge der Daten im Speicher wird mit *data #0-#3* verdeutlicht. Überschüssige Pixel in *data #2* werden in Kauf genommen und verworfen, um das Alignment der Registerinhalte nicht zu verletzen.

Das Deskriptorhistogramm wird aus einem 16×16 px großen Fenster um die Merkmalsposition herum gebildet. Zunächst werden die Gradienten in x- und y-Richtung durch einfache Pixeldifferenzen bestimmt (siehe Abb. 3.15(b)), wofür eine Umgebung von 18×18 px um die Merkmalsposition nötig wäre. Da das Merkmal während der Subpixel-Lokalisierung um bis zu einen Pixel in jede Richtung verschoben werden kann (Abb. 3.15(c)), vergrößert sich die benötigte Umgebung auf 20×20 px. Der VLIW-Prozessor ist durch seine vektorisierten Instruktionen für die sich ergebenden regulären Datenzugriffsmuster sehr gut geeignet, sodass die Fensterverschiebung und die Gradienten mithilfe von 8-/16-bit SIMD Subword-Permutationen und Subtraktionen sehr effizient berechnet werden können.

Die Deskriptorberechnung benötigt im Vergleich zu den anderen Teilschritten die meisten Operationen, sodass der VLIW zur Reduktion der Anzahl an Instruktionen um mehrere Spezialinstruktionen erweitert wurde. Die Instruktion *subexpandL (subexpandH)* erweitert die unteren (oberen) vier 8-bit Subwords der Operanden auf 16-bit und subtrahiert diese voneinander:

$$\begin{aligned}[z_3, z_2, z_1, z_0] &= subExpandX\left([x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0]\,,\,[y_7, y_6, y_5, y_4, y_3, y_2, y_1, y_0]\right) \\ &= [\,x_{3+i} - y_{3+i}\,,\, x_{2+i} - y_{2+i}\,,\, x_{1+i} - y_{1+i}\,,\, x_{0+i} - y_{0+i}\,] \\ \text{mit} \qquad i &= \begin{cases} 0 & \text{für } subExpandL \\ 4 & \text{für } subExpandH \end{cases}\end{aligned} \tag{3.15}$$

Ein Codebeispiel dazu ist in Abb. 3.16 gegeben. Die sechs Instruktionen in VLIW-Basiskonfiguration reduzieren sich durch die Verwendung des X2-Modus für die Arithmetic-Unit auf vier Instruktionen. Durch die Spezialinstruktion lässt sich die gleiche Funktionalität in zwei bzw. einer Instruktion bei Duplizierung und X2-Modus realisieren. Zur Berechnung des Gradientenbetrags wird die L1-Norm verwendet. Dazu wurde die Spezialinstruktion *absAbsAdd* hinzugefügt:

$$\begin{aligned}[z_3, z_2, z_1, z_0] &= absAbsAdd\left([x_3, x_2, x_1, x_0]\,,\,[y_3, y_2, y_1, y_0]\right) \\ &= [\,|x_3| + |y_3|\,,\, |x_2| + |y_2|\,,\, |x_1| + |y_1|\,,\, |x_0| + |y_0|\,]\end{aligned} \tag{3.16}$$

Das Codebeispiel dazu in Abb. 3.17 zeigt eine Reduktion der Anzahl an Instruktionen von sechs auf eins.

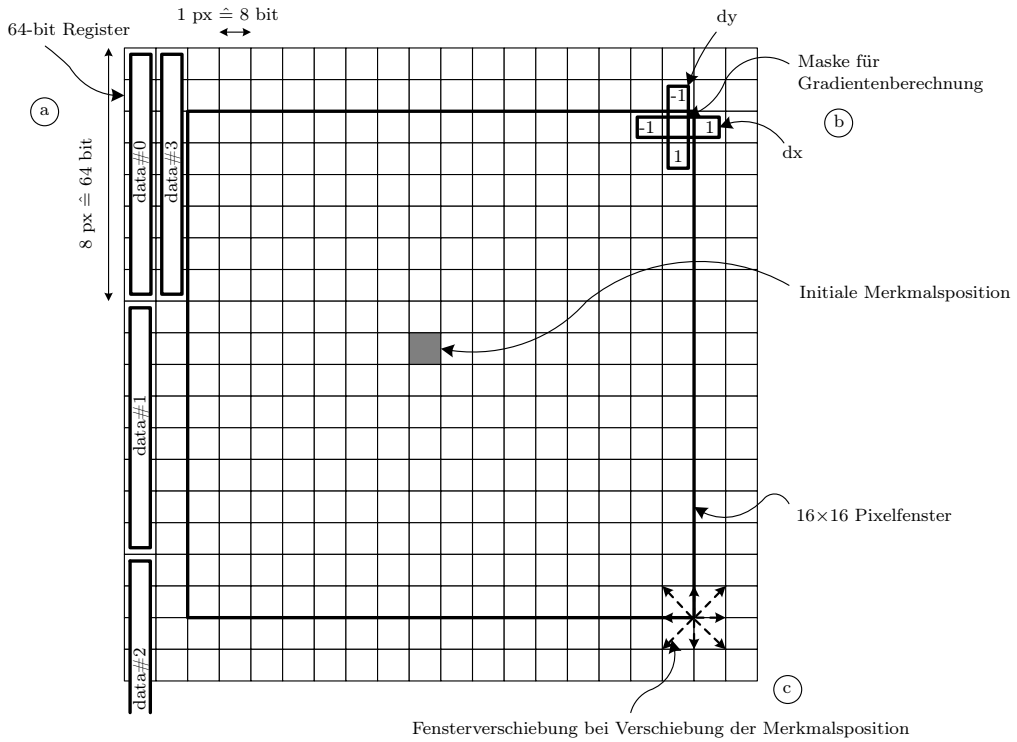



**Abbildung 3.15:** 20×20 px große Umgebung um die initiale Merkmalsposition, aus dessen inneren 16×16 px der Deskriptor gebildet wird. (a) Datenreihenfolge im Speicher, (b) Gradientenberechnung durch Pixeldifferenzen, (c) Verschiebung des inneren Fensters um 1 px aufgrund einer Änderung der Merkmalsposition während der Subpixel-Lokalisierung.

Das Deskriptorhistogramm akkumuliert die Gradientenbeträge in acht Winkelbins. Eine genaue Berechnung eines Gradientenwinkels ist daher nicht erforderlich, sondern nur die Bestimmung des entsprechenden Bins. Dieser kann durch die drei einfachen Vergleiche $|dx_i| \geq |dy_i|$, $dx_i \geq 0$ und $dy_i \geq 0$ bestimmt werden. Eine Spezialinstruktion *getIdxOri*

$$\begin{aligned} [z_3, z_2, z_1, z_0] &= getIdxOri\left([dx_3, dx_2, dx_1, dx_0], [dy_3, dy_2, dy_1, dy_0]\right) \\ \text{mit} \qquad z_i &= (|dx_i| \geq |dy_i| \,?\, 1 : 0) + (dx_i \geq 0 \,?\, 2 : 0) + (dy_i \geq 0 \,?\, 4 : 0) \end{aligned} \tag{3.17}$$

kann dies in einem einzigen Takt realisieren. Die daraus resultierende Indizierung der Bins ist in Abb. 3.14b dargestellt. Auch hier zeigt in Abb. 3.18 ein Code-Beispiel den Gewinn der Spezialinstruktion gegenüber einer Implementierung mit dem Standardinstruktionssatz. Dieser verwendet 16-bit SIMD-Instruktionen und die *predicated execution* des VLIW, um durch den *_CS*-Zusatz bei der Subtraktion Ergebnis-Flags zu schreiben und danach abhängig von der Bedingung *COND_GE* (größer gleich) mit dem Zusatz *_CR* die Addition nur auf denjenigen Subwords auszuführen, die die zuvor gesetzte Bedingung

```
1  // in Basiskonfiguration (6 Instruktionen)
2  ADDEXPANDL_8 VxR2, VxR0, R_ZERO // expandiere von 8 auf 16 bit
3  ADDEXPANDL_8 VxR3, VxR1, R_ZERO
4  ADDEXPANDH_8 VxR4, VxR0, R_ZERO
5  ADDEXPANDH_8 VxR5, VxR1, R_ZERO
6  SUB_16       VxR6, VxR3, VxR2 // dx[3:0]
7  SUB_16       VxR7, VxR5, VxR4 // dx[7:4]
8
9  // mit X2-Modus für Arithmetic-Unit (4 Instruktionen)
10 ADDEXPANDL_8_X2 VxR2+VxR3,  VxR0+VxR1, R_ZERO // expandiere von 8 auf 16 bit
11 ADDEXPANDH_8_X2 VxR4+VxR5,  VxR0+VxR1, R_ZERO
12 SUB_16          VxR6, VxR3, VxR2 // dx[3:0]
13 SUB_16          VxR7, VxR5, VxR4 // dx[7:4]
14
15 // mit Spezialinstruktionen SUBEXPANDL und SUBEXPANDH (2 Instruktionen)
16 SUBEXPANDL VxR6, VxR1, VxR0 // dx[3:0]
17 SUBEXPANDH VxR7, VxR1, VxR0 // dx[7:4]
18
19 // mit X2-Modus für Spezialinstruktionen SUBEXPANDL und SUBEXPANDH (1 Instruktion)
20 SUBEXPANDLH_X2 VxR6+VxR7, VxR1, VxR0 // dx[3:0] | dx[7:4]
```

**Abbildung 3.16:** Code-Beispiel für *subExpandL* und *subExpandH*

```
1  // in Basiskonfiguration (6 Instruktionen)
2  ABSADD_16 VxR2, R_ZERO, VxR0 // abs(dx[3:0])
3  ABSADD_16 VxR3, R_ZERO, VxR1 // abs(dx[7:4])
4  ABSADD_16 VxR6, R_ZERO, VxR4 // abs(dy[3:0])
5  ABSADD_16 VxR7, R_ZERO, VxR5 // abs(dy[7:4])
6  ADD_16    VxR8, VxR2,   VxR6 // mag[3:0]
7  ADD_16    VxR9, VxR3,   VxR7 // mag[7:4]
8
9  // mit X2-Modus für Arithmetic-Unit (3 Instruktionen)
10 ABSADD_16_X2 VxR2+VxR3, R_ZERO,    VxR0+VxR1 // abs(dx[3:0]) | abs(dx[7:4])
11 ABSADD_16_X2 VxR6+VxR7, R_ZERO,    VxR4+VxR5 // abs(dy[3:0]) | abs(dy[7:4])
12 ADD_16_X2    VxR8+VxR9, VxR2+VxR3, VxR6+VxR7 // mag[3:0] | mag[7:4]
13
14 // mit Spezialinstruktion ABS_ABS_ADD (2 Instruktionen)
15 ABS_ABS_ADD VxR8, VxR0, VxR4 // mag[3:0] = abs(dx[3:0]) + abs(dy[3:0])
16 ABS_ABS_ADD VxR9, VxR1, VxR5 // mag[7:4] = abs(dx[7:4]) + abs(dy[7:4])
17
18 // mit X2-Modus für Spezialinstruktion ABS_ABS_ADD (1 Instruktion)
19 ABS_ABS_ADD_X2 VxR8+VxR9, VxR0+VxR1, VxR4+VxR5 // mag[3:0] = abs(dx[3:0]) + abs(dy[3:0])
20                                                // | mag[7:4] = abs(dx[7:4]) + abs(dy[7:4])
```

**Abbildung 3.17:** Code-Beispiel für *absAbsAdd*

erfüllen. Durch die Spezialinstruktion und den X2-Modus können 15 Instruktionen im Basisprozessor in nur eine einzige Instruktion komprimiert werden.

Darüber hinaus wurden noch die Spezialinstruktionen $mvImmediateCond\_X$ erstellt:

$$z = mvImmediateCond\_X(x, y) = \begin{cases} C_P & x > y \\ C_Z & x = y \\ C_N & sonst \end{cases} \tag{3.18}$$

```
1  // in Basiskonfiguration (15 Instruktionen)
2  SMVI       V0CONDSEL, #COND_GE // schreiben der Bedingung für _CR (condition read)
3                                 // Instruktionen
4  MVI_16     VxR0,  #0           // init ori[3:0]
5  SUBCS_16   VxR10, VxR6, VxR8   // abs(dx[3:0]) - abs(dy[3:0])
6  ADDICR_16  VxR0,  VxR0, #1     // +1
7  SUBCS_16   VxR12, VxR2, R_ZERO // dx[3:0] - 0
8  ADDICR_16  VxR0,  VxR0, #2     // +2
9  SUBCS_16   VxR14, VxR4, R_ZERO // dy[3:0] - 0
10 ADDICR_16  VxR0,  VxR0, #4     // +4
11
12 MVI_16     VxR1,  #0           // init ori[7:4]
13 SUBCS_16   VxR11, VxR7, VxR9   // abs(dx[7:4]) - abs(dy[7:4])
14 ADDICR_16  VxR1,  VxR1, #1     // +1
15 SUBCS_16   VxR13, VxR3, R_ZERO // dx[7:4] - 0
16 ADDICR_16  VxR1,  VxR1, #2     // +2
17 SUBCS_16   VxR15, VxR5, R_ZERO // dy[7:4] - 0
18 ADDICR_16  VxR1,  VxR1, #4     // +4
19
20 // mit X2-Modus (8 Instruktionen)
21 SMVI          V0CONDSEL,    #COND_GE
22 MVI_16_X2     VxR0+VxR1,    #0                    // init ori[3:0] | ori[7:4]
23 SUBCS_16_X2   VxR10+VxR11, VxR6+VxR7, VxR8+VxR9 // abs(dx[3:0]) - abs(dy[3:0]) |
24                                                  // abs(dx[7:4]) - abs(dy[7:4])
25 ADDICR_16_X2  VxR0+VxR1,    VxR0+VxR1, #1        // +1
26 SUBCS_16_X2   VxR12+VxR13, VxR2+VxR3, R_ZERO    // dx[3:0] | dx[7:4] - 0
27 ADDICR_16_X2  VxR0+VxR1,    VxR0+VxR1, #2        // +2
28 SUBCS_16_X2   VxR14+VxR15, VxR4+VxR5, R_ZERO    // dy[3:0] | dy[7:4] - 0
29 ADDICR_16_X2  VxR0+VxR1,    VxR0+VxR1, #4        // +4
30
31 // mit Spezialinstruktion GET_IDX_ORI (2 Instruktionen)
32 GET_IDX_ORI VxR0, VxR2, VxR4 // ori(dx[3:0], dy[3:0])
33 GET_IDX_ORI VxR1, VxR3, VxR5 // ori(dx[7:4], dy[7:4])
34
35 // mit X2-Modus für Spezialinstruktion GET_IDX_ORI (1 Instruktionen)
36 GET_IDX_ORI_X2 VxR0+VxR1, VxR2+VxR3, VxR4+VxR5 // ori(dx[3:0], dy[3:0]) | ori(dx[7:4],
37                                                 // dy[7:4])
```

**Abbildung 3.18:** Code-Beispiel für *getIdxOri*

Auf Basis der Größenverhältnisse der Inputoperanden zueinander (größer, kleiner oder gleich) wird eine unterschiedliche Konstante ins Zielregister geschrieben. Gleiche Funktionalität könnte auch hier mit *predicated execution* erzielt werden, allerdings mit deutlich mehr benötigten Instruktionen.

Gradientenbetrag und Winkel stehen zu diesem Zeitpunkt als 16-bit Subwords in Registern zur Verfügung. Im nächsten Schritt folgt die Histogrammgenerierung mittels dedizierter Koprozessoren. Dazu wird die 16×16 px Umgebung in ein 4×4 Raster eingeteilt, in dem die einzelnen Blöcke wieder aus 4×4 px bestehen (siehe Abb. 3.19). Für jeden Block wird ein Histogramm mit den acht Winkelbins erstellt, in denen die Gradientenbeträge aufaddiert werden. Aufgrund des Programmablaufs, der Limitierung in der Anzahl an Registern und der Datenabhängigkeiten von der Berechnung der Gradienten bis zur Histogrammerstellung werden die Histogramme der Blöcke *0*, *4*, *8* und *12* gleichzeitig generiert. Dazu wurden vier Koprozessoren entwickelt, die zur Berechnung der vier

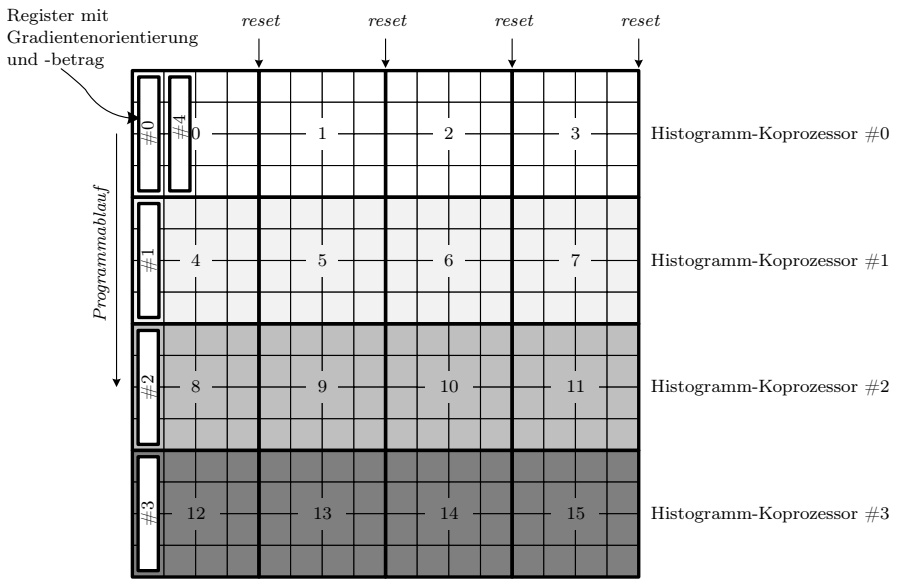



**Abbildung 3.19:** Datenabhängigkeit und Zuordnung der Histogramm-Koprozessoren zu den einzelnen Deskriptor-Teilhistogrammen. Die vier Koprozessoren arbeiten in vertikaler Richtung für jedes Teilhistogramm (bestehend aus vier Bildspalten) parallel, bevor sie zurückgesetzt werden (*reset*), um in horizontaler Richtung für weitere Teilhistogramme wiederverwendet werden zu können.

Histogramme einer Spalte des 4×4-Rasters gleichzeitig arbeiten (siehe Abb. 3.19). Der Aufbau eines Histogramm-Koprozessors ist in Abb. 3.20 dargestellt. Die Anzahl der Subwords sowie die Breite der Akkumulationsregister und die Anzahl der unterschiedlichen Bins sind dabei konfigurierbar, sodass der Koprozessor auch für andere Anwendungen eingesetzt werden könnte. In diesem Fall erhält ein einzelner Koprozessor jeweils einen Inputoperanden mit vier 16-bit Subwords der Gradientenbeträge (*Inkrement-Register*) und einen zweiten Operanden mit vier Winkelbins (*Bin-Index-Register*), welche im Intervall $[0, 7]$ liegen. Innerhalb von vier Takten werden die einzelnen Subwords in internen Akkumulationsregistern (16-bit ausreichend ohne Overflow) aufaddiert, wobei der Bin-Index als Adresse verwendet wird, um das jeweilige Akkumulationsregister auszuwählen. Ein globales Akkumulationsregister addiert alle Beträge unabhängig vom jeweiligen Bin auf. Nach den vier Takten, können neue Eingangsdaten angenommen werden. Pro Koprozessor wird dies mit anderen Daten viermal wiederholt, bis ein Histogramm eines 4×4-Blocks zur Verfügung steht. Die Akkumulationsregister werden ausgelesen und danach genullt (Koprozessor-Reset), um die Berechnung eines neuen Blocks durchführen zu können.

Die Aneinanderreihung der so erzeugten 16 Einzelhistogramme bildet den Deskriptor mit 128 Einträgen. Das globale Akkumulationsregister im Histogramm-Koprozessor dient

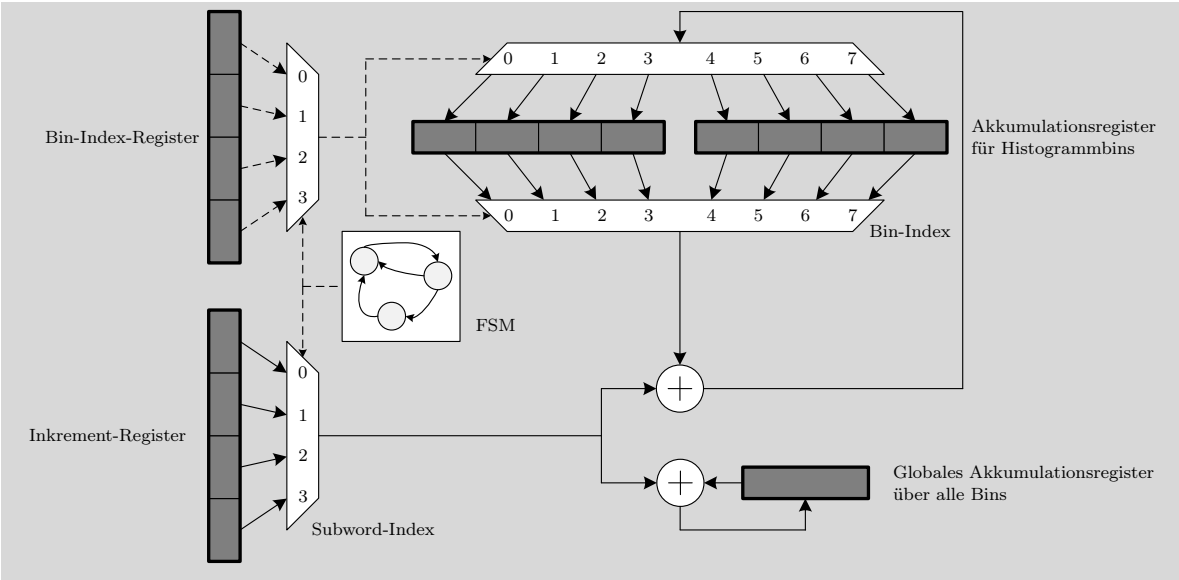



**Abbildung 3.20:** Aufbau des Histogramm-Koprozessors. Nach dem Schreiben des Bin-Index- und des Inkrement-Registers durchläuft die Schaltung vier Zustände für die vier Subwords in den Registern.

der finalen Normalisierung des Deskriptors. Dabei wird jeder Bin durch die Summe aller 128 Bins dividiert. Diese Summe steht direkt durch das globale Akkumulationsregister zur Verfügung. Ein weiterer Koprozessor implementiert die notwendige Division. Im Unterschied zur Division in der Subpixel-Lokalisierung werden hier vier vektorisierte Divisionen (16-bit Input-Dividenden) gleichzeitig durchgeführt. Dabei ist der Koprozessor komplett gepipelinet, sodass die nötigen $128/4 = 32$ Divisionen direkt in aufeinander folgenden Takten gestartet werden können. Da intern mit 32-bit gearbeitet wird, ergibt sich eine Latenz zu $1 + 32/x + 2$ Takten, bis das erste Ergebnis bereitsteht, wobei $x$ hier die Werte 2, 4, 8 und 16 annehmen und Hardwareaufwand gegen Latenz abtauschen kann. Die realisierbare Taktfrequenz bei hohen Werten von $x$ ist allerdings auch deutlich geringer.

Der fertige Deskriptor wird abschließend per DMA in den externen Speicher übertragen und der SDE kann nach einem neuen Merkmalskandidaten angefragt werden.

### Zusammenfassung

Der VLIW-Prozessor wird in dieser Arbeit als ASIP untersucht und daher in mehreren Konfigurationen betrachtet, welche in Tab. 3.4 aufgelistet sind. In Basiskonfiguration #*b*0 wurde die Clip-Max-Min-Unit entfernt, da diese in dieser Anwendung nicht gebraucht wird. Drei Spezialinstruktionen und die Koprozessoren sind bereits hier nötig, um die Anwendung bei der betrachteten Auflösung in Echtzeit ausführen zu können. In weiteren Konfigurationen wurden Arithmetic-Unit, Permute-Unit und Divisionskoprozessoren

verdoppelt (#*b*1-#*b*4). Dies erhöht die Flexibilität für den Instruction Scheduler zur Anordnung der Instruktionen, z. B. können dadurch zwei ADD-Instruktionen in den zwei Issue-Slots parallel ausgeführt werden. Somit kann eine höhere Code-Kompaktheit und damit ein höherer IPC erreicht werden, was zu einer schnelleren und effizienteren Abarbeitung des Programmcodes führt. Darüber hinaus wurde der VLIW um die anderen, in den vorangegangenen Abschnitten erläuterten Spezialinstruktionen erweitert (#*c*1, #*c*2). Danach wird noch der X2-Modus aktiviert (#*b*3*x*2, #*b*4*x*2, #*c*1*x*2, #*c*2*x*2), sodass alle doppelt vorhandenen FUs und Koprozessoren gleichzeitig mit nur einer X2-Instruktion angesteuert werden können (z. B. ADD_X2).

Zur Einordnung der Komplexität der einzelnen Einheiten sind die Logikressourcen in LUTs und Registern nach Place&Route gegeben (Zielplattform Xilinx ML605 Entwicklungsboard, Virtex-6 FPGA, 5 ns Clock Constraint). Der Hardwareaufwand für die applikationsspezifischen Erweiterungen ist insgesamt gering. Die einzige Ausnahme bildet der *division_2* Koprozessor, da der non-restoring Algorithmus dort komplett entrollt wurde. Zum Vergleich mit *division_1* (2×32-bit Division mit 8 Bits pro Takt) benötigt *division_2* (4×32-bit Division mit 16 Bits pro Takt, zwei entrollte Stufen) etwa die fünffache Menge an LUTs und die doppelte Menge an Registern. Die Ressourcen des VLIW-Prozessors insgesamt (default FUs enthalten) sind ebenfalls gelistet. Der VLIW besitzt einen Instruktionsspeicher von 32 KB und einen Datenspeicher von 64 KB.

Um einen Überblick über den Rechenaufwand der einzelnen SIFT-Teilschritte und die Effektivität der applikationsspezifischen Erweiterungen zu bekommen, ist in Tab. 3.5 die Anzahl an Instruktionen jedes Teilschrittes aufgeführt. Der Deskriptor hat den höchsten Anteil mit $322 + 12 \times 95 + 4 \times 124 = 1.958$ zur Laufzeit ausgeführten Instruktionen in Prozessorkonfiguration #*b*0. Zwischen #*b*3 und #*b*4 wurden die Koprozessoren für die Division dupliziert. Duplizieren hat keinen Einfluss auf die Menge der Instruktionen, sondern bietet ausschließlich die Möglichkeit, zwei Instruktionen, die auf dieselbe Hardware zugreifen, parallel auszuführen. #*c*2*x*2 führt die Deskriptorberechnung aufgrund der Erweiterungen mit 744 Instruktionen durch, wobei X2-Instruktionen hier nicht wie bei der IPC-Berechnung als zwei, sondern als eine einzige Instruktion gezählt werden.

Die Anzahl der Instruktionen lässt noch keinen direkten Rückschluss auf die benötigten Takte für die Ausführung zu, da diese sich erst aus dem Instruction Scheduling unter Berücksichtigung der Datenabhängigkeiten und der limitierten Hardwareressourcen ergeben. Bei maximaler Code-Kompaktheit (keine NOPs) und zwei Issue-Slots lässt sich für 744 Instruktionen eine untere Grenze von $744/2 = 372$ Takten abschätzen.

Die Teilschritte aus Tab. 3.5, die ein Merkmalskandidat tatsächlich durchläuft, sind datenabhängig. Beispielsweise wird die Schleife in der Subpixel-Lokalisierung 1-2× ausgeführt, und ein Kandidat kann dabei eventuell verworfen werden. Die übrigen Teile des Algorithmus wie z. B. die Deskriptorberechnung entfallen für diesen Kandidaten. Nur stabile Merkmale durchlaufen alle Teile des Algorithmus. Durch diese Dynamik liegt die durchschnittliche Anzahl an Gesamtinstruktionen pro Merkmalskandidat - auf ein ganzes Eingangsbild bzw. eine Bildsequenz bezogen - unter der Summe der einzelnen Teilschritte. Die Programmstruktur ist im Anhang in Abb. A.4 als Kontrollflussgraph dargestellt.

**Tabelle 3.4:** Übersicht über die applikationsspezifischen Erweiterungen und die betrachteten VLIW-Prozessorkonfigurationen (Ressourcen exemplarisch für Konfiguration #c2x2_3, siehe Tab. A.8 im Anhang)

| | Name | Typ | LUTs | Register | Betrachtete VLIW-Prozessorkonfigurationen | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | #b0 | #b1 | #b2 | #b3 | #b4 | #c1 | #c2 | #b3x2 | #b4x2 | #c1x2 | #c2x2 |
| Default FUs | AU | FU[1] | 485 | 0 | 1 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| | PERM | FU | 559 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| | SRU | FU | 1375 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | BLU | FU | 195 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | CMM | FU | 126 | 0 | - | - | - | - | - | - | - | - | - | - | - |
| | MMU | FU | 1273 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Applikationsspez. Erweiterungen | subExpandL | FU | 34 | 0 | - | - | - | - | - | 1 | 1 | - | - | 1 | 1 |
| | subExpandH | FU | 42 | 0 | - | - | - | - | - | 1 | 1 | - | - | 1 | 1 |
| | absAbsAdd | FU | 177 | 0 | - | - | - | - | - | 2 | 2 | - | - | 2 | 2 |
| | getIdxOri | FU | 152 | 0 | - | - | - | - | - | 2 | 2 | - | - | 2 | 2 |
| | getIdxPatch | FU | 116 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | sl64_24 | FU | -[4] | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | sl64_27 | FU | -[4] | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | mvImmediateCond_0 | FU | 14 | 0 | - | - | - | - | - | 1 | 1 | - | - | 1 | 1 |
| | mvImmediateCond_1 | FU | 14 | 0 | - | - | - | - | - | 1 | 1 | - | - | 1 | 1 |
| | mvImmediateCond_2 | FU | 14 | 0 | - | - | - | - | - | 1 | 1 | - | - | 1 | 1 |
| | mvImmediateCond_3 | FU | 14 | 0 | - | - | - | - | - | 1 | 1 | - | - | 1 | 1 |
| | division_1[5] | CU[2] | 966 | 438 | 1 | 1 | 1 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |
| | division_2[6] | CU | 4829 | 972 | 1 | 1 | 1 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |
| | histogramm | CU | 417 | 312 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| | X2-Modus | | | | - | - | - | - | - | - | - | ✓ | ✓ | ✓ | ✓ |
| | Baseline VLIW[3] | | 44096 | 12040 | | | | | | | | | | | |

[1] Functional Unit (innerhalb der Pipeline)

[2] Coprocessor Unit (Kopplung durch LOAD/STORE Operationen)

[3] Nach Place & Route für 5 ns Clock Constraint (13.800 Logic Slices, 16 DSPs, 2368 LUTRAMs, 16 BRAMs Datenspeicher, 8 BRAMs Instruktionsspeicher)

[4] Shift um eine Konstante wird bei der Synthese durch Verdrahtung ersetzt und benötigt deshalb keine Ressourcen

[5] Konfiguration mit 3+4=7 Takte Latenz bis das Ergebnis bereitsteht, Ressourcen verringern sich bei Latenzerhöhung

[6] Konfiguration mit 3+2=5 Takte Latenz bis das Ergebnis bereitsteht, Anzahl Register erhöht sich bei Latenzerhöhung signifikant durch gepipelinete Implementierung der Division

**Tabelle 3.5:** Statische Reduktion der Instruktionen durch Spezialisierung des Prozessors und Verwendung des X2 instruction merging (X2-Modus). Der Dynamik-Faktor gibt an, wie oft ein Teilschritt bei realer Ausführung des Programms wiederholt wird (z. B. aufgrund von Programmschleifen, vgl. Abb. A.4 im Anhang).

| | Konfiguration | Anzahl statische Instruktionen pro SIFT-Teilschritt | | |
|---|---|---|---|---|
| | | Subpixel-Lokalisierung[1] | Stabilitätsprüfung | Deskriptorberechnung[2] |
| | #*b*0 | 62 / 213 | 74 | 322 / 95 / 124 |
| | #*b*1 | 62 / 213 | 74 | 322 / 95 / 124 |
| | #*b*2 | 62 / 213 | 74 | 322 / 95 / 124 |
| | #*b*3 | 62 / 213 | 74 | 322 / 95 / 124 |
| | #*b*4 | 62 / 213 | 74 | 322 / 95 / 124 |
| | #*c*1 | 62 / 213 | 74 | 294 / 47 / 76 |
| | #*c*2 | 62 / 213 | 74 | 294 / 47 / 76 |
| X2-Modus | #*b*3*x*2 | 41 / 182 | 71 | 258 / 59 / 79 |
| X2-Modus | #*b*4*x*2 | 41 / 182 | 71 | 194 / 59 / 79 |
| X2-Modus | #*c*1*x*2 | 41 / 182 | 71 | 232 / 31 / 51 |
| X2-Modus | #*c*2*x*2 | 41 / 182 | 71 | 168 / 31 / 51 |

[1] Dynamik-Faktor: 1× / 1-2×
[2] Dynamik-Faktor: 1× / 12× / 4×

## 3.6 Taktgenerierung

Die Taktgenerierung erfolgt im vorliegenden FPGA durch den *Mixed-Mode Clock Manager (MMCM)* [Xil14a]. Dieser enthält im Kern eine PLL, welche aus einem externen Eingangstakt über mehrere Ausgänge bis zu sieben Ausgangstakte generieren kann. Dabei unterliegt die Taktgenerierung folgenden Formeln:

$$f_{VCO} = f_{clkin} \cdot \frac{M}{D} \tag{3.19}$$

$$f_{out,i} = f_{clkin} \cdot \frac{M}{D \cdot O_i}, \tag{3.20}$$

mit $M$, $D$ und $O_i$ als ganzzahlige Parameter, aus denen sich die Taktteilung ergibt. Die generierten Ausgangstakte $f_{out,i}$ an einem MMCM haben alle dieselben Werte für $M$ und $D$ und unterscheiden sich lediglich im jeweiligen Wert für $O_i$. Der Eingangstakt $f_{clkin}$ hat im vorliegenden Fall eine Frequenz von 200 MHz. Die VCO-Frequenz $f_{VCO}$ muss laut Datenblatt im Bereich zwischen 600 MHz und 1.200 MHz liegen. Zudem ergeben sich weitere Randbedingungen wie die maximale Bitbreite bzw. das Intervall, in dem die Parameter liegen müssen, und die Frequenzen am Phasen-Frequenz-Detektor innerhalb der PLL. Dadurch kann zwar eine Vielzahl unterschiedlicher Ausgangstakte eingestellt werden, jedoch sind diese nicht beliebig wählbar und innerhalb eines MMCM auch nicht komplett unabhängig voneinander.

Um den Durchsatz des VLIW-Prozessors und des SDE zu kontrollieren und diese bei unterschiedlichen Performance-Anforderungen an das System optimal aufeinander

abstimmen zu können, befinden sich die beiden Komponenten in zwei getrennten Taktdomänen. Bei der Verwendung von TDM×2, TDM×3 und TDM×6 werden die DSP-Slices für die Berechnung der Gauß-Pyramide noch von einem dritten Takt gesteuert, der ein Vielfaches des SDE-Takts darstellt. Alle drei Taktsignale werden von demselben MMCM generiert. Innerhalb des Systems existieren noch weitere Takte für den Bus (100 MHz) und für Ethernet- und Speicher-Controller, welche von einem zweiten MMCM ausgehen.

Mit diesem Aufbau ist es möglich, die Takte zur Generierung der SIFT-Merkmale komplett unabhängig vom Rest des Systems einzustellen. Der MMCM ermöglicht dabei nicht nur die Konfiguration der Taktausgänge zur Design- und Synthesezeit, sondern auch dynamisch während der Laufzeit des Systems [Xil10]. Der Zugriff auf die veränderbaren Register erfolgt über einen speziellen *Dynamic Reconfiguration Port (DRP)* am MMCM. Zur vollständigen Rekonfiguration müssen dabei 15 Einzelregister ausgelesen, verändert und in den MMCM zurückgeschrieben werden. Die Parameter $M$, $D$ und $O$ sind in den Registern in kodierter Art und Weise hinterlegt. Es werden C-Funktionen zur Verfügung gestellt, die bei der Generierung der Registereinträge helfen. Während der Rekonfiguration befindet sich der MMCM im Reset, d. h. alle Taktausgänge sind deaktiviert. Nachdem alle Rekonfigurationsregister geschrieben wurden, wird der Reset aufgehoben, und ein *lock*-Signal am MMCM zeigt an, dass die PLL eingeschwungen ist und die neuen Takte zur Verfügung stehen. Eine erneute Rekonfiguration wäre nun möglich. Im Datenblatt ist eine maximale Latenz bis zum Setzen des *lock*-Signals von 100 $\mu$s spezifiziert. In der Praxis liegt der Wert deutlich darunter.

Um die Rekonfigurationsmöglichkeiten des MMCM zu nutzen, wurde eine Schaltung entworfen, mit der der MMCM sowohl vom Host-PC als auch vom VLIW-Prozessor kontrolliert werden kann. Der Aufbau ist in Abb. 3.21 dargestellt. Zunächst werden offline die Rekonfigurationsregister für verschiedene Arbeitspunkte des MMCM, d. h. unterschiedliche Konfigurationen der Taktausgänge, berechnet. Die Registerinhalte werden im Konfigurationsspeicher abgelegt, nachdem das FPGA eingeschaltet wurde. Die Steuerung der Rekonfiguration übernimmt eine FSM, die mit Bustakt arbeitet und dadurch unabhängig von der Rekonfiguration weiterläuft. Die FSM erwartet ein *start*-Signal und die ID des Zielarbeitspunktes. Diese können sowohl über den Bus vom Host-PC als auch vom VLIW-Prozessor geschrieben werden. Danach führt die FSM die Rekonfiguration durch und schreibt Status und Dauer der Rekonfiguration in zwei Register.

Wenn der VLIW-Prozessor die Rekonfiguration startet, schreibt er zunächst das *start*-Signal und die ID und fragt danach in einer Schleife den Zustand des MMCM ab. Ab dem Zeitpunkt, ab dem sich der MMCM im Reset befindet, friert der Program Counter ein und der VLIW-Prozessor steht still. Steht wieder ein Takt zur Verfügung, so arbeitet der VLIW-Prozessor an derselben Programmstelle weiter und kann aufgrund des veränderten Status die Schleife verlassen. Während der Rekonfiguration befindet sich der SDE im *IDLE*-Zustand. Die Rekonfiguration erfolgt immer vor der Verarbeitung eines neuen Eingangsbildes. Insgesamt dauert die Rekonfiguration weniger als 2 $\mu$s, zuzüglich der Zeit bis zum Setzen des *lock*-Signals am MMCM. Die Schaltung ermöglicht es, dynamisch den Takt zwischen zwei Eingangsbildern zu ändern und damit auf datenabhängige

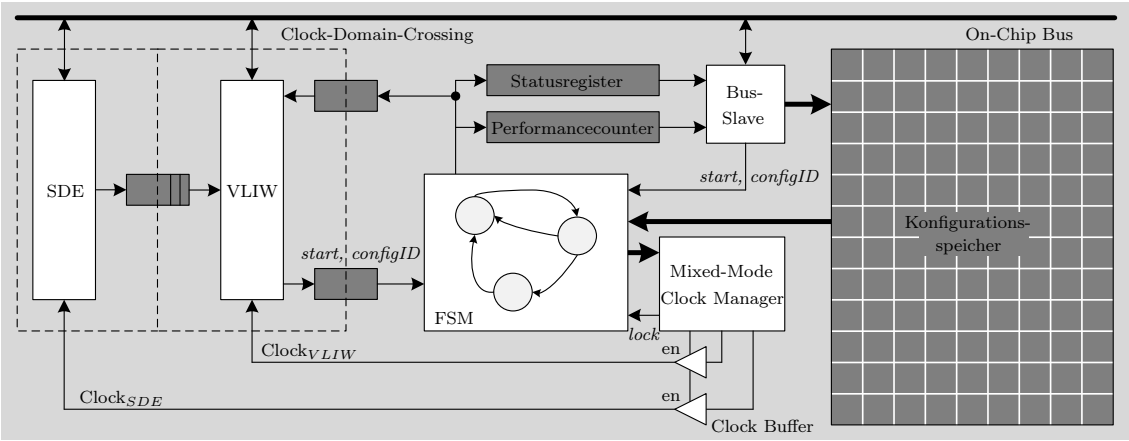



**Abbildung 3.21:** Rekonfigurationsschaltung des MMCM innerhalb der Bus-Taktdomäne

Performance-Anforderungen flexibel zu reagieren.

Um Daten zwischen Taktdomänen sicher zu transferieren (Clock-Domain-Crossing), muss berücksichtigt werden, dass Flip-Flops einen instabilen Zustand erreichen können, wenn sich die Daten am Eingang während der Setup- bzw. Hold-Zeiten verändern. Dieser Zustand wird nach [Chu06] als *metastabiler* Zustand bezeichnet. Besonders bei Übergängen zwischen komplett asynchronen Takten müssen spezielle Vorkehrungen gegen Metastabilität und zur Datensynchronisation getroffen werden. Hier werden beide genutzten MMCMs des Systems mit demselben externen Eingangstakt gespeist und insbesondere VLIW und SDE arbeiten mit voneinander abgeleiteten Takten desselben MMCM, wodurch die Wahrscheinlichkeit für Instabilitäten von vornherein reduziert ist. Darüber hinaus wurden für Clock-Domain-Crossings spezielle Schaltungen implementiert. Zum einen wurde die Wahrscheinlichkeit für metastabile Zustände durch Mehrfachregistrierung eines Datenwortes signifikant verringert. Zum anderen wurde für Register, die mit einem Taktsignal geschrieben und mit einem anderen Taktsignal gelesen werden, auf beiden Seiten eine FSM implementiert, die ein Handshake-Protokoll verfolgt [Chu06]. Dadurch ist sichergestellt, dass auf der lesenden Seite die Daten sicher übernommen wurden, bevor auf der schreibenden Seite neue Daten angelegt werden können. Dies gilt sowohl für Register zwischen Bus und VLIW, als auch zwischen SDE und VLIW.

Asynchrone FIFOs werden von Xilinx direkt unterstützt und als Makro zur Verfügung gestellt [Xil14c]. Clock-Domain-Crossing ist dort enthalten. Wenn mehrere FIFOs, bestehend aus einem einzelnen Block RAM, für eine größere FIFO-Tiefe zusammengeschaltet werden, dann muss entweder der Lese- oder der Schreibtakt als interner Takt verwendet werden, abhängig davon, welche Taktfrequenz höher ist. Da in diesem System beide Szenarien (Lese- bzw. Schreibtakt höher) in einem einzigen Bitstream zur Laufzeit unterstützt werden sollen, kommen spezielle Clockbuffer zum Einsatz, die wie ein 2:1-Multiplexer funktionieren und ein Umschaltsignal entgegennehmen können. Somit ist die Funktionalität der Schaltung auch beim dynamischen Rekonfigurieren gewährleistet.

**Abbildung 3.22:** Verwendete Testbilder mit Störeffekten aus dem Oxford Datensatz. Von links nach rechts: *wall* (Blickpunkt), *boat* (Zoom und Rotation), *leuven* (Belichtung), *trees* (Blur), *graffitiRotation* (Rotation), *graffitiScale* (Skalierung) [Mik05a]

## 3.7 Algorithmische Charakterisierung

In den vorangegangenen Abschnitten dieses Kapitels wurde die entwickelte FPGA-Architektur zur Bildmerkmalsextraktion beschrieben. Als Referenzanwendung dient der SIFT-Algorithmus nach [Low04b]. Um einen hohen Durchsatz zu erreichen, wurde die Komplexität des Algorithmus für die FPGA-Implementierung begrenzt (siehe Abschnitt 3.2). In diesem Abschnitt sollen dadurch entstandene Auswirkungen auf die algorithmische Leistungsfähigkeit anhand von Störeffekten im Bild und einer realen Anwendung charakterisiert werden.

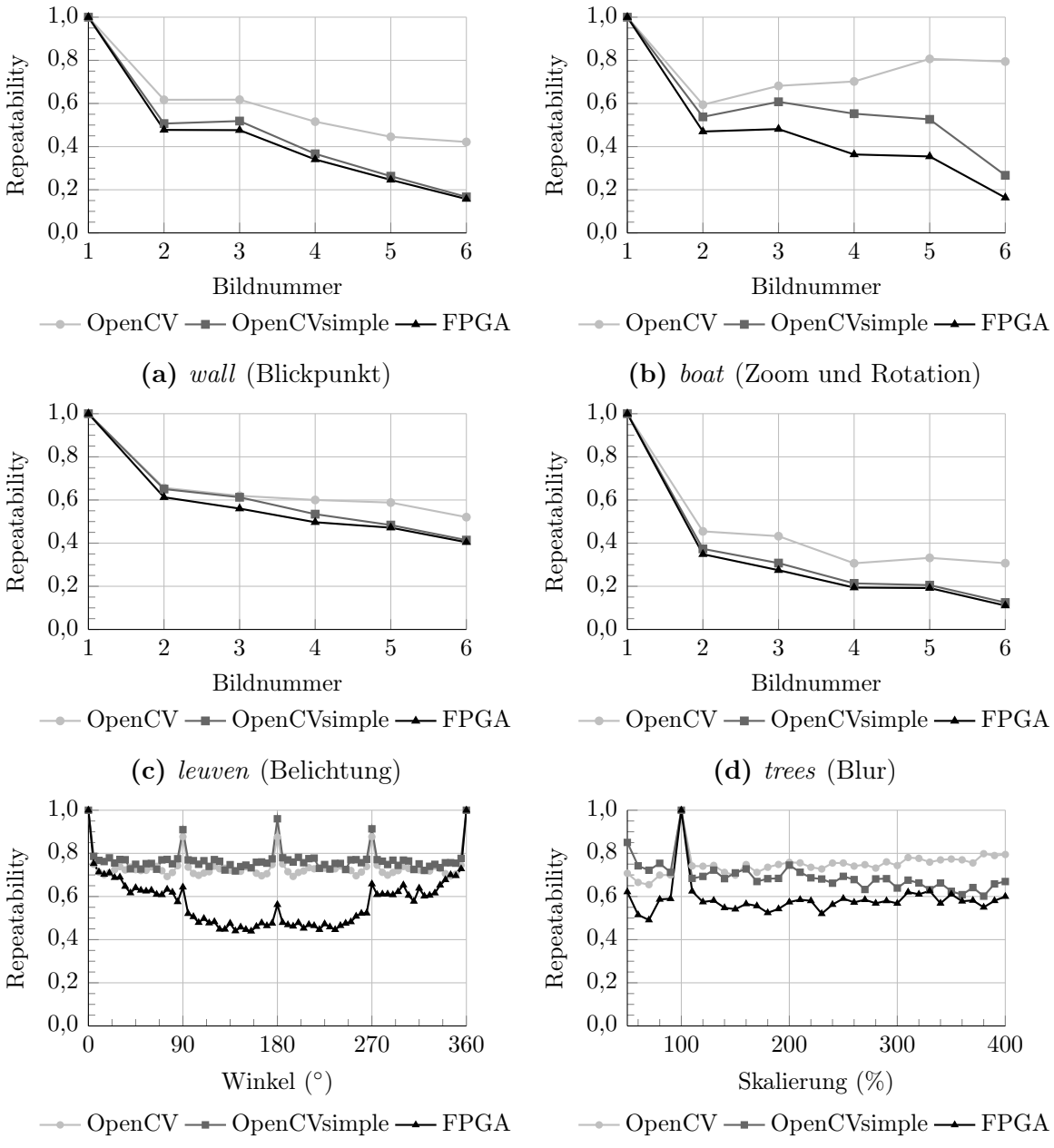
### 3.7.1 Störeffekte

Ein idealer Merkmalsdetektor mit einem zugehörigen idealen Merkmalsdeskriptor sollte unabhängig von einem vorliegenden Störeffekt im Bild in der Lage sein, Merkmale zuverlässig zu detektieren und eindeutig zuzuordnen, sofern diese im gleichen Bildbereich sichtbar sind. Die Distanz zwischen den Deskriptoren einer Merkmalskorrespondenz steigt allerdings, je deutlicher die Transformation zwischen den zu vergleichenden Bildbereichen ist, was eine korrekte Zuordnung erschwert [Men18]. Zur Untersuchung von Störeffekten werden hier Testbilder des Oxford-Datensatzes verwendet (siehe Abb. 3.22) [Mik05a], welcher auch Homographien enthält, die die verschiedenen perspektivischen Änderungen zwischen den Bildern innerhalb einer Testbildsequenz beschreiben. Die Untersuchung orientiert sich hierbei methodisch am Vorgehen in [Men18]. Die Bildsequenzen sind nummeriert und der Einfluss des jeweiligen Störeffekts steigt mit steigender Bildnummer. Als Bewertungsmetriken werden die *Repeatability* für den Detektor und die *Precision* für den Deskriptor eingesetzt, welche in Abschnitt 2.2.2 vorgestellt wurden. Die Repeatability verwendet einen Schwellwert von zwei Pixeln. Für die Precision muss eine Überlappung der Bildbereiche von mindestens 40% vorliegen, um als *true positive* akzeptiert zu werden. Untersucht werden drei verschiedene Algorithmen zur Merkmalsextraktion. *OpenCV* extrahiert SIFT-Merkmale nach der Referenzimplementierung aus der Open Source Computer Vision Library OpenCV (v3.0.0) [Ope15] unter Standardeinstellungen. *Open-*

*CVsimple* verwendet ebenfalls die OpenCV-Implementierung, allerdings ohne die Schritte der Hochskalierung der Eingangsbilddaten, wodurch weniger Merkmale gefunden werden, und der Berechnung der Hauptorientierung, wodurch der Deskriptorbereich nicht gedreht wird und damit die Rotationsinvarianz entfällt. *FPGA* ist schließlich derjenige Algorithmus, der als FPGA-Architektur, wie in den vorangegangenen Abschnitten dargestellt, implementiert wurde. Die algorithmischen Veränderungen der FPGA-Implementierung gegenüber der SIFT-Referenz wurden in Abschnitt 3.2 dargestellt.

Abb. 3.23 zeigt die Repeatability für die ausgewählten Testfälle als Charakterisierung des Merkmalsdetektors. Bei Veränderungen des Blickpunktes (Abb. 3.23a) liegen *OpenCVSimple* und *FPGA* gleichauf und verschlechtern sich in gleichem Maße im Bezug auf die wiederholte Detektion von Merkmalen. *OpenCV* erfährt ebenfalls eine Verschlechterung, liegt dabei aber bei insgesamt höheren Werten. Dadurch ergibt sich der Schluss, dass zwar die geringere Merkmalsanzahl durch die fehlende Hochskalierung einen Einfluss auf die Qualität des Detektors hat, nicht jedoch die Überführung des Algorithmus in Festkomma-Arithmetik und alle weiteren Veränderungen des Detektors in der FPGA-Architektur im Vergleich zur Software-Referenz. Dieselben Tendenzen sind bei veränderter Bildbelichtung (Abb. 3.23c) und Weichzeichnung (engl. Blur) des Bildes (Abb. 3.23d) zu beobachten. Bei Zoom und Rotation (Abb. 3.23b) liegt die Repeatability bei *FPGA* im Vergleich zu *OpenCVsimple* um etwa 30% niedriger. In Abb. 3.23e zeigt sich eine Anfälligkeit von *FPGA* gegenüber Rotation, sodass sich im Maximum etwa nur halb so viele Merkmale wiederholen als in den Referenzimplementierungen, die beide etwa gleich liegen. Mit Blick auf die Skalierung (Abb. 3.24f) zeigen alle drei Implementierungen etwa das selbe Ergebnis, was im hohen Aufwand der FPGA-Architektur bei der Implementierung der Gauß-Pyramide zu begründen ist.

In Abb. 3.24 sind die Untersuchungen zum Merkmalsdeskriptor anhand der Precision dargestellt. Auch hier liegen die Implementierungen in Hinblick auf Blickpunkt (Abb. 3.24a), Belichtung (Abb. 3.24c) und Weichzeichnung (Abb. 3.24d) etwa gleich, sodass davon auszugehen ist, dass die fehlende Gauß-Gewichtung und geänderte Akkumulation und Normalisierung des Deskriptors in *FPGA* die algorithmische Qualität nicht signifikant beeinflusst. Bei Zoom und Rotation (Abb. 3.24b) zeigt sich, dass hier wie erwartet die Rotationsinvarianz essentiell ist. Ohne eine gedrehte Berechnung des Deskriptors lassen sich hier keine korrekten Korrespondenzen finden. Dies wird in der systematischen Betrachtung der Rotation in Abb. 3.24e noch deutlicher. Alleine *OpenCV* zeigt über beliebige Drehwinkel gute Ergebnisse. Die anderen Implementierungen können ab etwa 20° keine Korrespondenzen mehr finden. Der Einfluss der Skalierung bei der Korrespondenzsuche (Abb. 3.24f) ist auch beim vereinfachten Deskriptor in *FPGA* ab einer Skalierung von 200% vergleichbar mit der Referenz und darunter etwas schlechter.




**(a)** *wall* (Blickpunkt)

**(b)** *boat* (Zoom und Rotation)

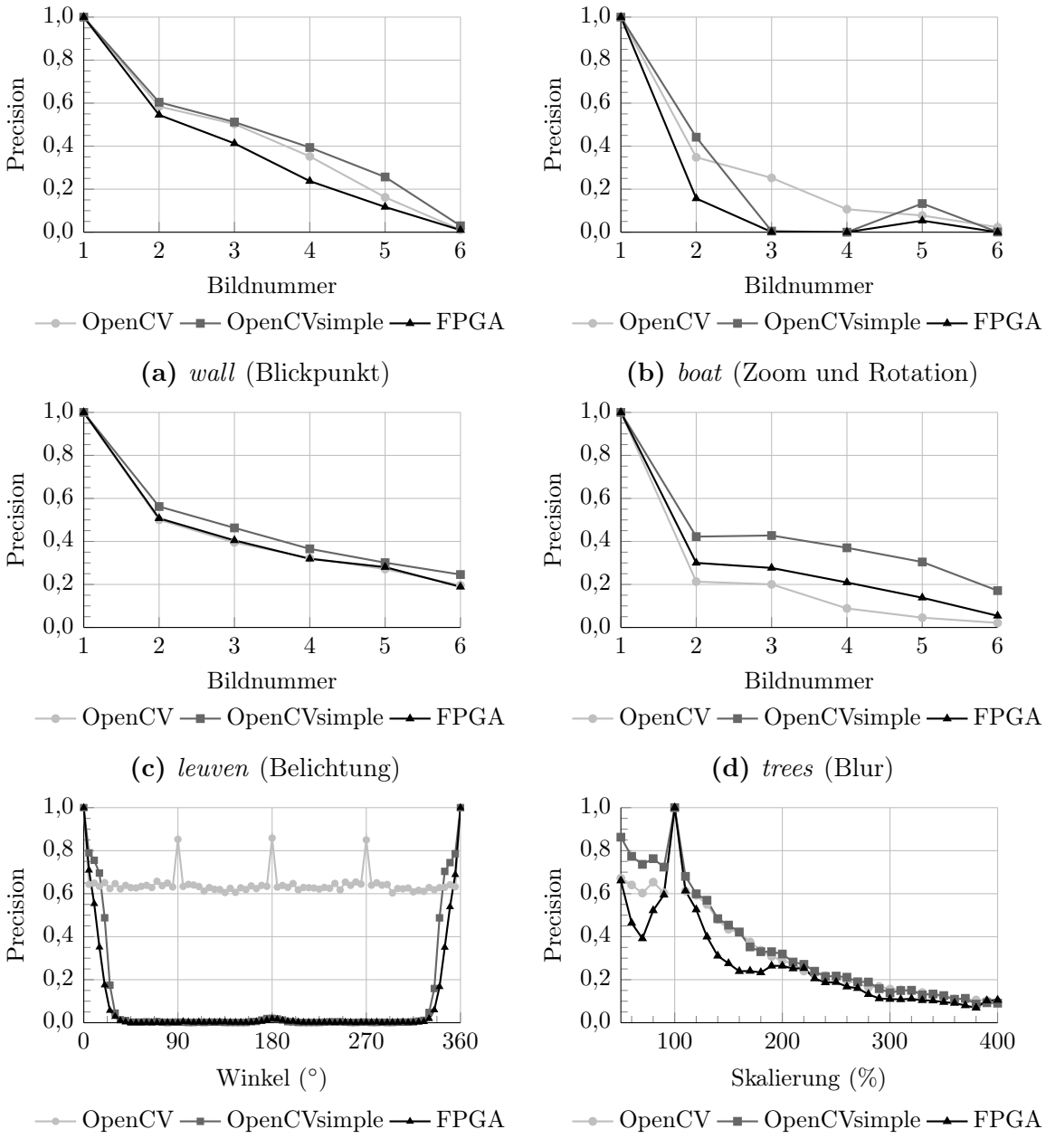
**(c)** *leuven* (Belichtung)

**(d)** *trees* (Blur)

**(e)** *graffitiRotation* (Rotation)

**(f)** *graffitiScale* (Skalierung)

**Abbildung 3.23:** *Repeatability* auf Testbildern des Oxford Datensatzes [Mik05a]




**(a)** *wall* (Blickpunkt)

**(b)** *boat* (Zoom und Rotation)

**(c)** *leuven* (Belichtung)

**(d)** *trees* (Blur)

**(e)** *graffitiRotation* (Rotation)

**(f)** *graffitiScale* (Skalierung)

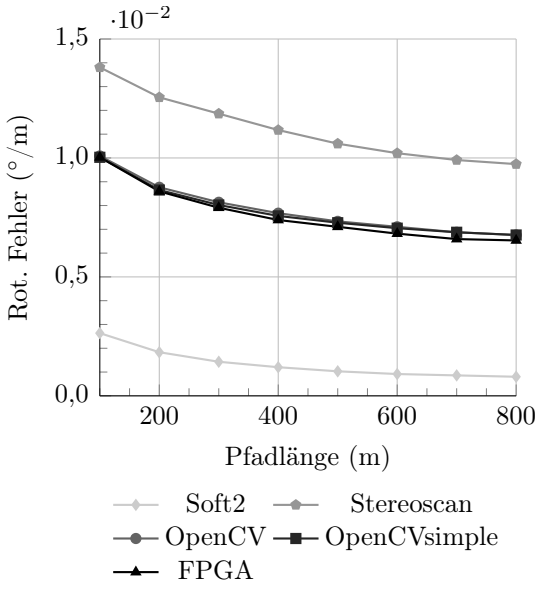
**Abbildung 3.24:** *Precision* auf Testbildern des Oxford Datensatzes [Mik05a]

### 3.7.2 Merkmalsbasierte Eigenbewegungsschätzung

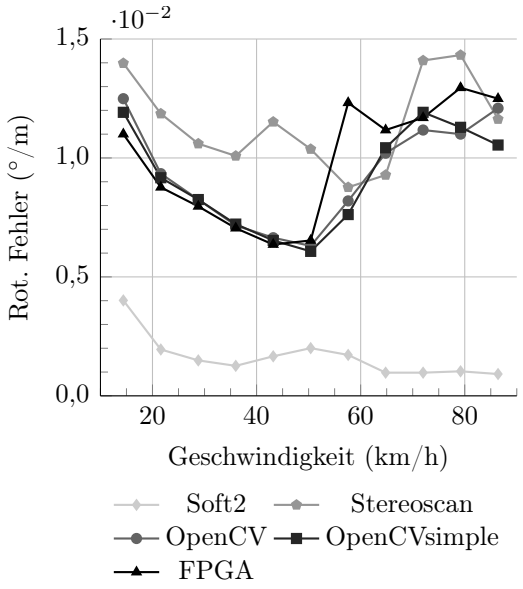
Eine Charakterisierung hinsichtlich elementarer Bildeffekte wie im vorangegangenen Abschnitt gibt Rückschlüsse über Anzahl und Qualität der Pixelkorrespondenzen unter bestimmten Randbedingungen. Nichtsdestotrotz lässt sich daraus noch keine Aussage über die algorithmische Leistungsfähigkeit im Kontext einer realen Anwendung treffen, in der die extrahierten Merkmale lediglich als Eingangsdaten dienen. Selbst weniger Korrespondenzen bei schlechterer Qualität können noch genug Informationen liefern, sodass sie in einem robusten Verfahren zuverlässig eingesetzt werden können. Daher wird hier die in Abschnitt 2.2.2 ab Seite 18 beschriebene merkmalsbasierte Eigenbewegungsschätzung als Anwendung eingesetzt, um die Merkmale zu untersuchen, die mit der in diesem Kapitel beschriebenen FPGA-Architektur extrahiert wurden.

Als Quelle für Testsequenzen wird der öffentlich zugängliche Visual Odometry Datensatz für Fahrerassistenzanwendungen aus der KITTI Vision Benchmark Suite sowie das enthaltene Bewertungsframework verwendet [Gei13]. Dieser besteht aus 22 Referenzfahrten (Wohngebiet, Stadt, Autobahn), wovon bei elf von ihnen auch Ground-Truth-Informationen der Trajektorie (rotatorische und translatorische Bewegungsparameter) sowie GPS-Positionen bekannt sind. Der Datensatz enthält Bildsequenzen, die mit einer Stereokamera bei einer Basislinie von 54 cm und einer Bildrate von zehn Bildern pro Sekunde aufgenommen wurden. Ein exemplarisches Bild jeder Sequenz ist im Anhang in Abschnitt A.5 dargestellt. Als Eingangsbilder für die Merkmalsextraktion dienen synchronisierte, rektifizierte Stereobildpaare in 8-bit Graustufen und einer Auflösung von 1240×376 bzw. 1226×370 px. Für die elf Referenzfahrten mit Ground-Truth-Daten werden dabei anhand der Bildsequenz Merkmale extrahiert und die Trajektorie geschätzt. Um die Rotations- und Translationsfehler als Funktion der gefahrenen Fahrstrecke darzustellen, werden in 100 m Schritten für alle 100 m bis 800 m langen Teilabschnitte der Trajektorien die Fehler bestimmt und über die elf Sequenzen gemittelt. Da aufgrund der GPS-Daten auch die Fahrtgeschwindigkeiten bekannt sind, können die Fehler in den Teilabschnitten auch über die Geschwindigkeit aufgetragen werden. Für diesen Datensatz und mit diesen Metriken existiert ein Online-Ranking [Kit19]. Dort lassen sich Verfahren aus der Literatur miteinander vergleichen, und die jeweiligen Schätzfehler sind frei zugänglich.

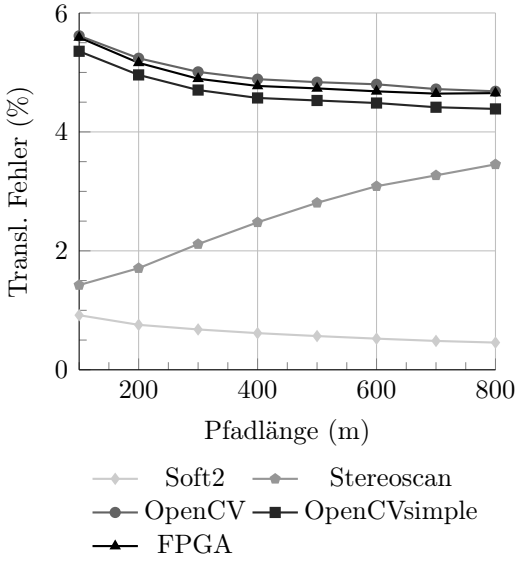
Abb. 3.25 zeigt einen Vergleich zwischen fünf verschiedenen Eigenbewegungsschätzern. Die Rohdaten zu Abb. 3.25 sind im Anhang A.4.1 in Tab. A.1 und Tab. A.2 aufgelistet. *Soft2* [Cvi17] ist zum Zeitpunkt der Ausarbeitung im Online-Ranking unter den führenden kamerabasierten Verfahren. *Stereoscan* [Gei11] ist der Referenzalgorithmus für den Eigenbewegungsschätzer aus der Literatur, an dem sich die Implementierung des in dieser Arbeit verwendete Schätzers orientiert. *OpenCV* verwendet diese Eigenimplementierung der Eigenbewegungsschätzung mit SIFT-Merkmalen als Eingangsdaten, die mit der Open Source Computer Vision Library OpenCV (v3.0.0) [Ope15] unter Standardeinstellungen extrahiert wurden. *OpenCVsimple* verwendet ebenfalls die Eigenimplementierung des Eigenbewegungsschätzers mit OpenCV SIFT-Merkmalen als Eingangsdaten, aller-



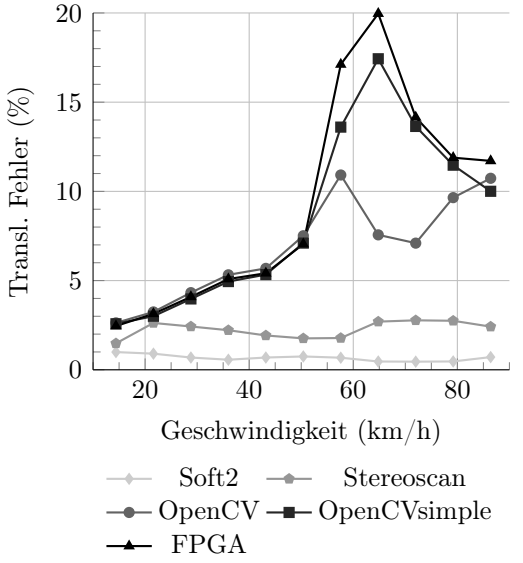

**(a)** Rot. Fehler über die Strecke




**(b)** Rot. Fehler über die Geschwindigkeit




**(c)** Transl. Fehler über die Strecke




**(d)** Transl. Fehler über die Geschwindigkeit

**Abbildung 3.25:** Algorithmischer Vergleich merkmalsbasierter Eigenbewegungsschätzungen nach den KITTI-Metriken [Gei12]

dings ohne die Schritte der Hochskalierung der Eingangsbilddaten, wodurch weniger Merkmale gefunden werden, und der Berechnung der Hauptorientierung, wodurch der Deskriptorbereich nicht gedreht wird (Rotationsinvarianz entfällt). *FPGA* ist derselbe Eigenbewegungsschätzer mit SIFT-Merkmalen als Eingangsdaten, die mit der in diesem

Kapitel beschriebenen FPGA-Architektur extrahiert wurden.

Gezeigt sind der rotatorische Fehler in Grad pro Meter und der translatorische Fehler in Prozent über der geschätzten Strecke und über der Geschwindigkeit. Insbesondere der Vergleich zwischen *OpenCV*, *OpenCVsimple* und *FPGA* ist entscheidend, da alle den gleichen Eigenbewegungsschätzer verwenden und daher dort der Einfluss der algorithmischen Vereinfachungen der FPGA-Merkmale auf die Eigenbewegungsschätzung deutlich wird. Die anderen beiden Verfahren sollen der Einordnung der Zahlen zur Literatur mit speziell auf Eigenbewegungsschätzung optimierten Algorithmen dienen.

Beim rotatorischen und translatorischen Fehler über der Strecke liegen die OpenCV-Merkmale und die FPGA-Merkmale nahezu gleichauf (siehe Abb. 3.25a und 3.25c). Der Restfehler ist somit für diese Anwendung merkmalsunabhängig, bzw. lässt sich auch mit Merkmalen aus der algorithmisch wesentlich komplexeren SIFT-Referenzimplementierung aus OpenCV nicht verringern.

Höhere Fahrgeschwindigkeiten ab etwa 50 km/h bei der konstanten Aufnahme-Bildrate des Datensatzes von 10 fps führen zu großen projektiven Veränderungen (Änderungen im Blickwinkel) innerhalb der Szene von Bild zu Bild. Das Verhältnis zwischen Fahrgeschwindigkeit und Bildrate entspricht dabei der zurückgelegten Distanz zwischen zwei aufgenommenen Bildern. Diese liegt in diesem Beispiel bei mehr als 1,39 m. Durch die fehlende Hochskalierung des Eingangsbildes und fehlende Rotationsinvarianz in der FPGA-Implementierung nehmen hier Anzahl und Qualität der Korrespondenzen deutlich ab, und die Eigenbewegungsschätzung, insbesondere die der Translation, wird ungenau (siehe Abb. 3.25b und 3.25d). Diese Schwierigkeiten waren schon nach der Untersuchung der Störeffekte im vorherigen Abschnitt zu erwarten. Wären die Sequenzen mit höheren Fahrgeschwindigkeiten mit einer höheren Bildrate als 10 fps aufgenommen worden, so wäre nicht nur die zurückgelegte Distanz zwischen zwei zeitlich aufeinander folgenden Bildern geringer, sondern auch die damit verbundenen Änderungen im Blickwinkel auf die Szene und es wären dadurch bessere Schätzungen der Trajektorie zu erwarten.

Allgemein lässt sich daraus folgern, dass das vorliegende Verhältnis zwischen Fahrgeschwindigkeit und Bildrate, also die Distanz zwischen zwei aufeinander folgenden Bildern, für die Eigenbewegungsschätzung einen großen Einfluss auf die Wahl eines geeigneten Algorithmus zur Merkmalsextraktion hat. Bei einem großen Verhältnis ist eine höhere Robustheit des Algorithmus erforderlich.

In der in dieser Arbeit entwickelte FPGA-Implementierung der Merkmalsextraktion wird durch algorithmische Vereinfachungen *Komplexität* gegen *Durchsatz* abgetauscht. Es ist dadurch möglich, durch eine hohe Bildrate die Distanz zwischen zwei Bildern gering zu halten und fehlende algorithmische Robustheit zu kompensieren. Die Bildrate in diesem Benchmark ist konstant bei 10 fps, wodurch dieser Kompensation nicht zum Tragen kommen kann. In einer realen Anwendung würde eine höhere Bildrate als 10 fps gewählt werden können.

# 4 Architekturmodellierung

Nachdem im vorangegangenen Kapitel die FPGA-Architektur zur Bildmerkmalsextraktion erläutert wurde, wird in diesem Kapitel ein Modellierungsansatz dieser Architektur präsentiert, in dessen Kern der Merkmals-FIFO als Warteschlange steht. Zunächst wird in Abschnitt 4.1 auf die Parametrierung des Gesamtsystems sowie auf die angewendete Methodologie zur Generierung der Modelle eingegangen. Anschließend wird neben der Laufzeit zur Verarbeitung eines Bildes (Abschnitt 4.2) auch ein Modell für die entstehende Verlustleistung (Abschnitt 4.3) anhand einer Beispielkonfiguration des Systems erarbeitet. Durch die Kombination beider Modelle kann der Energiebedarf des Systems modellgestützt ermittelt werden. Abschnitt 4.4 fasst das Vorgehen zur Modellableitung für neue Systemkonfigurationen zusammen.

## 4.1 Parametrierung und Methodologie

Die veränderlichen Parameter der FPGA-Architektur werden im Folgenden *Designparameter* genannt. Diese sind zu unterscheiden von den zu modellierenden Größen, die als *Zielindikatoren* bezeichnet werden (vgl. Abschnitt 2.1.1). Die Zielindikatoren sind im hier vorliegenden Fall Laufzeit und Verlustleistung, sowie sich daraus ergebend die Energie.

Die hier betrachteten Designparameter, die als Eingangsgrößen für das System und somit auch für die abgeleiteten Modelle dienen, sind in Tab. 4.1 dargestellt und nach SDE- bzw. VLIW-Zugehörigkeit unterschieden. Diejenigen Parameter, die das Architekturdesign betreffen, können ausschließlich zur Synthesezeit verändert werden und bestimmen Größe und Timing-Verhalten der FPGA-Architektur. $n_c$ steht für die Anzahl der Merkmalskandidaten im Eingangsbild und ist damit datenabhängig. Dadurch ergibt sich eine nicht-deterministische Veränderung der Rechenlast, was durch das Modell abbildbar sein muss. Des Weiteren bestimmen die verwendeten Taktfrequenzen des SDE und des VLIW-Prozessor signifikant das Systemverhalten. Die Takte können dynamisch zur Laufzeit des Systems durch den in Abschnitt 3.6 beschriebenen Mechanismus verändert werden.

Darüber hinaus ergeben sich die internen *Modellparameter*, die das Modellverhalten bestimmen und während der Architekturmodellierung ermittelt werden müssen. Die Anzahl dieser Parameter ist abhängig vom gewählten Modellierungsansatz. Zur Ermittlung der Modellparameter werden hier methodisch zwei Herangehensweisen kombiniert. Zum einen werden Modellgleichungen aus der Schaltungsbeschreibung der Hardwarearchitektur abgeleitet. Zum anderen wird eine Architekturcharakterisierung anhand von Benchmark-

**Tabelle 4.1:** Designparameter des Systems

| | Parameter | Beschreibung | Veränderlichkeit |
|---|---|---|---|
| SDE | $H_{img}$ | Bildhöhe des Eingangsbildes | Synthese |
| | $W_{img}$ | Bildbreite des Eingangsbildes | Synthese |
| | $L_{Filter}$ | Filtergröße der Gauß-Filter | Synthese |
| | $L_{img,min}$ | Minimale Seitenlänge des kleinsten Pyramidenbildes, aus der sich die Anzahl an Oktaven $N_{oct}$ ergibt (vgl. Gl. 3.7) | Synthese |
| | TDM-Modus | Architekturvariante der Gauß-Filter | Synthese |
| | $f_{SDE}$ | Taktfrequenz SDE | Laufzeit |
| VLIW | Konfiguration | Prozessorkonfiguration ($b0$-$b4$, $c1$, $c2$, $b3x2$, $b4x2$, $c1x2$, $c2x2$, vgl. Tab. 3.4) | Synthese |
| | Pipeline | Pipeline-Konfiguration (vgl. Tab. 3.3) | Synthese |
| | $n_c$ | Anzahl an Merkmalskandidaten im Bild | Eingangsdaten |
| | $f_{VLIW}$ | Taktfrequenz VLIW-Prozessor | Laufzeit |

Bildsequenzen durchgeführt, aus der sich mithilfe von Regression die Modellparameter aus unterschiedlichen Messwerten (Performance-Counter, Profiling, Strommessung) ergeben.

Als Benchmark wird derselbe Datensatz verwendet, der bereits in Abschnitt 3.7.2 zur merkmalsbasierten Eigenbewegungsschätzung genutzt wurde. Dieser besteht aus 22 Stereo-Bildsequenzen, die während der Fahrt (Wohngebiet, Stadt, Autobahn) mit einem Auto aufgenommen wurden [Gei13]. Ein exemplarisches Bild jeder Sequenz ist im Anhang in Abschnitt A.5 dargestellt. Der Datensatz wird für die Modellierung in zwei Teile geteilt. Die ersten elf Sequenzen dienen der Architekturcharakterisierung und werden im Folgenden als *Trainingssequenzen* bezeichnet. Die Autobahn-Sequenz 01 wird hier aufgrund der Ergebnisse der Eigenbewegungsschätzung bei hohen Fahrgeschwindigkeiten nicht verwendet (siehe Abschnitt 3.7.2). Die Trainingssequenzen bestehen insgesamt aus 44.200 Einzelbildern. Die zweiten elf Sequenzen (ca. 40.700 Einzelbilder), im Folgenden *Testsequenzen* genannt, dienen der Modellverifikation.

## 4.2 Modellierung der Laufzeit

### 4.2.1 SDE-Modell

Aufgrund der Streaming-Verarbeitung der Pixel des Eingangsbildes (vgl. Abb. 3.9) ist die Laufzeit des SDE deterministisch. Die Ausführungszeit für ein Bild sowie die dazugehörige Bildrate ergibt sich aus

$$\begin{aligned} t_{SDE,total} &= F(L_{Filter},\, H_{img},\, W_{img},\, f_{SDE},\, TDM) = \frac{N_{SDE,total}}{f_{SDE}} \\ \text{und} \qquad R_{SDE} &= \frac{1}{t_{SDE,total}}, \end{aligned} \tag{4.1}$$

mit $N_{SDE,total}$ der Anzahl an Takten (Latenz), $R_{SDE}$ der Bildrate und $f_{SDE}$ der Taktfrequenz des SDE. Die Verarbeitungslatenz für ein Bild berechnet sich dabei nach folgender

Gleichung:

$$N_{SDE,total} = N_{SDE,work} + N_{SDE,idle/blocked}. \tag{4.2}$$

$N_{SDE,idle/blocked}$ ist dabei zunächst unbekannt und nur im Systemkontext in Kombination mit dem VLIW-Prozessor von Bedeutung. Im idealen Fall ist $N_{SDE,idle/blocked}$ zu vernachlässigen. Die Latenz der Berechnungen ergibt sich zu

$$\begin{aligned} N_{SDE,work} = N_{Latenz,initial} + \sum_{oct=0}^{N_{oct}-1} N_{Latenz,Zeilenspeicher}\left(\frac{H_{img}}{2^{oct}}, \frac{W_{img}}{2^{oct}}, L_{Filter}\right) \\ + N_{oct} \cdot \left(N_{Latenz,Filter}(L_{Filter}) + N_{Latenz,Pipeline}\right) \end{aligned} \tag{4.3}$$

Bis die ersten Pixel beim Start des SDE bereitstehen, entsteht eine initiale Latenz von $N_{Latenz,initial}$. Diese ist abhängig von dem Taktfrequenzverhältnis zwischen SDE und Bus, kann aber vernachlässigt werden. Der größte Anteil der Latenz ergibt sich aus dem Streaming der Pixel durch den Zeilenspeicher. Abhängig von der Anzahl an Zeilen, welche der Filtergröße entspricht, und der Bildauflösung wird für die verschiedenen Oktaven nach folgender Gleichung akkumuliert:

$$N_{Latenz,Zeilenspeicher}(h, w, l_{Filter}) = (h + l_{Filterradius}) \cdot (w + 2 \cdot l_{Filterradius}) \tag{4.4}$$

$$\text{mit} \qquad l_{Filterradius} = \left\lfloor \frac{l_{Filter}}{2} \right\rfloor \tag{4.5}$$

Die Anzahl der Oktaven $N_{oct}$ wird nach Gl. 3.7 aus der Bildauflösung berechnet. Zusätzlich muss pro Oktave noch die Latenz durch die Gauß-Pyramide und die nachgeschalteten Pipeline-Strukturen hinzuaddiert werden. Dabei ist $N_{Latenz,Filter}$ nach

$$N_{Latenz,Filter}(L_{Filter}) = \begin{cases} 2 \cdot L_{Filter} + 7 - L_{Filterradius} & \text{wenn TDM} \times 1 \\ \frac{3 \cdot L_{Filter}+7}{2} - L_{Filterradius} & \text{wenn TDM} \times 2 \\ \frac{4 \cdot L_{Filter}+7}{3} - L_{Filterradius} & \text{wenn TDM} \times 3 \\ \frac{7 \cdot L_{Filter}+7}{6} - L_{Filterradius} & \text{wenn TDM} \times 6 \end{cases} \tag{4.6}$$

abhängig von dem gewählten TDM-Modus. Höhere TDM-Modi benötigen mehr Speicherelemente (siehe Abschnitt 3.4.1 ab Seite 49) und haben eine längere Latenz (z. B. $3 \cdot L_{Filter} + 7$ bei TDM×2). Aufgrund der höheren Taktfrequenz, mit der die Daten diese Speicherelemente durchlaufen, reduziert sich die Latenz aber wieder aus Sicht der umliegenden, *langsameren* Gesamtschaltung (Division mit 2, 3 bzw. 6). $N_{Latenz,Pipeline}$ entspricht den nachfolgenden Pipeline-Stufen nach der Filterung und beträgt 16 Takte. Sowohl die Filterlatenz, als auch die Pipeline-Latenz können gegenüber der Latenz des Zeilenspeichers vernachlässigt werden. Zur Vollständigkeit addiert sich bei der Verarbeitung eines Bildes noch $N_{Latenz,idle/blocked}$, was derjenigen Anzahl an Takten entspricht, die der SDE entweder am Ende eines Bildes zusätzlich warten muss, bis der VLIW-Prozessor

**Tabelle 4.2:** Modellierte Laufzeit des SDE für verschiedene TDM-Modi und Bildauflösungen ($L_{Filter} = 41$).

| TDM-Modus | $f_{SDE,max}$ | Bildauflösung 640×480 px | | 1024×376 px[1] | | 1920×1080 px | |
|---|---|---|---|---|---|---|---|
| | (MHz) | Takte | ms (fps) | Takte | ms (fps) | Takte | ms (fps) |
| ×1 | 200 | 471.565 | 2,36 (424) | 581.525 | 2,91 (344) | 2.930.110 | 14,65 (68) |
| ×2 | 200 | 471.469 | 2,36 (424) | 581.429 | 2,91 (344) | 2.929.966 | 14,65 (68) |
| ×3 | 133 | 471.437 | 3,54 (282) | 581.397 | 4,36 (229) | 2.929.918 | 21,97 (46) |
| ×6 | 66 | 471.405 | 7,07 (144) | 581.365 | 8,72 (115) | 2.929.870 | 43,95 (23) |

[1] Bildauflösung derjenigen Benchmark-Sequenzen, die in dieser Arbeit zur Evaluation verwendet werden

alle Merkmalskandidaten im Merkmals-FIFO abgearbeitet hat, bzw. warten muss, weil der Merkmals-FIFO voll ist.

Tab. 4.2 zeigt exemplarisch die Gesamttakte mit $N_{Latenz,idle} = 0$ für die Verarbeitung eines Bildes bei verschiedenen TDM-Modi und Bildauflösungen. Bei Erhöhung des TDM-Faktors sinkt die Anzahl der Takte durch die sinkende Filterlatenz. Im Vergleich zur Gesamtzahl an Takten ist dies jedoch vernachlässigbar. Einen größeren Einfluss auf den Gesamtdurchsatz hat die durch TDM sinkende maximal mögliche Taktfrequenz des SDE, sodass der Durchsatz bei höheren TDM-Faktoren ebenfalls sinkt (vgl. Seite 51).

### 4.2.2 VLIW-Modell

Die Verarbeitungsdauer der Merkmalskandidaten auf dem VLIW-Prozessor ist datenabhängig. Die Anzahl an Takten für die Subpixel-Lokalisierung hängt von der Merkmalsumgebung ab. Zudem muss ein Deskriptor nur dann berechnet werden, wenn die Merkmalskandidaten den Schritt der Stabilitätsprüfung bestehen. Daraus ergibt sich der in Abb. A.4 im Anhang dargestellte Kontrollflussgraph, welcher von verschiedenen Merkmalskandidaten unterschiedlich durchlaufen wird.

Um die Laufzeit des VLIW-Prozessors zu modellieren, wird zunächst die Ausführungsstatistik des Kontrollflussgraphen erfasst. Dazu werden die Trainingssequenzen des Benchmark-Datensatzes für drei unterschiedliche Konfigurationen des VLIW-Prozessors durchlaufen, die Merkmalskandidaten extrahiert und ein Profiling durchgeführt. Diese Konfigurationen sind die Prozessorbasiskonfiguration $\#b0$ und die Konfigurationen mit maximaler Spezialisierung durch Modulverdoppelung bzw. dedizierte Erweiterungen ohne ($\#c2$) und mit ($\#c2x2$) X2-Modus (vgl. Tab. 3.4). Die Pipeline hat für alle Konfigurationen eine Basisstruktur. Das Instruction Scheduling wurde mit dem Populationsparameter $o = 14$ durchgeführt (vgl. [Gie17]). X2-Instruktionen wurden von Hand eingefügt.

Die Ergebnisse des Profilings sind in Abb. 4.1 in Histogrammform dargestellt. Die Abbildung zeigt die Häufigkeitsverteilungen beim Durchlaufen der Trainingssequenzen über den benötigten *Takten pro Merkmalskandidat*. Die zwei Bins mit der jeweils größten Anzahl an Takten fallen auf Merkmalskandidaten, für die ein Deskriptor berechnet

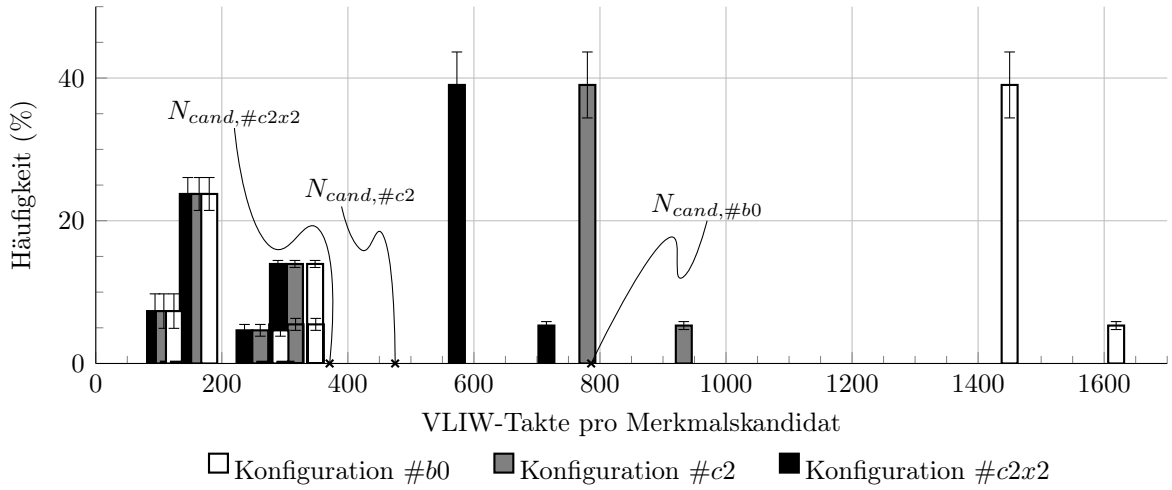



**Abbildung 4.1:** Ausführungsstatistik des VLIW beim Durchlaufen der Trainingssequenzen für drei unterschiedliche Prozessorkonfigurationen in Basispipeline. Zur Erläuterung der Konfigurationen siehe Tab. 3.4. Die Diskretisierung der Werte ergibt sich aus der Programmstruktur (vgl. Abb. A.4 im Anhang).

und bei denen zur Subpixel-Lokalisierung eine bzw. zwei Iterationen durchlaufen werden. Die Bins mit weniger Takten repräsentieren Merkmalskandidaten, die während der Subpixel-Lokalisierung bzw. während der Stabilitätsprüfung verworfen werden. Die Verteilung ist für die verschiedenen Trainingssequenzen etwas unterschiedlich. Daher wurden die Verteilungen aller Sequenzen gemittelt und die Standardabweichung für jeden Bin bestimmt. Der Erwartungswert der jeweiligen Verteilung, d. h. die mittlere Anzahl an VLIW-Takten pro Merkmalskandidat, wird mit $N_{cand}$ bezeichnet und stellt den wesentlichen Modellparameter der VLIW-Modellierung dar.

Der Modellierungsansatz für die Ausführungszeit auf dem VLIW geht von der Grundannahme aus, dass die Ausführungsstatistik des VLIW für jede Art von Eingangsdaten konstant ist. Die Histogramme in Abb. 4.1 zeigen nur geringe Standardabweichungen in den verschiedenen Bins für die Trainingssequenzen. Zur Bewertung des Modellfehlers wurde dieselbe Untersuchung auch auf den Testsequenzen des Datensatzes durchgeführt und jeweils $N_{cand}$ für verschiedene Prozessorkonfigurationen in Basispipeline ermittelt (siehe Tab. 4.3). Der Modellfehler, d. h. die Abweichung von $N_{cand}$ zwischen den Trainings- und den Testsequenzen, liegt zwischen 3,8% und 5,1%. Der mittlere Fehler beträgt 4,5%. Abweichungen in der Ausführungsstatistik entstehen durch verschobene Häufigkeiten insbesondere der beiden häufigsten Bins gegeneinander, sodass sich auch der Erwartungswert verschiebt. Die Annahme, dass der Wert für $N_{cand}$ für beliebige andere Bildsequenzen generalisiert, wird allerdings weitestgehend bestätigt, sodass zur Modellierung der Latenz pro Merkmalskandidat $N_{cand}$ gewählt wird. Durch unterschiedliche Prozessorkonfigurationen liegt dieser Wert hier zwischen 371 und 786 Takten. Durch Veränderung der Pipeline des Prozessors und ein daraus resultierendes verändertes Instruction Scheduling ändert

**Tabelle 4.3:** Benötigte Takte zur Verarbeitung eines Merkmalskandidaten für verschiedene VLIW-Prozessorkonfigurationen in Basispipeline. Zur Erläuterung der Konfigurationen siehe Tab. 3.4. Es ergibt sich ein mittlerer Fehler von 4,5%.

| Architektur | $N_{cand}$ Trainingssequenzen (Takte) | $N_{cand}$ Testsequenzen (Takte) | Modellfehler (%) |
|---|---|---|---|
| $\#b0$ | 786 | 749 | 4,7 |
| $\#b1$ | 667 | 638 | 4,3 |
| $\#b2$ | 771 | 734 | 4,8 |
| $\#b3$ | 652 | 622 | 4,6 |
| $\#b4$ | 652 | 622 | 4,6 |
| $\#c1$ | 475 | 457 | 3,8 |
| $\#c2$ | 475 | 457 | 3,8 |
| $\#b3x2$ | 545 | 518 | 5,0 |
| $\#b4x2$ | 532 | 505 | 5,1 |
| $\#c1x2$ | 383 | 367 | 4,2 |
| $\#c2x2$ | 371 | 354 | 4,6 |

sich $N_{cand}$, was durch ein erneutes Benchmarking ermittelt werden kann.

Die Verarbeitung im VLIW erfolgt zyklisch. Der VLIW greift auf den Merkmals-FIFO im SDE zu (*Polling*) und verarbeitet einen Merkmalskandidaten, falls der FIFO nicht leer ist. Dieses Vorgehen wiederholt sich, bis der SDE das Bildende signalisiert hat und der Inhalt des FIFOs abgearbeitet ist. Findet der VLIW während der Berechnung einen leeren FIFO vor, so erfolgt das Polling in einer Schleife erneut. Die Gesamtlatenz für ein komplettes Bild kann durch folgende Gleichung formuliert werden:

$$N_{VLIW,total} = N_{VLIW,work} + N_{VLIW,poll} + N_{VLIW,control} \tag{4.7}$$

$$\text{mit} \quad N_{VLIW,work} = n_c \cdot N_{cand} \tag{4.8}$$

$$\text{und} \quad N_{VLIW,poll} = (n_c + x) \cdot N_{poll} + N_{pollToFirst} \tag{4.9}$$

$$\text{und} \quad N_{pollToFirst} = L_{Filter} \cdot (W_{img} + 2 \cdot L_{Filterradius}) \tag{4.10}$$

mit $n_c$ der Anzahl an Merkmalskandidaten im Bild und $x$ derjenigen Anzahl an Polling-Aufrufen, in denen der VLIW einen leeren Merkmals-FIFO vorfindet. $x$ ist dabei zunächst vernachlässigbar und bekommt nur im Systemkontext in Kombination mit dem SDE eine Bedeutung. $N_{poll}$ ist abhängig von der Prozessorkonfiguration und liegt zwischen 10 und 30 Takten. $N_{control}$ repräsentiert den Kontrolloverhead am Anfang und am Ende eines Bildes und kann gegenüber den übrigen Anteilen vernachlässigt werden. $N_{pollToFirst}$ ergibt sich aus der initialen Wartezeit am Bildanfang, bevor der erste Merkmalskandidat detektiert wird und somit der Merkmals-FIFO noch leer ist.

Die Ausführungszeit des VLIW für ein einzelnes Bild ergibt sich anhand der Taktfrequenz $f_{VLIW}$ zu

$$t_{VLIW,total} = \frac{N_{VLIW,total}}{f_{VLIW}}. \tag{4.11}$$

#### Offline-Schätzung der Ausführungsstatistik

Die Konfiguration des VLIW-Prozessors ist sehr flexibel. Neben der Erhöhung der parallelen Rechenressourcen und Spezialisierung für eine Anwendung (*VLIW-Prozessorkonfiguration*) kann zur Verringerung des kritischen Pfades die *Pipeline* verändert werden. Der Aufwand, um ein Laufzeit-Modell für eine Vielzahl von Konfigurationen zu bestimmen, ist hoch, da zur Gewinnung der Profiling-Informationen bzw. der Ausführungsstatistik der Trainingsdatensatz durchlaufen werden muss.

Jede Konfiguration erzeugt neue Randbedingungen für den Instruction Scheduler, sodass der ausgeführte Code zwar funktional gleich, allerdings in der Anordnung und Anzahl der Instruktionen unterschiedlich ist. Jedoch unterscheidet sich der Code nach Scheduling für verschiedene Pipeline-Konfigurationen lediglich in der Länge der SLMs. Die Anzahl der SLMs und die Häufigkeit der SLM-Aufrufe ist bei gleichen Eingangsdaten deterministisch. Daher kann ein Profiling basierend auf einer Referenzmessung für neue Pipeline-Konfigurationen ohne weitere Benchmark-Durchläufe *offline geschätzt* werden. Dazu muss lediglich der geschedulte Code für die neuen Konfigurationen vorliegen. Abb. 4.2 zeigt schematisch links den Ablauf zur Messung der Ausführungsstatistik für eine Referenzkonfiguration. Rechts ist der Ablauf für die Schätzung dargestellt: das gemessene Profiling der Referenzkonfiguration, d. h. die Ausführungshäufigkeiten der Instruktionen in den einzelnen SLMs, kann vom Profiling-Schätzer direkt auf einen weiteren geschedulten Code, in dem die SLMs dieselben sind und nur unterschiedliche Längen haben, übertragen werden. Dazu werden die Ausführungshäufigkeiten der Referenz-SLMs für die Ziel-SLMs direkt übernommen.

Die Schätzung des Profilings ist in den hier betrachteten Fällen immer exakt. DMA-Transfers werden im Hintergrund ausgeführt und die hier vorliegenden Veränderungen in der Länge der SLMs haben keine Änderung des Kontrollflusses bzw. ein Warten auf DMA-Transfers zur Folge.

### 4.2.3 System-Modell

Die beiden vorangegangenen Abschnitte haben SDE und VLIW-Prozessor separat betrachtet. Zur Modellierung des Gesamtsystems werden nun beide Komponenten miteinander kombiniert und die Wechselwirkungen untersucht.

#### Warteschlange und Ausnutzungsgrad

Die Verbindung zwischen SDE und VLIW-Prozessor bildet der Merkmals-FIFO, wodurch das Gesamtsystem als *Warteschlangensystem* (z. B. [Zim08]) beschrieben werden kann.

Der SDE bildet die Inputquelle in das System, und die Merkmalskandidaten entsprechen den eintreffenden Elementen. Die Zwischenankunftszeit $\tau_a$, d. h. die Zeit zwischen zwei Elementen, entspricht der Rechenzeit von einem Merkmalskandidaten bis zum nächsten. Die Merkmalskandidaten sind im Bild niemals gleichverteilt (vgl. Abb. A.1 im Anhang), da die Merkmalspositionen vom Inhalt des Eingangsbildes abhängen, sodass

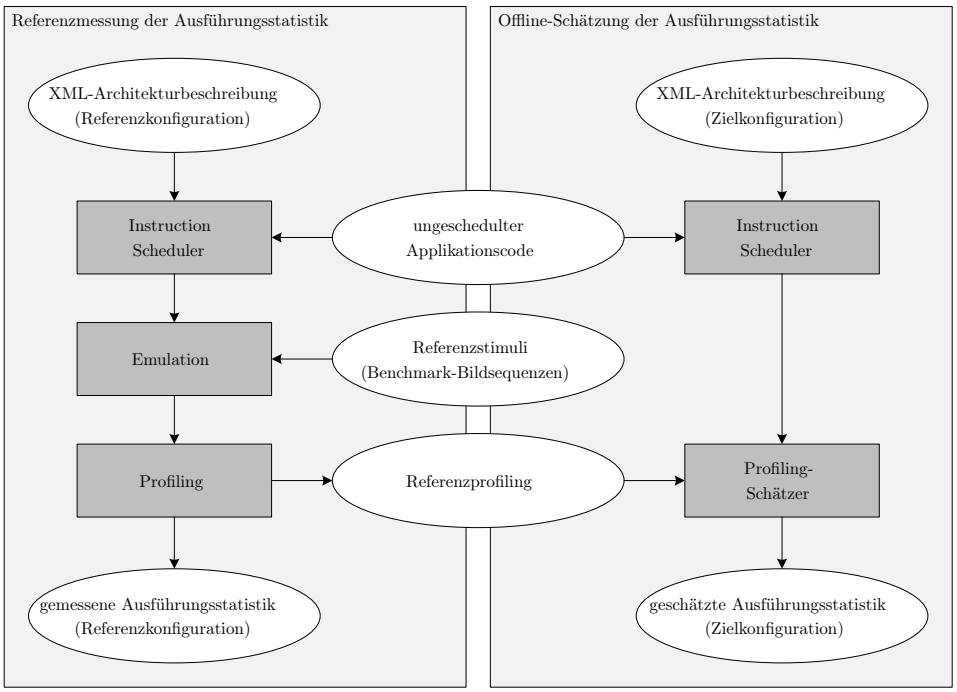



**Abbildung 4.2:** Ablauf zur Offline-Schätzung der Ausführungsstatistik für eine Zielkonfiguration des VLIW basierend auf einer Referenzmessung

bei der vorliegenden zeilenweisen Streaming-Verarbeitung der Pixelabstand zwischen zwei Merkmalen nicht deterministisch, sondern stochastisch ist. Untersuchungen haben gezeigt, dass der Pixelabstand grob durch eine Exponentialverteilung beschrieben werden kann. Lange Pixelabstände zwischen zwei Merkmalen treten weniger häufig auf als kurze. Allerdings sorgen kontrastarme Flächen (Straße, Himmel) mit wenig Merkmalen und gerade Linien (Fahrbahnrandbegrenzung) mit voneinander abhängigen Merkmalen für stellenweise deutliche, ausreißerartige Abweichungen von der Verteilung.

Zur Berechnung einer Merkmals-Ankunftsrate $\lambda$ wird vereinfacht die Anzahl an Merkmalskandidaten $n_c$ über die Verarbeitungszeit eines kompletten Bildes (mehrere Oktaven) gemittelt:

$$\lambda = \frac{1}{\tau_a} = \frac{n_c}{t_{SDE,work}} = \frac{n_c \cdot f_{SDE}}{N_{SDE,work}} \qquad \text{[Kandidaten pro Sekunde]} \tag{4.12}$$

Die Warteschlange (FIFO) hat ein endliches Fassungsvermögen und arbeitet nach dem first-come-first-serve Prinzip. Dabei gibt es genau eine Servicestation (VLIW-Prozessor), deren Bedienprozess durch eine konstante Servicerate $\mu$ beschrieben werden kann:

$$\mu = \frac{1}{\tau_s} = \frac{1}{t_{VLIW,workPerCand}} = \frac{f_{VLIW}}{N_{cand}} \qquad \text{[Kandidaten pro Sekunde]} \tag{4.13}$$

$N_{cand}$ entspricht, wie im vorangegangenen Abschnitt erläutert, dem Erwartungswert einer anwendungsabhängigen Verteilung.

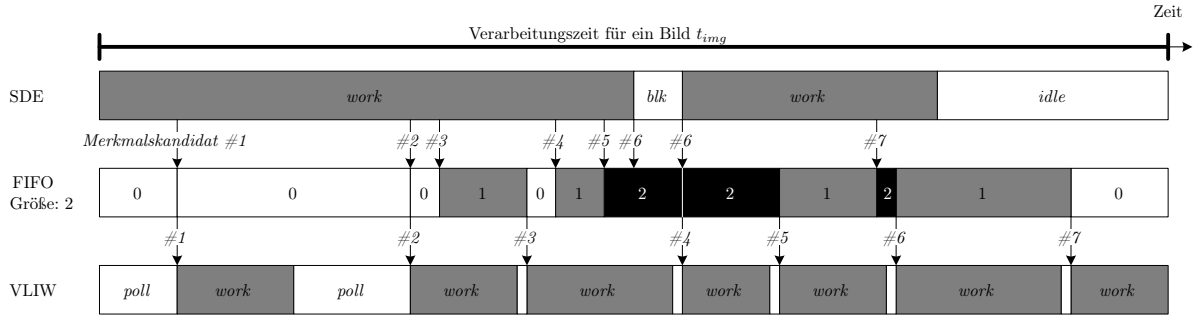



**Abbildung 4.3:** Qualitative Verdeutlichung des Systemverhaltens über der Zeit bei sieben Merkmalskandidaten und einer FIFO-Tiefe von zwei. Wartezeiten in weiß.

Mithilfe der beiden Raten $\lambda$ und $\mu$ lässt sich ein FIFO-*Ausnutzungsgrad* (in der Literatur auch Verkehrsintensität genannt [Zim08]) bestimmen:

$$\rho = \frac{\lambda}{\mu} = n_c \cdot \frac{N_{cand}}{N_{SDE,work}} \cdot \frac{f_{SDE}}{f_{VLIW}} \qquad \text{[einheitenlos]} \tag{4.14}$$

Der Fall $\rho > 1$ entspricht einer größeren Ankunfts- als Servicerate. Dies würde zu einer unendlich wachsenden Zahl von Elementen im System bzw. einer unendlich langen Warteschlange führen, da im speziellen Fall der VLIW-Prozessor die vom SDE generierten Merkmalskandidaten nicht mehr abarbeiten kann. Das System ist so konzipiert, dass im Fall eines komplett gefüllten FIFOs der SDE die Verarbeitung pausiert, sodass keine weiteren Merkmalskandidaten neu generiert werden, bis der FIFO wieder Kapazitäten aufweist. Da die Anzahl an Merkmalskandidaten in einem Bild zudem endlich ist und der SDE immer nur bildweise und nicht mit einem endlosen Datenstrom versorgt wird, muss das System hier nicht zwangsläufig in einem Betriebszustand mit $\rho < 1$ arbeiten.

### Laufzeitenmessung

Abb. 4.3 zeigt zur Verdeutlichung einmal qualitativ das Systemverhalten während der Verarbeitung eines Bildes mit sieben Merkmalskandidaten und einer FIFO-Tiefe von zwei. Die Kandidaten sind im Bild nicht gleichverteilt. Es ergeben sich zeitlich unterschiedlich lange Abschnitte, die jeweils einem Verarbeitungszustand zugeordnet werden können. Im Fall eines leeren bzw. vollen FIFOs entstehen Wartezeiten im VLIW-Prozessor bzw. SDE. Die Ausführungszeit im VLIW-Prozessor pro Kandidat variiert, wie in Abschnitt 4.2.2 erläutert. Im Zusammenspiel mit dem Host-PC erfolgt die Kommunikation durch Semaphor-Signale (vgl. Abb. A.5 im Anhang).

Um die Länge der einzelnen Abschnitte zu messen, sind an verschiedenen Stellen Performance-Counter implementiert, die vom VLIW-Prozessor kontrolliert werden. Insgesamt existieren Taktzyklenzähler für die SDE-Zustände *work*, *blk* (Wartezeit innerhalb eines Bildes) und *idle* (Wartezeit am Ende eines Bildes), für die Verarbeitungsdauer eines

kompletten Bildes, eventuelle MMCM-Rekonfigurationszeiten und den VLIW-Zustand *poll*. Die Zähler liegen in der Bus-Taktdomäne und können dort direkt vom Host-PC gelesen werden. Die MMCM-Rekonfigurationszeiten sind im Vergleich zu den anderen Zeiten um Größenordnungen kleiner, sodass sie in der Modellierung vernachlässigt werden. Zur Messung der Laufzeiten werden Bild-Sequenzen in Emulation auf der Zielplattform (siehe Abschnitt 3.3) durchlaufen und die Performance-Counter bildweise ausgelesen. Im Allgemeinen gilt

$$R_{img} = \frac{1}{t_{img}} \tag{4.15}$$

mit $R_{img}$ der Bildrate und $t_{img}$ der Gesamtlaufzeit für die Merkmalsextraktion eines Bildes.

### Modellgleichungen

Im Folgenden werden die Modellgleichungen für das Gesamtsystem erarbeitet und exemplarisch für eine ausgesuchte VLIW-Architektur belegt. Als VLIW-Konfiguration wird $\#c2x2$ mit einer erweiterten Pipeline gewählt, sodass der Prozessor mit einer maximalen Taktfrequenz von 100 MHz bei $N_{cand} = 563$ betrieben werden kann. Als Messwerte für die Modellbewertung werden die Testsequenz 19 mit durchschnittlich 1.052 Merkmalskandidaten ($n_c$) und 9.962 Bildern gewählt, für verschiedene Taktfrequenzen von VLIW und SDE durchlaufen und es werden die gemessenen Laufzeiten der unterschiedlichen Zähler gemittelt.

Abb. 4.4 zeigt die Messwerte durch verschiedene Symbole und die Modellierung durch Linien und verdeutlicht die einzelnen Anteile zur Laufzeit, die für die Herleitung des Modells relevant sind. Die Abbildung stellt die Ausführungszeiten beim Durchlaufen von Testsequenz 19 über $f_{SDE}$ bei $f_{VLIW} = 100$ MHz dar. Durch die Erhöhung von $f_{SDE}$ bei konstantem $f_{VLIW}$ und $n_c$ steigt der Ausnutzungsgrad $\rho$, welcher über eine zweite Achse ebenfalls dargestellt ist.

Die Laufzeit $t_{img,model}$ für die Verarbeitung folgt folgender Gleichung:

$$\begin{aligned} t_{img,model} &= t_{SDE,total} \stackrel{!}{=} t_{VLIW,total} \\ &= t_{SDE,work} + t_{SDE,idle/blocked} \stackrel{!}{=} t_{VLIW,work} + t_{VLIW,poll} \end{aligned} \tag{4.16}$$

Dem liegt die Annahme zugrunde, dass durch die Kopplung durch den Merkmals-FIFO die zeitlichen Anteile des SDE gleich den Anteilen des VLIW sein müssen. Dabei sind die Anteile $t_{SDE,idle/blocked}$ und $t_{VLIW,poll}$ zunächst nicht genau bekannt.

Unter Zuhilfenahme von Gl. 4.8 und Gl. 4.9 mit $x = 0$ lässt sich für den VLIW eine minimale Laufzeit $t_{VLIW,total,min}$ nach folgender Gleichung bestimmen:

$$\begin{aligned} t_{VLIW,total,min} &= t_{VLIW,work} + t_{VLIW,poll,x=0} \\ &= \frac{n_c \cdot N_{cand}}{f_{VLIW}} + \frac{n_c \cdot N_{poll} + N_{pollToFirst}}{f_{VLIW}} \end{aligned} \tag{4.17}$$

Aufgrund der Annahme $x = 0$ ist dies als Laufzeit des VLIW bzw. für das gesamte Bild nur in dem Fall korrekt, in dem der VLIW mit Ausnahme des Bildanfangs niemals einen leeren Merkmals-FIFO vorfindet, wenn also $t_{VLIW,total,min} > t_{SDE,work}$. Der VLIW ist im Vergleich zum SDE langsam und benötigt länger für die Weiterverarbeitung der Kandidaten als der SDE für deren Generierung. In diesem Fall entspricht die sich ergebende Zeitdifferenz derjenigen Zeit, in der sich der SDE im *blk* bzw. *idle* Zustand befindet:

$$t_{SDE,idle/blocked} = \begin{cases} t_{VLIW,total,min} - t_{SDE,work} & \text{wenn } t_{SDE,work} < t_{VLIW,total,min} \\ 0 & \text{sonst} \end{cases} \tag{4.18}$$

Liegt der umgekehrte Fall vor, d. h. $t_{SDE,work} > t_{VLIW,total,min}$, so ist der SDE im Vergleich zum VLIW langsam ($x > 0$). Daraus ergibt sich die tatsächliche Zeit, in der sich der VLIW im *poll* Zustand befindet:

$$t_{VLIW,poll} = \begin{cases} t_{SDE,work} - t_{VLIW,work} & \text{wenn } t_{SDE,work} > t_{VLIW,total,min} \\ t_{VLIW,poll,x=0} & \text{sonst} \end{cases} \tag{4.19}$$

Die Gesamtlaufzeit für die Verarbeitung ergibt sich somit unter Verwendung von Gl. 4.19 zu

$$\begin{aligned} t_{img,model} &= t_{VLIW,work} + t_{VLIW,poll} \\ &= \begin{cases} t_{SDE,work} & \text{wenn } t_{SDE,work} > t_{VLIW,total,min} \\ t_{VLIW,total,min} & \text{sonst} \end{cases} \end{aligned} \tag{4.20}$$

Abb. 4.4 zeigt die modellierten Teil-Ausführungszeiten mit durchgezogenen Linien. Die gestrichelte Linie entspricht einer Modellkorrektur nach Gl. 4.24, auf die weiter unten eingegangen wird. Die Grenze der Fallunterscheidung $t_{SDE,work} = t_{VLIW,total,min}$ ist durch eine gepunktete Linie dargestellt. Die modellierten Zeiten bilden den gemessenen Verlauf insgesamt gut ab, weichen jedoch im Bereich um $\rho = 1$ davon ab. Die Modellierung basiert auf Mittelwerten sowohl der Merkmalsanzahl über die Sequenz hinweg, als auch der Merkmalsverteilung im Bild und der Ausführungsstatistik des VLIW. Gerade im Bereich annähernd gleicher Ankunfts- und Serviceraten sind die Wechselwirkungen zwischen den beiden Komponenten am größten, sodass hier ein Modellfehler entsteht.

Die Wartezeiten im System können als

$$t_{waiting} = t_{SDE,idle/blocked} + t_{VLIW,poll} \tag{4.21}$$

zusammengefasst werden und sind in Abb. 4.5 für verschiedene Taktfrequenzen des VLIW dargestellt. Es ergeben sich klare Minima dieser Wartezeiten, in denen das System in *Balance* ist, d. h. VLIW und SDE am besten aufeinander abgestimmt arbeiten. Diese Minima befinden sich in etwa im Bereich von $\rho = 1$ für alle Taktfrequenzkombinationen. Trotz des vorliegenden Modellfehlers kann das Modell die Minima gut abbilden. Der Ausnutzungsgrad $\rho$ eignet sich als Beschreibung für den Betriebspunkt des Systems.

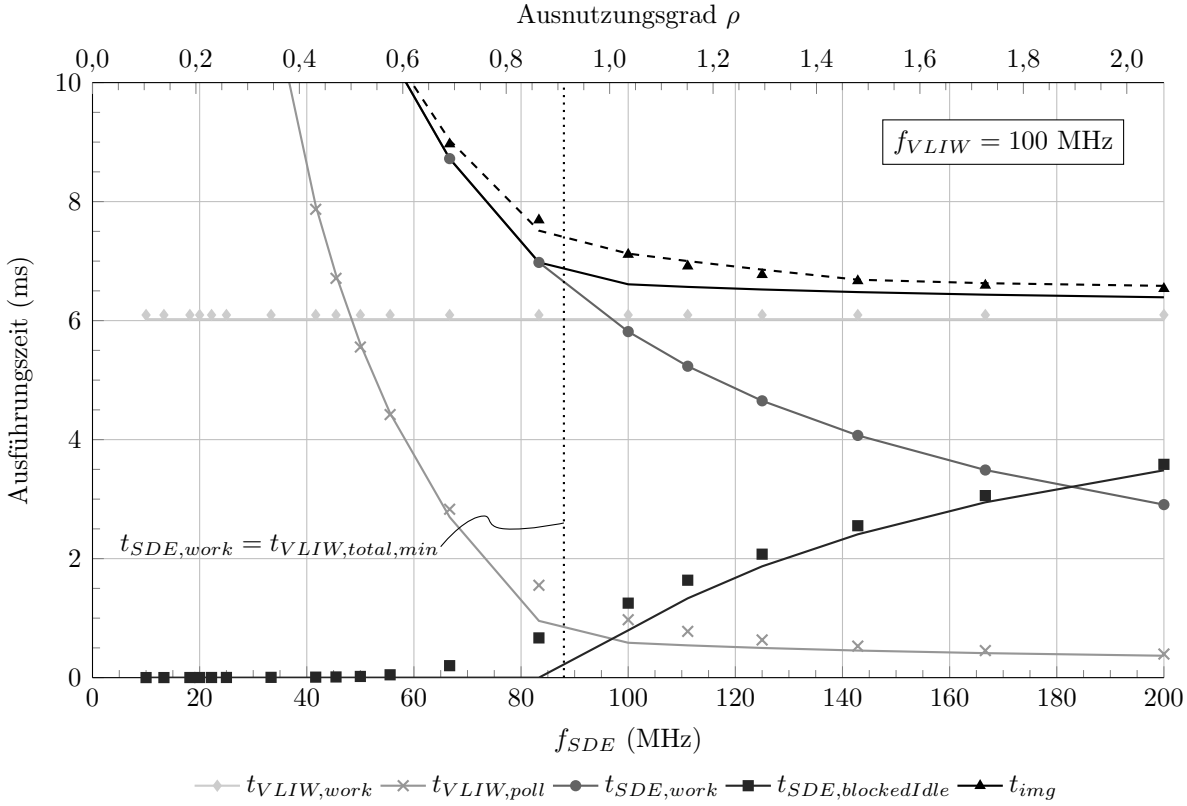



**Abbildung 4.4:** Unterschiedliche Anteile der Laufzeit. Die Symbole entsprechen gemessenen Werten (Testsequenz 19), die durchgezogenen Linien der Modellierung und die gestrichelte Linie der Modellkorrektur nach Gl. 4.24.

### Modellfehler und Korrektur

Im Folgenden soll der Modellfehler genauer betrachtet werden. Dieser wird hier als relativer Bildratenfehler $\Delta R$ definiert:

$$\Delta R = \left(1 - \frac{R_{img,mess}}{R_{img,model}}\right) \cdot 100 \qquad [\%] \tag{4.22}$$

Ein positiver Fehler bedeutet, dass das Modell die Messung überschätzt.

Der Modellfehler ist in Abb. 4.6a (links) für unterschiedliche Sequenzen über dem Ausnutzungsgrad bei $f_{VLIW} = 100$ MHz dargestellt (durchgezogene Linien). Dabei ist die direkte Zuordnung der einzelnen Verläufe zu den jeweiligen Sequenzen nicht relevant, sondern die allgemeine Tendenz. Für $\rho < 0,6$ ist der Fehler nahe 0, steigt danach für $\rho \approx 0,9$ auf ein jeweiliges Maximum und pendelt sich bei einem jeweils unterschiedlichen konstanten Wert ein.

Der unterschiedliche Verlauf der Modellfehler für verschiedene Sequenzen entsteht durch die unterschiedliche Ausführungscharakteristik im VLIW für verschiedene Sequenzen.

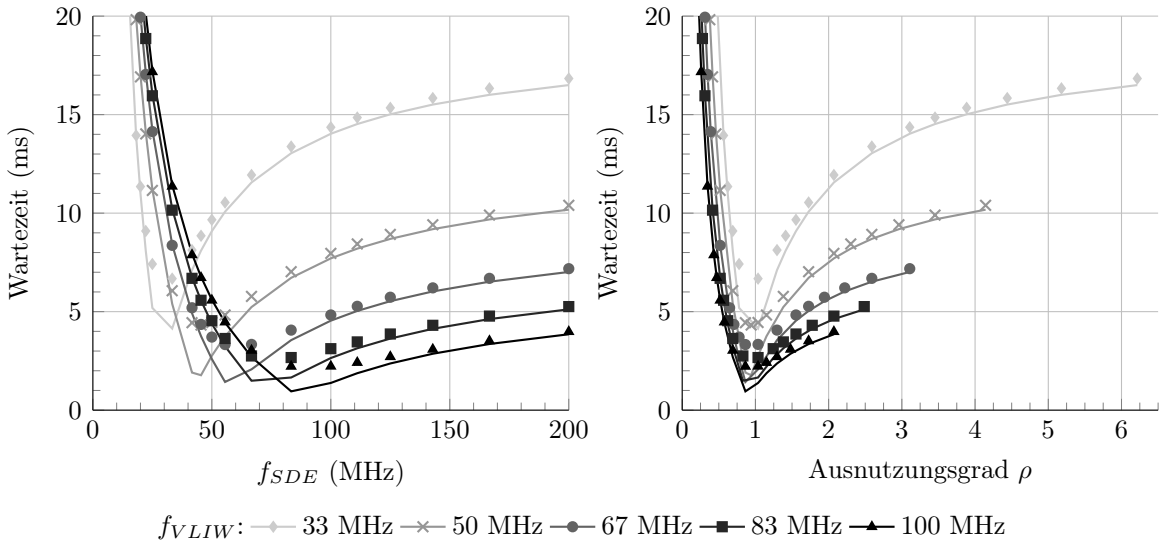



**Abbildung 4.5:** Wartezeiten im System für verschiedene Taktfrequenzen beim Durchlaufen von Testsequenz 19. Die Symbole entsprechen gemessenen Werte, die durchgezogenen Linien sind durch das Modell generiert.

$N_{cand}$ variiert leicht, wodurch das Modell von den Messwerten abweichende Zeiten liefert. Ändert sich das Verhältnis zwischen stabilen und verworfenen Merkmalen, so verschiebt sich die Verteilung nach Abb. 4.1. Dieses Verhältnis ist für unbekannte Sequenzen ebenfalls unbekannt und auch nicht konstant. Empirisch kann eine a-posteriori $N_{cand}$-Korrektur gefunden werden:

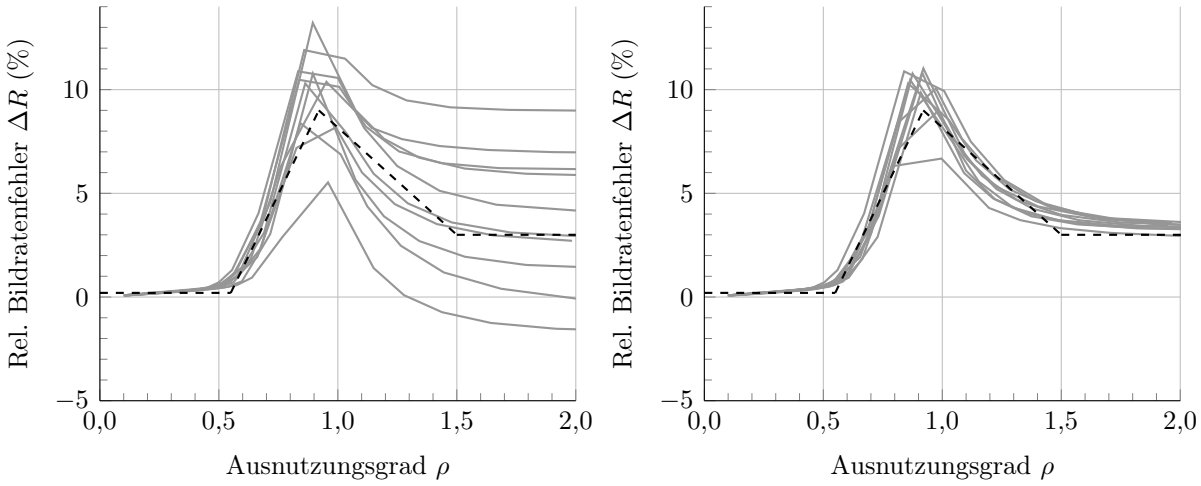
$$N_{cand,korr} = \left( \frac{\#\text{Stabile Merkmale}}{\#\text{Merkmalskandidaten}} - 0,45 + 1 \right) \cdot N_{cand} \tag{4.23}$$

Durch die $N_{cand}$-Korrektur ergibt sich Abb. 4.6a (rechts). Für alle Sequenzen ergibt sich nun ein etwa gleicher Verlauf des Modellfehlers. Eine solche Korrektur entspricht einer Anpassung des Modells an bestehende Sequenzen. Damit das Modell bei unbekannten Sequenzen immer noch gültig ist, wird die $N_{cand}$-Korrektur im Folgenden nicht verwendet und dient lediglich dem Vergleich.

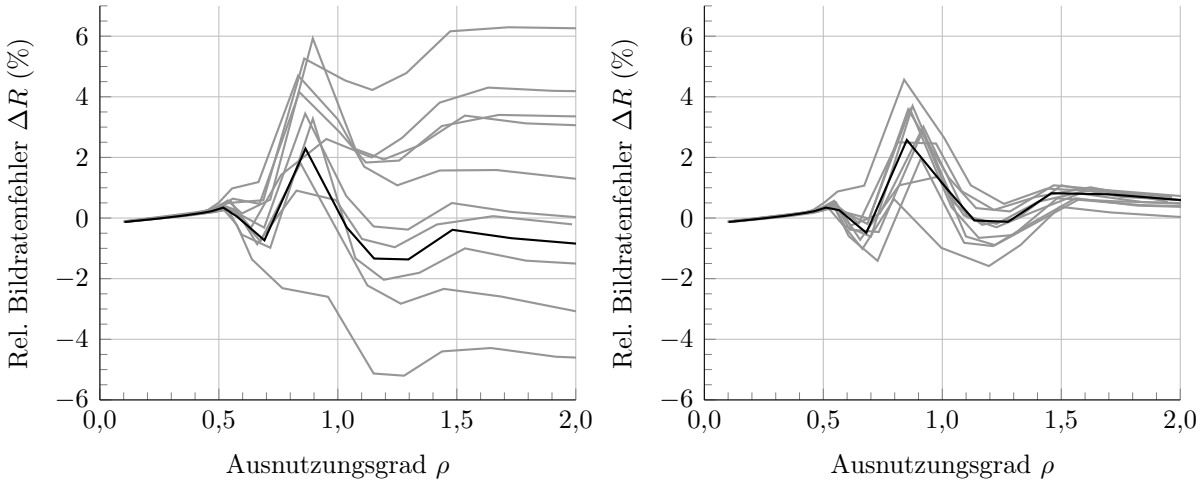
Aufgrund der systematisch aussehenden Fehlercharakteristik wurde eine empirische Korrekturfunktion $\kappa(\rho)$ definiert

$$t_{img,model,korr} = t_{img,model} \cdot \left( 1 + \frac{\kappa(\rho)}{100} \right) \tag{4.24}$$

$$\text{mit} \qquad \kappa(\rho) = \begin{cases} 0,2 & \text{für } \rho \leq 0,55 \\ \frac{9-0,2}{0,92-0,55} \cdot (\rho - 0,92) + 9 & \text{für } 0,55 < \rho < 0,92 \\ -\frac{9-3}{1,5-0,92} \cdot (\rho - 0,92) + 9 & \text{für } 0,92 \leq \rho < 1,5 \\ 3 & \text{für } \rho \geq 1,5 \end{cases} \tag{4.25}$$



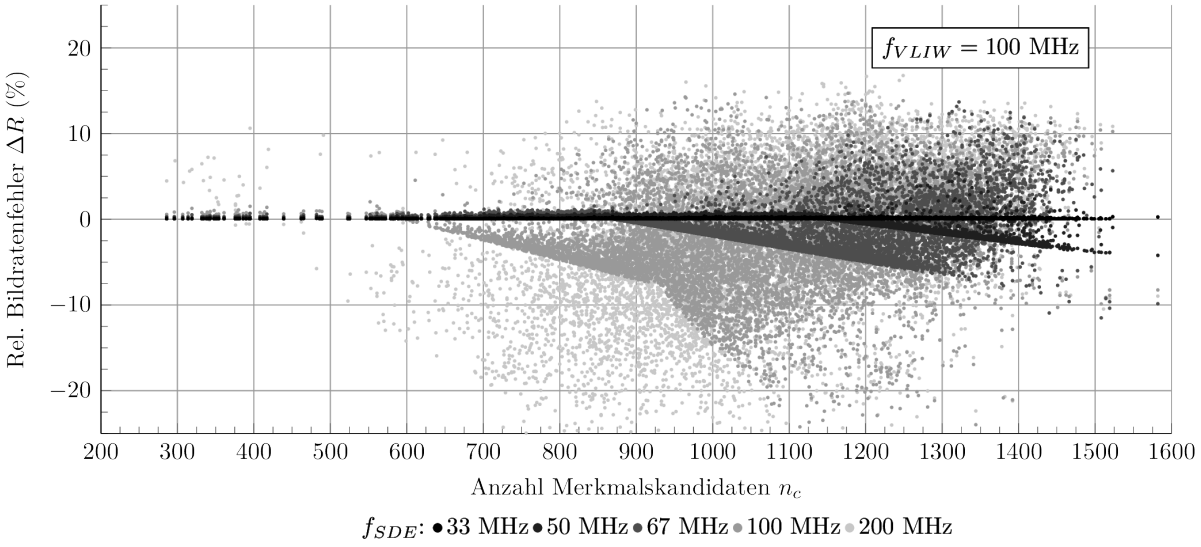

**(a)** Ohne Korrektur der Laufzeit



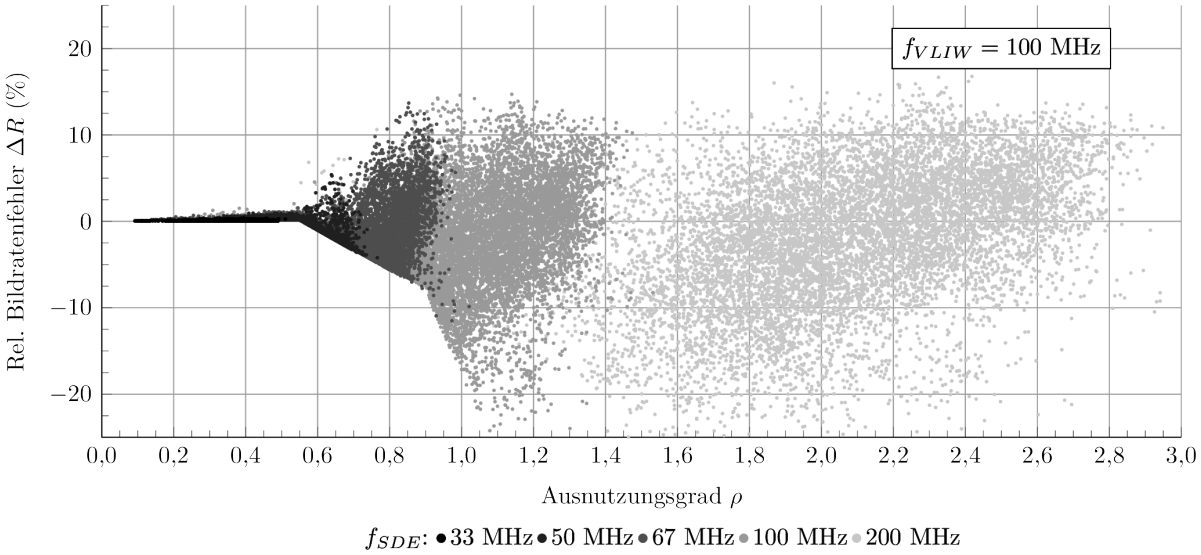

**(b)** Mit Korrektur der Laufzeit nach Gl. 4.24

**Abbildung 4.6:** Modellfehler links vor und rechts nach der a-posteriori $N_{cand}$-Korrektur (Gl. 4.23). Die verschiedenen Kurven entsprechen den unterschiedlichen *train*-Sequenzen.

die die modellierte Laufzeit korrigieren und damit den Modellfehler ausgleichen soll. Die Korrekturfunktion ist in Abb. 4.6a (links) durch eine gestrichelte Linie dargestellt. Wenn die Korrekturfunktion nach Gl. 4.24 angewendet wird, so ergeben sich für alle Sequenzen reduzierte Modellfehler (siehe Abb. 4.6b (links)). Der korrigierende Einfluss auf die Testsequenz 19 wird zusätzlich in Abb. 4.4 (gestrichelte Linie) deutlich. Der maximale Modellfehler für Testsequenz 19 beträgt vor der Korrektur 9,3% und liegt nach der Korrektur bei 2,3%.




**(a)** Modellfehler über $n_c$




**(b)** Modellfehler über $\rho$

**Abbildung 4.7:** Modellfehler der Testsequenz 19 bei verschiedenen Taktfrequenzen. Jeder Punkt entspricht einem Einzelbild.

Die Laufzeiten, die den bisherigen Abbildungen zugrunde liegen, ergeben sich aus dem Mittelwert der Laufzeiten über jedes Einzelbild einer kompletten Sequenz. Die Modellierung erfolgt dabei auf Basis der mittleren Anzahl an Merkmalen pro Bild $n_c$. Da sich aber die Bilddaten über die Sequenz hinweg ändern, ist auch $n_c$ in jedem Einzelbild unterschiedlich. Um den Modellfehler für verschiedene Werte von $n_c$ und Taktfrequenzkombinationen zu beurteilen, zeigt Abb. 4.7a den Fehler für die modellierte Laufzeit jedes Einzelbildes der Testsequenz 19. Der maximale absolute Fehler liegt mit bis

zu 20% deutlich höher als der zuvor ermittelte mittlere Fehler mit einer angenommenen mittleren Anzahl an Merkmalen. Die Ursache dafür wird in Abb. 4.7b deutlich, in der der Modellfehler über dem Ausnutzungsgrad $\rho$ aufgetragen ist. Der Ausnutzungsgrad hängt dabei linear von $n_c$ ab und wird durch die Taktfrequenzkombination von VLIW und SDE zusätzlich beeinflusst (siehe Gl. 4.14). Eine hohe Abweichung des Modells von den Messwerten entsteht insbesondere bei hohen Werten von $\rho$. In diesem Bereich ist die Gesamtlaufzeit maßgeblich durch den VLIW definiert, sodass die reale Verteilung der Merkmale im Bild und das Verhältnis von stabilen zu verworfenen Merkmalen einen hohen Einfluss hat, welcher nicht durch das Modell abgebildet wird, da er datenabhängig ist und nicht vorhergesagt werden kann.

Insgesamt fällt auch auf, dass für kleinere $\rho$ der Modellfehler nahe 0 liegt. Diese Tatsache ist sogar unabhängig von $n_c$ und den Taktfrequenzen. In diesem Bereich ist die Gesamtlaufzeit durch die Laufzeit des SDE bestimmt, die durch das Modell verlässlich vorhergesagt werden kann, da sie architekturabhängig und deterministisch ist. Somit sind für eine hohe Verlässlichkeit der Modellierung Betriebspunkte des Systems mit $\rho < 0,6$ am besten geeignet.

### Erreichbare Bildraten

Bisher wurde nur der Modellfehler als relativer Bildratenfehler betrachtet, nicht aber die Bildrate selbst. Dazu zeigt Abb. 4.8 die beim Durchlaufen von Testsequenz 19 erreichte mittlere Bildrate $R_{img}$ über $f_{SDE}$ bzw. $\rho$ für verschiedene Werte von $f_{VLIW}$.

Für kleine Werte von $f_{SDE}$ bzw. $\rho$ ist die Gesamtlaufzeit durch den SDE bestimmt, was zu einer zunächst linear ansteigenden Bildrate führt. Für größere Werte erreicht die Bildrate eine Sättigung, in der die maximale Bildrate durch $f_{VLIW}$ definiert wird. Der VLIW kann die vom SDE generierten Merkmalskandidaten nicht mehr schnell genug verarbeiten. Eine Erhöhung der Taktfrequenz des SDE hat keine Bildratensteigerung zur Folge. Die mit der hier betrachteten Prozessorkonfiguration und Pipeline maximal erreichbare Bildrate liegt bei etwa 150 fps. Insbesondere durch die Modellkorrektur nach Gl. 4.24 kann der gemessene Bildratenverlauf sehr gut nachgebildet werden.

Darüber hinaus zeigt Abb. 4.9 die Bildrate in Abhängigkeit von $n_c$. Die Veränderung des zugrunde liegenden Ausnutzungsgrads an jedem Punkt ist mittels Konturlinien dargestellt. Bei hohen Werten von $n_c$ und ebenfalls hohem $\rho$ ergibt sich in der Messung für den identischen Wert von $n_c$ keine eindeutige, sondern eine gestreute Bildrate, wie aufgrund der Modellfehleranalyse zu erwarten war. Bei geringerem Ausnutzungsgrad ergeben sich zwar konstante, jedoch nicht so hohe Bildraten. Insgesamt sind die erreichbaren Bildraten bei weniger Merkmalen deutlich größer, da sich das System dort im linearen Bereich befindet und somit niemals durch den VLIW limitiert ist. Dieser ist der Bottleneck im System. Wie bereits in Tab. 4.2 gezeigt, sind nur durch den SDE allein sehr viel größere Bildraten erzielbar (bis zu 344 fps für die hier verwendete Bildauflösung), was in der Realität lediglich bei einer geringen Anzahl an Merkmalen im Bild erreicht werden kann.

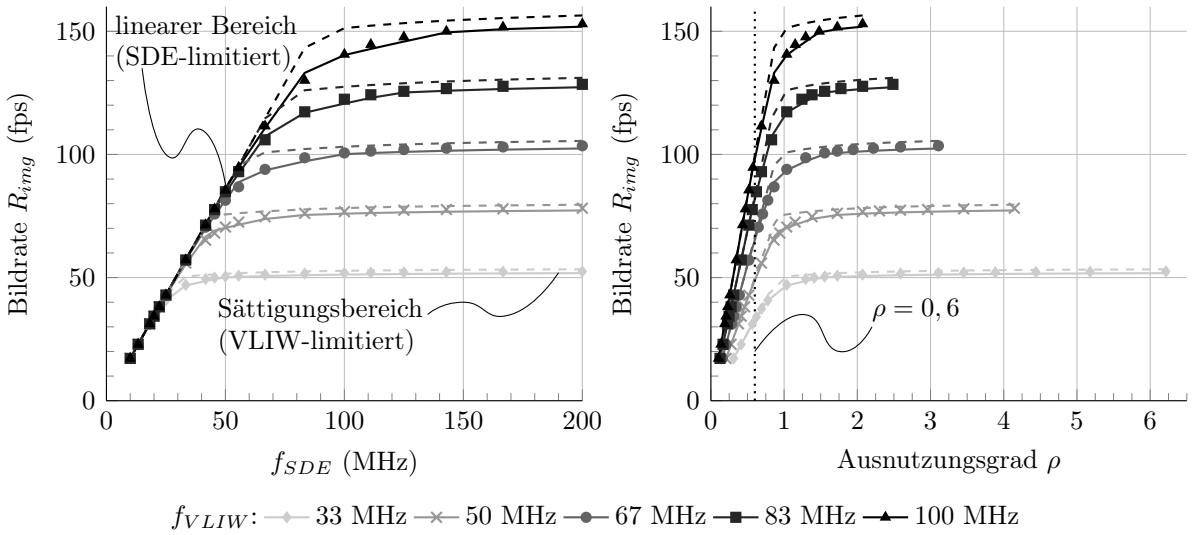



**Abbildung 4.8:** Mittlere Bildrate für verschiedene Taktfrequenzen beim Durchlaufen von Testsequenz 19. Die Symbole stehen für Messwerte, die gestrichelten Linien für unkorrigierte Modellwerte und die durchgezogenen Linien für korrigierte Modellwerte nach Gl. 4.24.

#### Fazit

Um höchste Bildraten zu erzielen, muss ein Betriebspunkt mit hohem $\rho$ gewählt werden, allerdings ist der Modellfehler hier am größten. Sind geringere Bildraten für eine gewählte Anwendung ausreichend und ist eine präzise Modellierung notwendig, so sollte, wie im vorangegangen Abschnitt zum Modellfehler diskutiert, $\rho < 0,6$ sein. Das System ist im Bezug auf Wartezeiten der beiden Komponenten VLIW-Prozessor und SDE bei $\rho \approx 1$ in der besten Balance. Dieser Betriebspunkt entspricht dem Knickpunkt zwischen linearem und Sättigungsbereich in Abb. 4.8.

### 4.2.4 Inverses Laufzeitenmodell

Bei der Anwendung des Modells interessiert oft auch die Umkehrfunktion der abgeleiteten Modellgleichungen. Ausgehend von einer Spezifikation soll unter bestimmten Randbedingungen eine Systemkonfiguration ermittelt werden, mit der die Spezifikation erreicht bzw. eingehalten werden kann.

Eine typische Vorgabe ist eine Zielbildrate $R_{soll}$. Diese sollte von einem System sicher erreicht werden können, sodass das System konservativ für eine etwas höhere Bildrate ausgelegt sein muss. Es ergibt sich folgender Zusammenhang:

$$R_{soll} = \frac{1}{t_{R,soll}} \overset{!}{\leq} \frac{1}{t_{img,model,korr}} = R_{img,model} \tag{4.26}$$

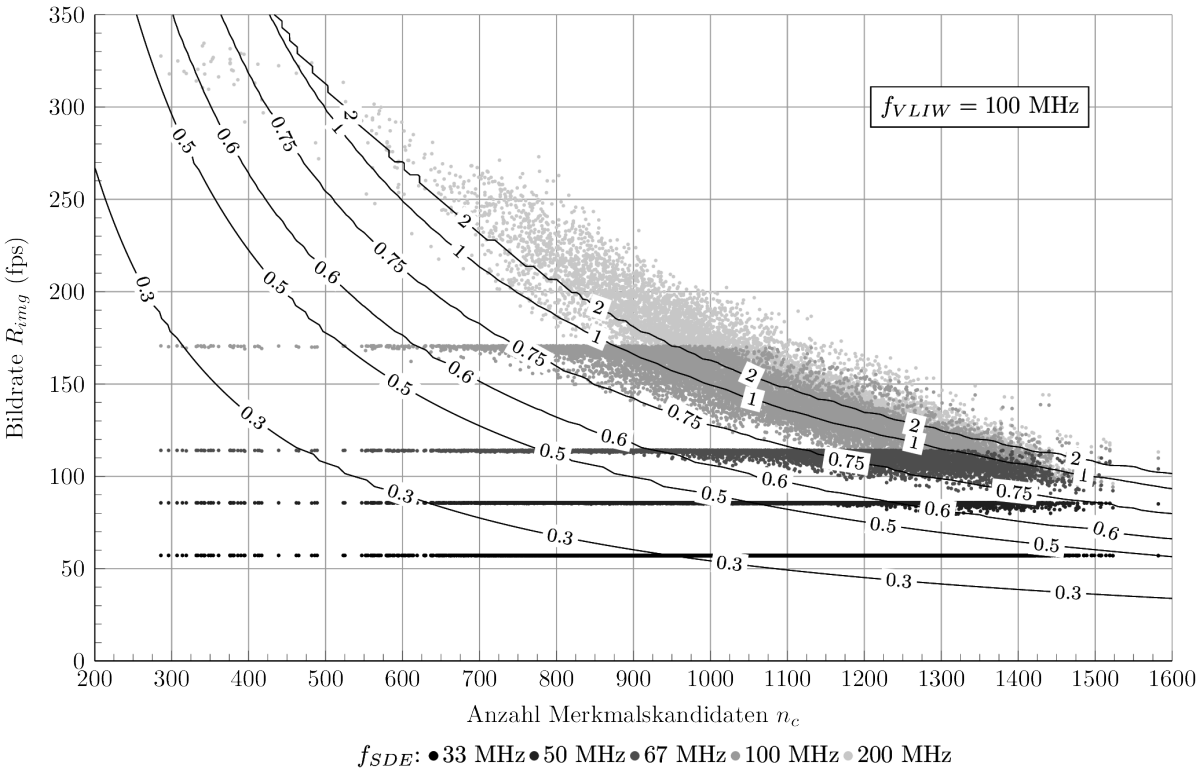



**Abbildung 4.9:** Gemessene Bildraten für jedes Einzelbild der Testsequenz 19 bei verschiedenen Taktfrequenzen. Die Konturlinien stellen den Ausnutzungsgrad dar, der sich durch die Modellbildung ergibt.

Weitere Zielvorgaben sind hier die Anzahl an Merkmalskandidaten $n_c$, bei der die Bildrate noch erreichbar sein muss. Da diese von den Eingangsdaten abhängt, muss auch hier eine konservative Systemauslegung geschehen. Darüber hinaus ist der Betriebspunkt des Systems $\rho$ eine weitere Vorgabe.

Neben Architekturparametern, die zur FPGA-Synthesezeit festgelegt werden, müssen die beiden Taktfrequenzen für VLIW-Prozessor und SDE ermittelt werden. Diese sind zur Laufzeit des Systems bei aktuellen Werten von $n_c$ dynamisch anpassbar, um den spezifizierten Betriebspunkt $\rho$ noch einzuhalten. Es wird das *inverse Laufzeitenmodell* benötigt, aus dem sich die Taktfrequenzen ergeben. Dazu lässt sich Gl. 4.20 mithilfe von Gl. 4.14 umkehren. Die Bedingung zur Fallunterscheidung ergibt sich durch Umformung des Ausdrucks $t_{SDE,work} > t_{VLIW,total,min}$, Ersetzung der Taktfrequenzen anhand von Gl. 4.14 und Auflösung nach $\rho$:

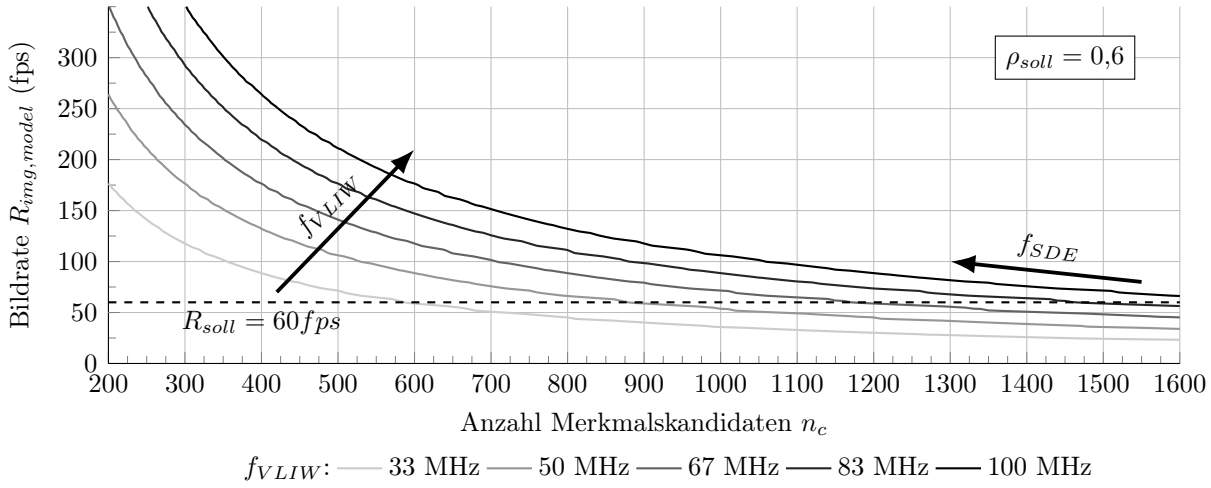



**Abbildung 4.10:** Modellgenerierter Entwurfsraum mit exemplarischen Zielvorgaben an das System. Die beiden einstellbaren Taktfrequenzen stehen in Abhängigkeit.

$$
\boldsymbol{f} = F(n_c,\ R_{soll},\ \rho_{soll})
$$
$$
= \begin{cases}
f_{SDE} \geq N_{SDE,work} \cdot R_{soll} \cdot \left(1 + \dfrac{\kappa(\rho_{soll})}{100}\right) \\
f_{VLIW} = \dfrac{n_c \cdot N_{cand}}{\rho_{soll} \cdot N_{SDE,work}} \cdot f_{SDE} \\
\qquad\qquad \text{wenn } \rho_{soll} < \frac{N_{cand}}{N_{cand}+N_{poll}+\frac{N_{pollToFirst}}{n_c}} \\
f_{VLIW} \geq (n_c \cdot (N_{cand} + N_{poll}) + N_{pollToFirst}) \cdot R_{soll} \cdot \left(1 + \dfrac{\kappa(\rho_{soll})}{100}\right) \\
f_{SDE} = \dfrac{\rho_{soll} \cdot N_{SDE,work}}{n_c \cdot N_{cand}} \cdot f_{VLIW} \\
\qquad\qquad \text{sonst}
\end{cases}
\tag{4.27}
$$

Abb. 4.10 verdeutlicht den modellgenerierten Entwurfsraum. Bei $R_{soll} = 60$ fps, $\rho_{soll} = 0{,}6$ und unterschiedlichen Werten von $n_c$ ergeben sich jeweils andere minimale Taktfrequenzen, die eingestellt werden müssen, damit das System nach Spezifikation arbeitet. Die beiden Taktfrequenzen stehen dabei in der aus Gl. 4.27 folgenden Abhängigkeit. Würde das System immer mit maximalen Taktfrequenzen betrieben werden, so würde sich der Betriebspunkt durchgehend in Abhängigkeit der Eingangsdaten ($n_c$) ändern, was aus Effizienzgründen (längere Wartezeiten im System) möglicherweise nicht gewollt ist.

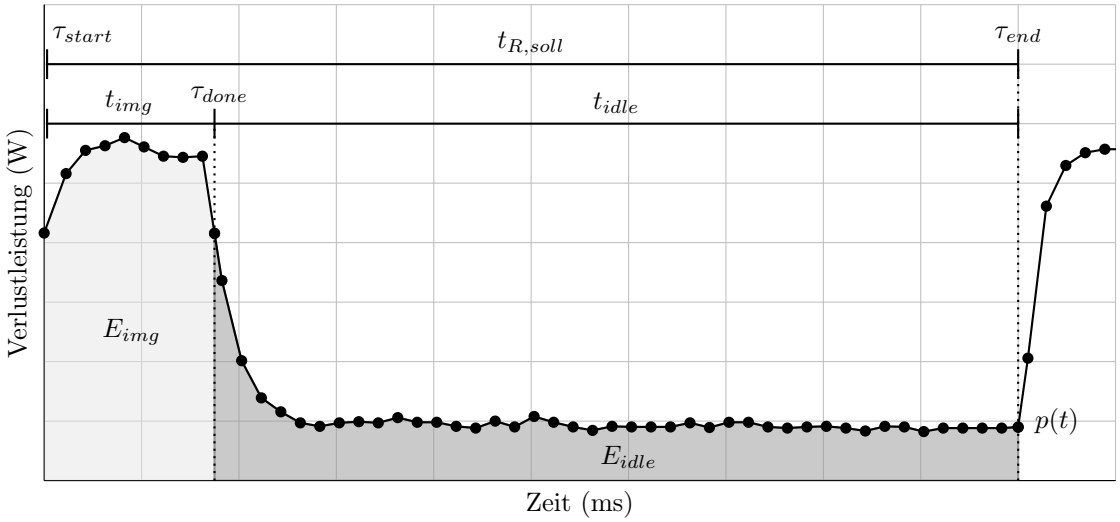



**Abbildung 4.11:** Exemplarischer Verlauf der Verlustleistung bei der Verarbeitung eines Bildes (Bildzyklus). Das System wechselt vom Verarbeitungsmodus in den Ruhemodus.

## 4.3 Modellierung der Verlustleistung

### 4.3.1 Beschreibung eines Bildzyklus

Neben der Laufzeit des Gesamtsystems für die Bildmerkmalsextraktion wird im Folgenden ein Modell für die Verlustleistungsaufnahme erarbeitet. Ziel der Verlustleistungsmodellierung ist eine valide Vorhersage der anfallenden Verlustleistung während eines Bildzyklus sowie die Berechnung der dabei umgesetzten Energie. Dafür muss zunächst definiert werden, wie sich die Verlustleistung zusammensetzt und wie ein Bildzyklus abläuft. Abb. 4.11 zeigt exemplarisch den Verlauf der Verlustleistung bei der Verarbeitung eines einzelnen Bildes.

Es wird zunächst eine Zielbildrate $R_{soll}$ spezifiziert, aus der sich die maximal erlaubte Verarbeitungsdauer $t_{R,soll}$ für ein Bild ergibt. Diese setzt sich zusammen aus der tatsächlich benötigten Laufzeit für die notwendigen Berechnungen $t_{img}$ und der Wartezeit im Ruhemodus $t_{idle}$ nach der Verarbeitung bis zum Ablauf der zeitlichen Zielvorgabe:

$$t_{R,soll} = t_{img} + t_{idle}. \tag{4.28}$$

Der Beginn der Verarbeitung liegt im Zeitpunkt $\tau_{start}$. Dort können die Taktfrequenzen für VLIW-Prozessor und SDE, mit denen das System arbeiten soll, konfiguriert werden. Danach folgt bis $\tau_{done}$ die eigentliche Berechnung. Nach dem Abschluss der Berechnung befindet sich das System in einer zweiten Phase, dem Ruhemodus. Dort können die Taktfrequenzen erneut umkonfiguriert werden. Das System verbleibt im Ruhemodus bis zum Ende der zeitlichen Vorgabe zum Zeitpunkt $\tau_{end}$. Danach beginnt die Verarbeitung eines weiteren Bildes und die Taktfrequenzen können wieder verändert werden. Die

Rekonfigurationszeiten sind im Vergleich zu den anderen Zeiten vernachlässigbar klein (<100 $\mu$s). Der zeitliche Ablauf bei der Kommunikation insbesondere zwischen VLIW-Prozessor und Host-PC wird in Abb. A.5 im Anhang detailliert dargestellt.

Im Allgemeinen setzt sich die Verlustleistung einer CMOS-Schaltung aus einem dynamischen Anteil $P_{dyn}$ und einem statischen Anteil $P_{static}$ zusammen [Vee17]:

$$\begin{aligned} P &= P_{dyn} + P_{static} \\ &= \sigma \cdot f \cdot (CU^2 + W_{sc}) + P_{static}. \end{aligned} \tag{4.29}$$

Dabei steht $\sigma$ für die Schaltaktivität, $f$ für die Taktfrequenz, $C$ für die durch die Logikressourcen anfallenden Kapazitäten, $U$ für die Versorgungsspannung und $W_{sc}$ für einen Kurzschlussanteil von Versorgungsspannung zu Ground während eines Umschaltvorgangs. $P_{static}$ entsteht durch Leckströme, die in nicht schaltenden, jedoch mit Spannung versorgten Bauteilen fließen, und geht somit einher mit der Anzahl an verwendeten Logikressourcen. Da das FPGA einen großen SRAM-basierten Konfigurationsspeicher besitzt, hat die statische Verlustleistung einen hohen Anteil an der Gesamtverlustleistung.

Aus der Teilung des Bildzyklus in zwei Bereiche resultiert zum einen ein Verlustleistungsanteil während der aktiven Verarbeitung eines Bildes und zum anderen ein Verlustleistungsanteil im Ruhemodus, welche jeweils separat modelliert werden. Der genaue Zeitpunkt der Teilung ist definitionsabhängig. Es ergeben sich $\overline{P_{img}}$ und $\overline{P_{idle}}$ als gemittelte Verlustleistung im jeweiligen Zeitintervall.

Die Gesamtenergie, die vom System während eines einzelnen Bildzyklus umgesetzt wird, entspricht in Abb. 4.11 der Fläche unterhalb der Verlustleistungskurve und berechnet sich zu

$$\begin{aligned} E_{total} &= E_{img} + E_{idle} \\ &= \int_{\tau_{start}}^{\tau_{done}} p(t)\ dt\ + \int_{\tau_{done}}^{\tau_{end}} p(t)\ dt \\ &= \overline{P_{img}} \cdot (\tau_{done} - \tau_{start}) + \overline{P_{idle}} \cdot (\tau_{end} - \tau_{done}). \end{aligned} \tag{4.30}$$

### 4.3.2 Verlustleistungsmessung

Zur Ableitung des Verlustleistungsmodells werden Messwerte der realen Leistungsaufnahme des System benötigt. Dazu werden auf der Zielplattform (Xilinx ML605 Board [Xil11b], siehe Abschnitt 3.3) die Trainingssequenzen und Testsequenz 19 durchlaufen. Der verwendete Virtex-6 FPGA wird durch einen Power Controller von TI [Tex08] mit Energie versorgt. Dieser liefert aktuelle Abtastwerte von Spannung und Strom und stellt diese über den PMBus (Power Management Bus basierend auf $I^2C$) zur Verfügung. Durch einen USB-Adapter [Tex06] werden diese Werte durch ein selbst entwickeltes PC-Programm ausgelesen. Die so erhaltenen Werte für die Verlustleistung werden dabei zusätzlich mit einem Zeitstempel versehen, sodass sie dem jeweiligen Zeitpunkt bzw. Systemzustand während der Bildverarbeitung zugeordnet werden können.

Die gemessene Verlustleistung entspricht derjenigen Leistung, die über die FPGA Power Rail *VCCINT* (1 V), die interne Versorgungsspannung des FPGAs, aufgenommen wird [Xil14b]. Der Power Controller selbst tastet den Strom in Intervallen von 600 $\mu s$ mit einem 12-bit ADC ab. Anschließend durchlaufen die Messwerte zur Rauschreduktion noch ein digitales Glättungsfilter (Zeitkonstante 2,1 ms) [Tex08]. Das Host-Programm, das über USB auf den PMBus zugreift, liefert etwa alle 2 ms einen neuen Abtastwert für die Verlustleistung mit einer Auflösung von 9 Bit (entspricht 1,95 mW).

### 4.3.3 Modellgleichungen

Aus Abb. 4.11 wird deutlich, dass ein Bildzyklus aus zwei Bereichen besteht. Dies ist zum einen der Bereich während der Bildverarbeitung und zum anderen der Ruhemodus, d. h. der Bereich bis zum Ablaufen der spezifizierten Zeitvorgabe zur Einhaltung der Zielbildrate. Beide Bereiche werden separat durch Regression modelliert.

#### Verlustleistung während der Bildverarbeitung

Als Grundlage für die Regression wird die mittlere Verlustleistung $\overline{P_{img}}$ durch Mittelwertbildung im entsprechenden Zeitintervall ermittelt. Die zeitliche Synchronisation ergibt sich durch Abgleichen der Zeitstempel während der Berechnung der Merkmale (Applikation) mit den abgetasteten Leistungswerten. Es werden Bilder aus den Trainingssequenzen (45 Trainingsbilder, siehe Abschnitt 4.3.4) für verschiedene Taktfrequenzen mit der gleichen Prozessorkonfiguration wie zur Laufzeitmodellierung durchlaufen.

Untersuchungen haben gezeigt, dass sich als Ansatz für die Modellgleichung und zur Nachbildung der partiellen Ableitung der Verlustleistung nach dem Ausnutzungsgrad eine vereinfachte Hyperbel eignet:

$$\frac{\partial P_{img,model}}{\partial \rho} = c_0 \cdot \frac{1}{\rho} \tag{4.31}$$

Die Ableitung wird durch Differenzenquotienten bestimmt und die Hyperbel zur Bestimmung von $c_0$ darauf angepasst. Dabei wird die MATLAB-Funktion *fminsearch* verwendet, welche intern die Simplex-Methode nach [Lag98] verwendet. Durch Integration ergibt sich

$$P^*(\rho) = \int c_0 \cdot \frac{1}{\rho}\, d\rho = c_0 \cdot \ln(\rho) + C. \tag{4.32}$$

Durch den Ausnutzungsgrad wird die Abhängigkeit von $n_c$ und dem Verhältnis der beiden Taktfrequenzen nachgebildet. Um den Einfluss der jeweiligen Taktfrequenzen separat in die Gleichung zu bringen, wird die Formel erweitert:

$$P_{img,model_0} = \underbrace{c_0 \cdot \ln(\rho)}_{P^*(\rho)} + c_1 \cdot f_{VLIW} + c_2 \cdot f_{SDE} + c_3. \tag{4.33}$$

$c_0$ ist dabei bereits bekannt. Für die übrigen drei Parameter ergibt sich folgendes überbestimmtes Gleichungssystem

$$\begin{pmatrix} f_{VLIW,0} & f_{SDE,0} & 1 \\ f_{VLIW,1} & f_{SDE,1} & 1 \\ \vdots & & \\ f_{VLIW,N} & f_{SDE,N} & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} \overline{P_{img,0}} - P_0^*(\rho) \\ \overline{P_{img,1}} - P_1^*(\rho) \\ \vdots \\ \overline{P_{img,N}} - P_N^*(\rho) \end{pmatrix} \tag{4.34}$$

mit den Messwerten $\overline{P_{img,0}}$ bis $\overline{P_{img,N}}$ aus dem Zeitintervall $[\tau_{start}, \tau_{done}]$. Durch die Methode der Minimierung der kleinsten Quadrate ergibt sich für die vorliegende Architektur die vollständige Modellgleichung

$$P_{img,model_0} = 88{,}6\ mW \cdot \ln(\rho) + 7{,}8\ \frac{mW}{MHz} \cdot f_{VLIW} + 1{,}1\ \frac{mW}{MHz} \cdot f_{SDE} + 2{,}1128\ W. \tag{4.35}$$

Diese wird im Folgenden *Modell_0* genannt.

Zusätzlich wurde noch ein zweiter Ansatz untersucht. Da sich die Komponenten VLIW-Prozessor und SDE jeweils immer zeitweise in einem arbeitenden Zustand und einem ruhenden Zustand befinden (vgl. Abb. 4.3), kann ein Modell auch wie folgt aussehen:

$$\begin{aligned} P_{img,model_1} = {} & c_0 \cdot r_{VLIW} \cdot f_{VLIW} + c_1 \cdot (1 - r_{VLIW}) \cdot f_{VLIW} \\ & + c_2 \cdot r_{SDE} \cdot f_{SDE} + c_3 \cdot (1 - r_{SDE}) \cdot f_{SDE} + c_4 \end{aligned} \tag{4.36}$$

$$\text{mit} \qquad r_{VLIW} = \frac{t_{VLIW,work}}{t_{img}} \tag{4.37}$$

$$\text{und} \qquad r_{SDE} = \frac{t_{SDE,work}}{t_{img}} \tag{4.38}$$

Die Verhältnisse $r_{VLIW}$ bzw. $r_{SDE}$ ergeben sich aus dem Laufzeitmodell. Analog zu Gl. 4.34 lässt sich auch hier ein überbestimmtes Gleichungssystem aufstellen, aus denen die Parameter $c_0$ bis $c_4$ ermittelt werden können. Die Modellgleichung ergibt sich hier zu

$$\begin{aligned} P_{img,model_1} = {} & 5{,}1\ \frac{mW}{MHz} \cdot r_{VLIW} \cdot f_{VLIW} + 5{,}7\ \frac{mW}{MHz} \cdot (1 - r_{VLIW}) \cdot f_{VLIW} \\ & + 4{,}5\ \frac{mW}{MHz} \cdot r_{SDE} \cdot f_{SDE} + 2{,}0\ \frac{mW}{MHz} \cdot (1 - r_{SDE}) \cdot f_{SDE} + 2{,}1057\ W \end{aligned} \tag{4.39}$$

Dieser Ansatz wird im Folgenden mit *Modell_1* bezeichnet und besitzt im Vergleich zu Modell_0 einen weiteren Parameter.

Die beiden Modellgleichungen werden für Testsequenz 19 evaluiert. Dazu ist in Abb. 4.12 die gemessene Verlustleistung $\overline{P_{img}}$ beim Durchlaufen der Sequenz für verschiedene Taktfrequenzen und über dem Ausnutzungsgrad dargestellt. Die Werte der Verlustleistung sind über die komplette Sequenz gemittelt. Die durchgezogenen Linien gehören zu Modell_0, die gestrichelten Linien zu Modell_1. Es fällt auf, dass Modell_0 in der Lage

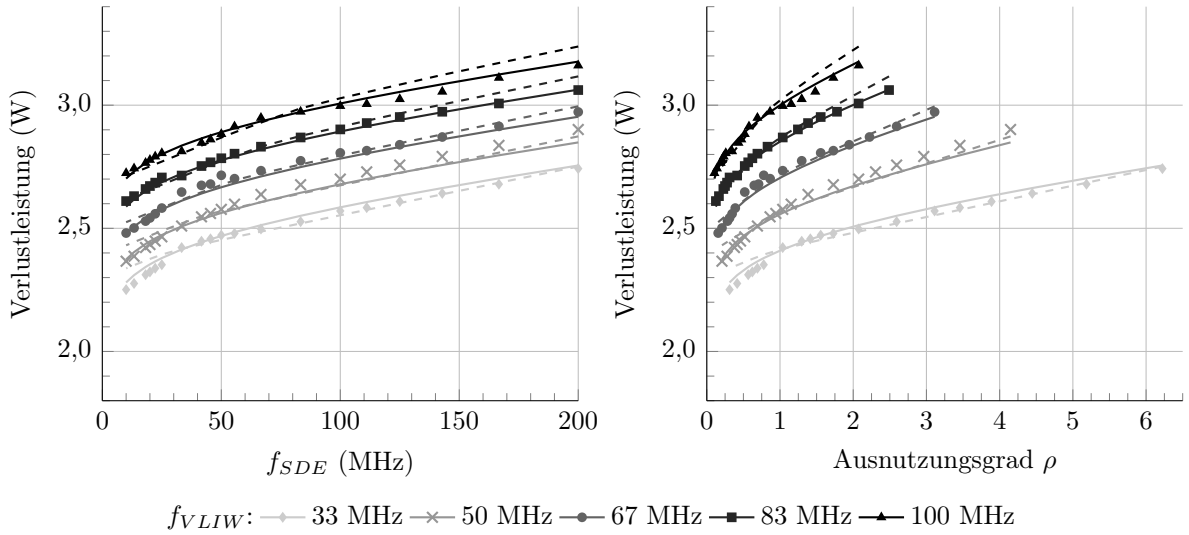



**Abbildung 4.12:** Gemessene Verlustleistung während der Bildverarbeitung beim Durchlaufen von Testsequenz 19 und modellierter Verlauf für Modell_0 (durchgezogene Linien) und Modell_1 (gestrichelte Linien). Die gemessenen Werte sind pro Taktkombination über alle Einzelbilder gemittelt.

ist, den gemessenen Verlauf der Verlustleistung etwas besser darzustellen. Insbesondere der Verlauf bei kleinen Werten von $f_{SDE}$ wird genauer nachgebildet. Der Modellfehler wird weiter unten diskutiert.

Während Abb. 4.12 über die ganze Sequenz gemittelte Werte der Verlustleistung darstellt, soll Abb. 4.13 die Einzelbilder betrachten. So, wie auch schon die Bildlaufzeit, unterliegt die Verlustleistung auch bei identischen Werten von $n_c$ einer Streuung. Insgesamt zeigt Modell_0 einen guten mittleren Verlauf. Modell_1 zeigt hier so starke Abweichungen, dass darauf verzichtet wird es hier darzustellen.

### Verlustleistung im Ruhemodus

Der exemplarische Bildzyklus in Abb. 4.11 zeigt neben dem Verarbeitungsmodus auch den Verlauf der Verlustleistung im Ruhemodus. Nach dem Ende der Bildverarbeitung wechselt das System in den Ruhemodus, und die gemessene Verlustleistung fällt aufgrund des internen Glättungsfilters exponentiell bis auf einen konstanten Wert ab. Als einfaches Modell könnte eine Konstante herangezogen werden. Allerdings wird mit einer konstanten Schätzung unter bestimmten Randbedingungen eine große Ungenauigkeit im Vergleich zur Messung in Kauf genommen. Ist die Zeitspanne $t_{idle}$ kurz, so fällt der Bereich des exponentiellen Abfalls bei der Berechnung einer mittleren Verlustleistung stärker ins Gewicht. Die Messwerte wurden hier bei einer Bildratenvorgabe von 10 fps ($t_{R,soll} = 100$ ms) erzeugt. Bei höheren Bildraten ist die Differenz $t_{R,soll} - t_{img}$ kleiner als im Beispiel, weshalb für ein allgemeines Modell eine Exponentialfunktion besser geeignet ist.

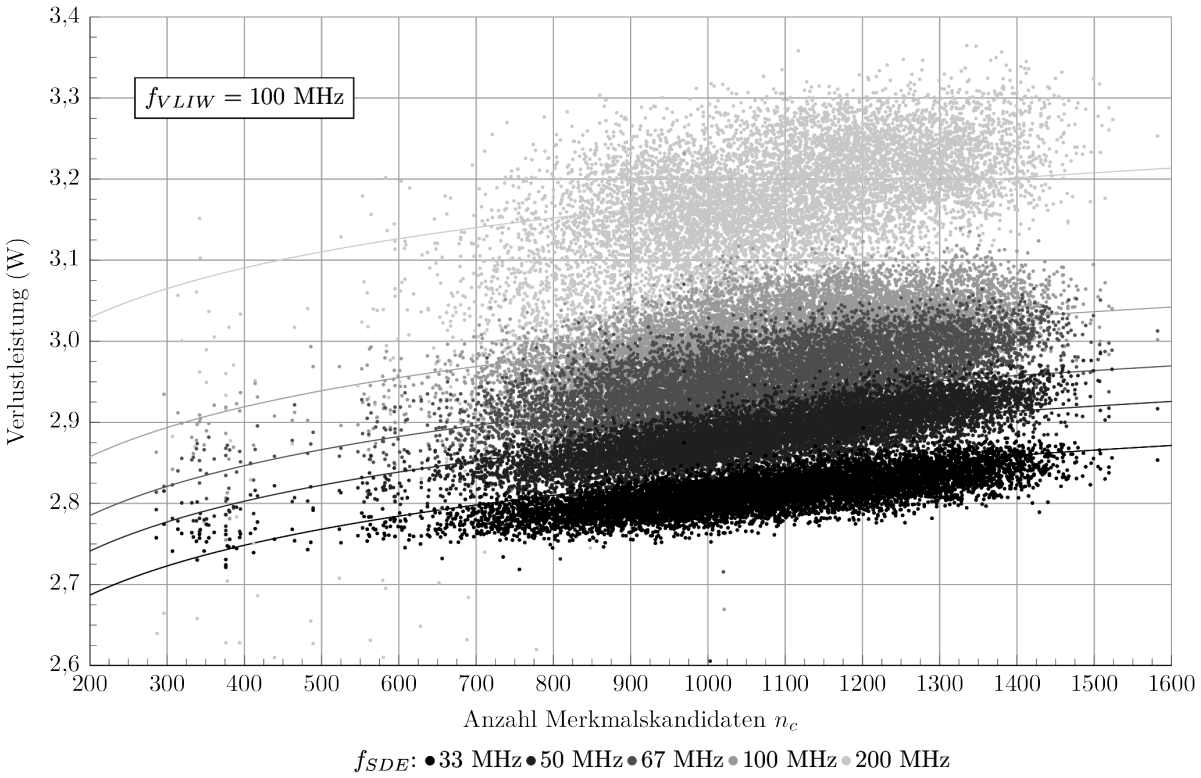



**Abbildung 4.13:** Gemessene Verlustleistung während der Verarbeitung für jedes Einzelbild der Testsequenz 19 bei verschiedenen Taktfrequenzen. Die durchgezogenen Linien zeigen den Verlauf von Modell_0.

Um den Verlauf der Verlustleistung im Ruhemodus nachzubilden, wird dabei folgender Ansatz gewählt:

$$P_{idle,model} = d_0 \cdot e^{-\frac{t}{d_1}} + \underbrace{d_2 \cdot f_{VLIW} + d_3 \cdot f_{SDE} + d_4}_{d_{idle,sat}(\boldsymbol{f})}. \tag{4.40}$$

Der Modellparameter $d_1$ entspricht der Zeitkonstanten im Glättungsfilter bei der Verlustleistungsmessung und wird daher zu 2,1 ms gewählt [Tex08] (vgl. Abschnitt 4.3.2).

$d_{idle,sat}$ stellt den Grenzwert der Verlustleistung im Zeitintervall $[\tau_{done}, \tau_{end}]$ dar. Dieser wird mittels Minimierung der kleinsten Quadrate anhand der Messwerte einer Trainingssequenz in einen frequenzabhängigen Ausdruck mit den Parametern $d_2$-$d_4$ entwickelt.

Für einen stetigen Verlauf der Verlustleistung hängt der Parameter $d_0$ von der Verlustleistung zum Zeitpunkt $\tau_{done}$ während der Verarbeitung des Bildes ab. Aufgrund dessen wird dieser Parameter mittels $P_{img,model}$ aus dem bestehenden Verlustleistungsmodell für

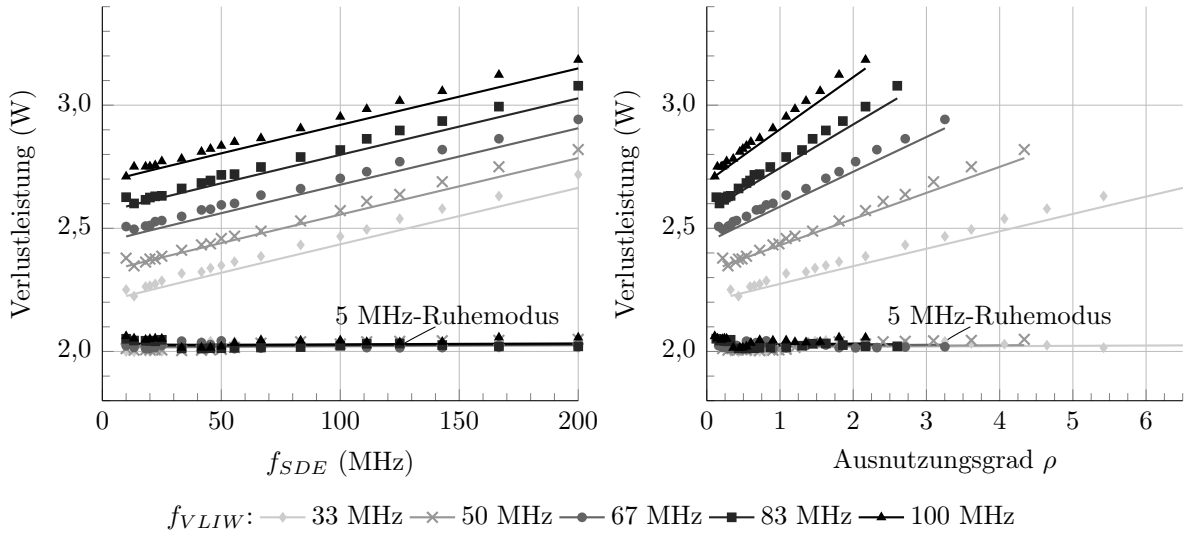



**Abbildung 4.14:** Gemessene Verlustleistung im Ruhemodus beim Durchlaufen von Testsequenz 19 und einer Bildratenvorgabe von 10 fps. Der modellierten Verläufe nach Gl. 4.40 sind als durchgezogene Linien zu sehen. Im 5 MHz-Ruhemodus sind VLIW-Prozessor und SDE mit 5 MHz getaktet.

die Bildverarbeitung konfigurationsabhängig angepasst:

$$P^*_{idle,model} = d^*_0 \cdot e^{-\frac{t}{2{,}1}} + d_{idle,sat}(\boldsymbol{f}) \tag{4.41}$$

$$\text{mit} \qquad d^*_0 = (P_{img,model} - d_{idle,sat}(\boldsymbol{f})) \cdot e^{\frac{\tau_{done}}{2{,}1}} \tag{4.42}$$

Die mittlere Verlustleistung im Ruhemodus für Testsequenz 19 bei einer Bildratenvorgabe von 10 fps ist in Abb. 4.14 dargestellt. Der modellierte Verlauf (45 Trainingsbilder, siehe Abschnitt 4.3.4) ist als Linie abgebildet. Es wird mithilfe des Laufzeitmodells für jede Taktkombination im Intervall $[\tau_{done}, \tau_{end}]$ die Energie als Integral über $P^*_{idle}$ nach der Zeit berechnet und durch $t_{idle}$ dividiert. Die Taktfrequenzen von VLIW-Prozessor und SDE sind dabei während der Bildverarbeitung und im Ruhemodus identisch. Zusätzlich ist ein *5 MHz-Ruhemodus* zu sehen, bei dem das System im Ruhemodus in einen Energiesparmodus wechselt. Dieser geht von Taktfrequenzen von 5 MHz für VLIW-Prozessor und SDE im Ruhemodus aus, unabhängig von der Taktfrequenz während der Verarbeitung.

Die Modellgleichung Gl. 4.40 ergibt sich bei der hier verwendeten exemplarischen Architekturkonfiguration quantitativ zu

$$P_{idle,model} = d_0 \cdot e^{-\frac{t}{2{,}1\ ms}} + 7{,}3\ \frac{mW}{MHz} \cdot f_{VLIW} + 2{,}3\ \frac{mW}{MHz} \cdot f_{SDE} + 1{,}9568\ W. \tag{4.43}$$

### Statische Verlustleistung

Als statische Verlustleistung wird hier derjenige Anteil der Gesamtverlustleistung betrachtet, der nicht von den beiden Takten $f_{VLIW}$ und $f_{SDE}$ abhängt. Dies umfasst somit nicht nur die durch Leckströme entstehende Leistung, sondern auch die Anteile aus den anderen Taktdomänen des Gesamtsystems. Modellhaft wird die statische Verlustleistung im Folgenden durch das Minimum aus den Konstanten $c_3$ (siehe Gl. 4.33) bzw. $c_4$ (siehe Gl. 4.36) und $d_4$ (siehe Gl. 4.40) abgeschätzt.

## 4.3.4 Modellfehler

Der relative Modellfehler $\Delta P$ für das Modell während der Bildverarbeitung ergibt sich als relative Abweichung des Modellwerts vom Messwert nach

$$\Delta P = \left(1 - \frac{\overline{P_{img}}}{P_{img,model}}\right) \cdot 100. \qquad [\%] \tag{4.44}$$

Der komplette Trainingsdatensatz besteht aus 44.200 Einzelbildern. Da die Ableitung der Modellgleichungen auf Regression basiert, wird hier untersucht, wie sich die Anzahl an für die Regression verwendeten Bildern des Datensatzes auf den Modellfehler auswirkt. Dazu wird zunächst die Liste aller Einzelbilder zufällig durchmischt. Danach werden von vorne beginnend die Einzelbild-Messwerte für die Verlustleistung bis zu einem jeweiligen Anteil der Liste für die Regression herangezogen (z. B. nur 10% der Einzelbilder). Dies wird für alle betrachteten Taktfrequenzen durchgeführt und mit 100 verschiedenen Durchmischungen der Liste wiederholt. Aus den generierten Modellfunktionen werden der mittlere absolute Fehler (engl. mean absolute error, kurz MAE) und der maximale absolute Fehler (engl. peak absolute error, kurz PAE) berechnet:

$$\mathrm{MAE}(x) = \frac{1}{100} \cdot \sum_{n=1}^{100} |\Delta P_n(x)| \tag{4.45}$$

$$\mathrm{PAE}(x) = \max_{n=1..100}(|\Delta P_n(x)|) \tag{4.46}$$

mit $x$ der verwendeten Anzahl an Trainingsbildern aus dem Trainingsdatensatz zur Generierung der Modellfunktion und $n$ dem Index der randomisierten Bildsequenzliste. Die Ergebnisse dieser Untersuchung sind in Abb. 4.15 mit Mittelwert und Standardabweichungen über die 100 Durchläufe für die beiden Modellierungsansätze Modell_0 und Modell_1 dargestellt. Die Messwerte werden beim Durchlaufen von Testsequenz 19 ermittelt.

Insgesamt liegt der MAE für Modell_0 mit 0,5% im Mittel und 1,2% im Maximum niedriger als bei Modell_1. Schon bei sehr wenigen Trainingsbildern ergibt sich ein kleiner mittlerer MAE, was die Modellierungsdauer stark beschleunigen kann. In vorherigen Abbildungen wurde der 0,1%-Wert (45 Bilder) für die Modellableitung gewählt. Dort liegt

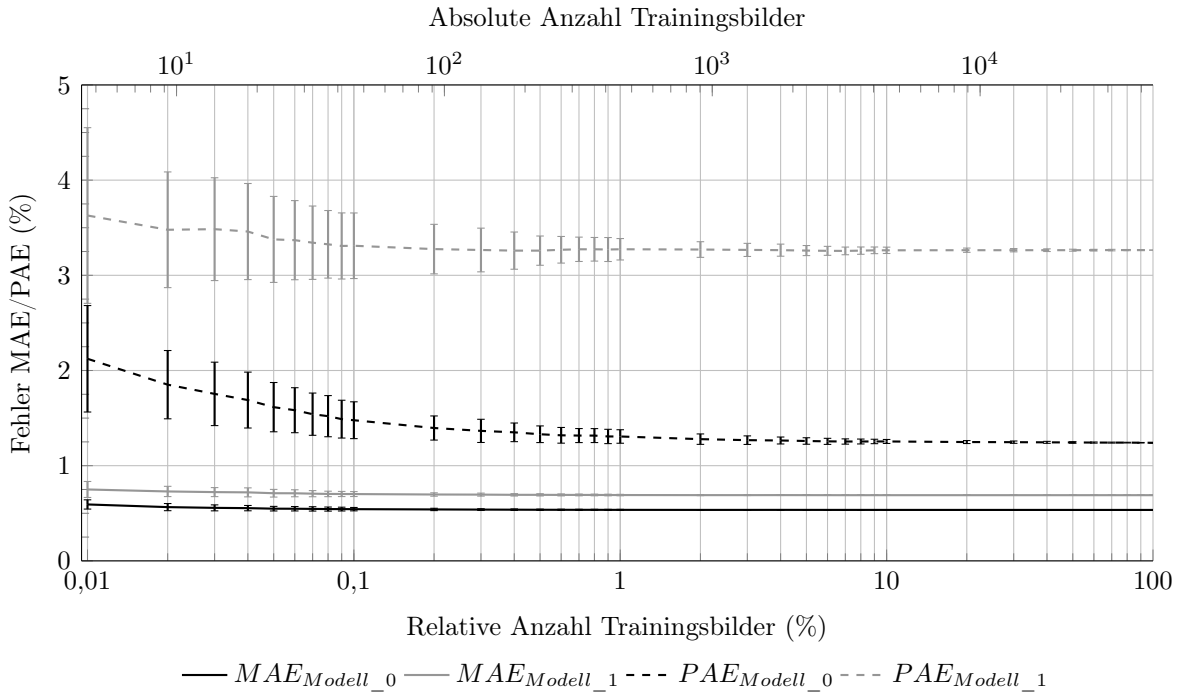



**Abbildung 4.15:** Mittlerer absoluter (MAE) und maximaler absoluter (PAE) Modellfehler der Verlustleistung während der Bildverarbeitung. Zur Generierung der Modellgleichungen wird eine unterschiedliche Anzahl an Trainingsbildern verwendet. Gezeigt sind Mittelwert und Standardabweichung über 100 Durchläufe mit unterschiedlich durchmischten Bildsequenzlisten des Trainingsdatensatzes (insgesamt 44.200 Einzelbilder) für Modell_0 und Modell_1.

der MAE für Modell_0 ebenfalls bei 0,5% und der PAE mit 1,5% bei einer Standardabweichung von 1,5% etwas höher als bei Verwendung des kompletten Trainingsdatensatzes mit allen Bildern. Aufgrund des niedrigeren Modellfehlers wird im Folgenden ausschließlich Modell_0 zur Verlustleistungsmodellierung verwendet.

Der Modellfehler pro Einzelbild beim Durchlaufen von Testsequenz 19 über Werte von $n_c$ wird in Abb. 4.16 gezeigt. Bei hohen Werten von $f_{SDE}$ und geringen für $n_c$ ergeben sich Ausreißer zum ansonsten gleichmäßig niedrigen Verlauf des Modellfehlers über jede Anzahl von Merkmalen hinweg. Es zeigt sich damit keine merkliche Abhängigkeit des Modellfehlers von den Eingangsdaten.

Analog zu den Fehlern während der Verarbeitung können MAE und PAE im Ruhemodus bestimmt werden. Dort liegen für 45 Bilder zur Modellgenerierung der MAE für Testsequenz 19 bei 0,8% und der PAE bei 2,0%.

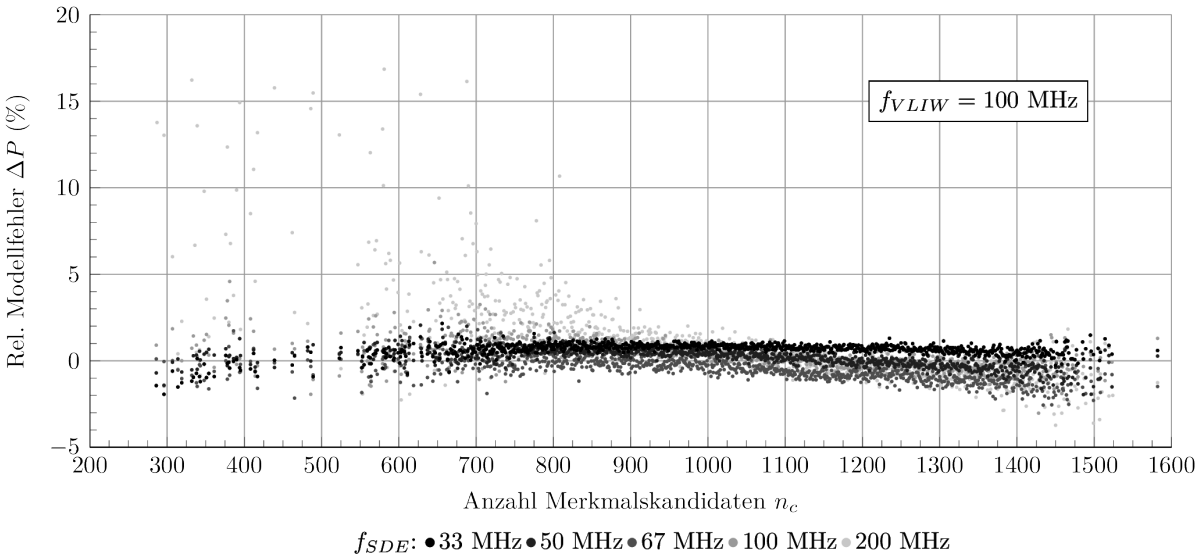



**Abbildung 4.16:** Rel. Modellfehler (Modell_0) für jedes Einzelbild der Testsequenz 19 bei verschiedenen Taktfrequenzen

## 4.4 Modellableitung für neue Systemkonfigurationen

Das Gesamtmodell des Systems besteht aus insgesamt 14 Parametern.

- Der SDE besitzt keine Modellparameter. Die Laufzeit ist deterministisch. Die Architekturabhängigkeiten werden mittels Designparametern zur Synthese exakt beschrieben.
- $N_{cand}$ beschreibt die Ausführungsstatistik im VLIW-Prozessor. Der Wert wird durch Profiling bestimmt und ist spezifisch für die Prozessor- und Pipeline-Konfiguration.
- Die Modellkorrektur des Laufzeitmodells nach Gl. 4.25 hat sechs Parameter. Diese werden empirisch ermittelt und enthalten keinen Bezug zu tatsächlichen Systemparametern.
- Die Verlustleistung während der Bildverarbeitung wird durch vier ($c_0$-$c_3$) und im Ruhemodus durch drei ($d_2$-$d_4$) Parameter modelliert ($d_0$ ist abhängig von der Verlustleistung während der Bildverarbeitung und $d_1$ ist konstant). Diese ergeben sich durch Regression und sind spezifisch für die jeweilige Systemkonfiguration.

Die Modellierung in diesem Kapitel erfolgte anhand einer Beispielkonfiguration des Systems mit einer ausgewählten Prozessorkonfiguration ($c2x2$) und Parametern des SDE (kein TDM). Eine Verallgemeinerung für neue Konfigurationen lässt sich nicht gänzlich analytisch herstellen, da zum einen der genetische Algorithmus im Instruction Scheduling und zum anderen das Place&Route beim Erzeugen des Bitstreams von Zufallsvariablen beeinflusst wird. Somit muss für eine neue Systemkonfiguration eine erneute Charakterisierung durchgeführt werden.

Diese Charakterisierung kann allerdings sehr effizient gestaltet werden. Für jede Konfiguration des VLIW-Prozessor muss ein Wert für $N_{cand}$ ermittelt werden, welche in Tab. 4.3 bereits für die hier betrachteten Prozessorkonfigurationen zur Verfügung stehen. Bei Veränderungen der Pipeline des VLIW-Prozessor kann der Wert auf Basis eines neuen Schedulings geschätzt werden. Somit ist keine Emulation, sondern nur ein Scheduling für eine neue Pipeline-Konfiguration erforderlich. Da die Modellkorrektur des Laufzeitmodells keine Systemparameter enthält, wird davon ausgegangen, dass für neue Systemkonfigurationen die gleiche Fehlercharakteristik entsteht und somit auch die Korrekturkurve gleich ist.

Für die Parameter des Verlustleistungsmodells wurde gezeigt, dass es ausreicht, lediglich 0,1% der Bilder (45) des Trainingsdatensatzes bei verschiedenen Taktfrequenzen zu durchlaufen. Da die Taktfrequenzen zur Laufzeit verändert werden können, muss dafür das FPGA nicht neu programmiert werden.

Der Vorteil der Modellfunktionen ist ihre Flexibilität. Aufgrund der hohen Anzahl an möglichen Systemkonfigurationen können der Entwurfsraum durch die analytischen Modellfunktionen schnell durchsucht und Designvarianten miteinander verglichen werden. Der Aufwand wäre viel höher, wenn zuvor jede Variante mit einer hohen Anzahl an Messungen untersucht werden müsste.

# 5 Modellgestützte Exploration des Entwurfsraums

In diesem Kapitel wird die in Kapitel 3 entworfene Architektur mithilfe der in Kapitel 4 erarbeiteten Modelle hinsichtlich unterschiedlicher Designparameter evaluiert und der sich ergebende Entwurfsraum aufgezeigt. Dazu wird zunächst in Abschnitt 5.1 das Vorgehen erläutert und es werden Zielindikatoren definiert. Danach erfolgen in Abschnitt 5.2 eine systematische Untersuchung und Optimierung verschiedener statischer Designparameter des Systems, die zur Designzeit geändert werden können. Abschnitt 5.3 evaluiert das Potential einer dynamischen Rekonfiguration der Taktfrequenzen zur Laufzeit. Abschließend erfolgt in Abschnitt 5.4 eine Einordnung der hier präsentierten Architektur in den aktuellen Stand der Forschung.

## 5.1 Methodologie

Um die Effektivität der getroffenen Designentscheidungen beim Entwurf der FPGA-Architektur zur Bildmerkmalsextraktion zu bewerten, sind reale Implementierungen und ein Benchmarking in Emulation unumgänglich. Untersucht werden sollen zum einen die in Tab. 3.4 unterschiedenen VLIW-Prozessorkonfigurationen, um den Einfluss von Parallelisierung und Spezialisierung zu beurteilen. Außerdem werden für jede Prozessorkonfiguration verschiedene Pipeline-Konfigurationen betrachtet. Zum anderen werden im SDE die verschiedenen TDM-Varianten untersucht. Darüber hinaus sind wesentliche Designparameter die Taktfrequenzen $f_{VLIW}$ und $f_{SDE}$ der beiden Architekturkomponenten, welche den FIFO-Ausnutzungsgrad $\rho$, d. h. den Betriebspunkt, bestimmen. Eine Übersicht über die Designparameter des Systems ist in Tab. 4.1 dargestellt.

Für jede Architekturkonfiguration bestehend aus VLIW-Prozessorkonfiguration, VLIW-Pipeline-Konfiguration und TDM-Variante wird eine FPGA-Synthese durchgeführt und ein Bitstream erzeugt (Timing Constraint 5 ns für VLIW-Prozessor und SDE). Dabei werden mithilfe des Xilinx-spezifischen Mapping-Parameters *Placer Cost Table* [Xil13] jeweils zehn funktional identische Bitstreams mit unterschiedlichen Platzierungen und dadurch bedingten unterschiedlichen realen Timings generiert. Dies stellt zum einen sicher, dass für jede Konfiguration eine Abbildung auf das FPGA gelingt, und verbessert zum anderen das Timing des Bitstreams, da derjenige Bitstream mit dem kürzesten kritischen Pfad in der VLIW-Domäne ausgewählt werden kann. Jede Veränderung am VLIW-Prozessor erfordert ein eigenes Instruction Scheduling.

Für einen Bitstream erfolgt eine *Modellierung* nach dem in Abschnitt 4.4 beschriebenen Vorgehen. Der für den VLIW-Prozessor charakteristische Wert $N_{cand}$ kann durch Offline-Schätzung erfolgen, was lediglich eine Analyse des Instruction Schedulings erfordert. Das Verlustleistungsmodell wird durch die Prozessierung von 45 zufälligen Bildern aus allen Testsequenzen generiert. Diese Bilder sind für jeden betrachteten Bitstream identisch. Die 45 Bilder werden dabei für eine Reihe von unterschiedlichen Taktperioden für VLIW-Prozessor und SDE prozessiert.[1]

Die erstellten Modelle dienen der einfacheren Navigation entlang der verschiedenen Designparameter des Entwurfsraums, da im Vergleich zur Messung Zwischenpunkte leichter erreicht werden können. Eine *modellgestützte Systemoptimierung* erfolgt durch Auswertung der Zielindikatoren mithilfe der jeweiligen Modellgleichungen an diskreten, real implementierbaren Betriebspunkten. Da das Verlustleistungsmodell auf Regression basiert, gilt die Einschränkung seiner Gültigkeit nur innerhalb eines bestimmten Bereichs des Entwurfsraums, d. h. nur innerhalb der zur Regression betrachteten Designparameterintervalle. Aufgrund der Nichtlinearitäten im realen System ergibt sich nur eine begrenzte Möglichkeit zur Extrapolation in zuvor unbekannte Regionen des Entwurfsraums.

Im Anschluss an die Modellauswertung erfolgt eine *emulationsbasierte Validierung* durch Prozessierung der Testsequenz 19. Informationen zu den verwendeten Bildsequenzen sind im Anhang in Abschnitt A.5 zu finden.

Die Laufzeitenmessung wird in Abschnitt 4.2.3 auf Seite 91 beschrieben. Auf die Messung der Verlustleistung wird in Abschnitt 4.3.2 genauer eingegangen. Im Rahmen dieser Arbeit wurde auch eine grafische Benutzeroberfläche zur Verlustleistungsmessung entwickelt, welche im Anhang in Abschnitt A.3 präsentiert wird.

### Modellgestützte Systemoptimierung

Das methodische Vorgehen einer *modellgestützten Systemoptimierung* ist in Abb. 5.1 in abstrahierter Weise dargestellt und wird im Folgenden im Kontext der hier entworfenen FPGA-Architektur erläutert. Die Anwendung der Methodik ist hier lediglich exemplarisch. Das Vorgehen ist nicht auf diese spezielle Fallstudie beschränkt.

Den Rahmen jeder Implementierung bildet eine *Anwendungsspezifikation*, welche den funktionalen Umfang angibt. Im vorliegenden Fall umfasst dies den Algorithmus (SIFT) sowie die zu erreichende algorithmische Qualität (vgl. Abschnitt 3.7). Zudem ergibt sich hier eine zu erwartende maximale Anzahl an Bildmerkmalen bzw. die minimal notwendige Anzahl an Merkmalen, um die algorithmische Qualität zu erreichen. Diese ist abhängig von algorithmischen Parametern und den jeweiligen Eingangsbilddaten. Die Merkmalsanzahl

[1] Untersuchte Perioden VLIW-Prozessor in ns: 9, 10, 12, 15, 20, 25, 27, 30, 35, 40, 50
Untersuchte Perioden SDE in ns (kein TDM): 100, 72, 54, 48, 42, 36, 30, 24, 18, 16, 12, 10, 9, 8, 7, 6, 5
Untersuchte Perioden SDE in ns (TDM×2): 100, 72, 54, 48, 42, 36, 30, 24, 18, 16, 12, 10, 8, 6
Untersuchte Perioden SDE in ns (TDM×3): 96, 72, 54, 48, 42, 36, 30, 24, 18, 15, 12, 9
Untersuchte Perioden SDE in ns (TDM×6): 96, 72, 54, 48, 42, 36, 30, 24, 18

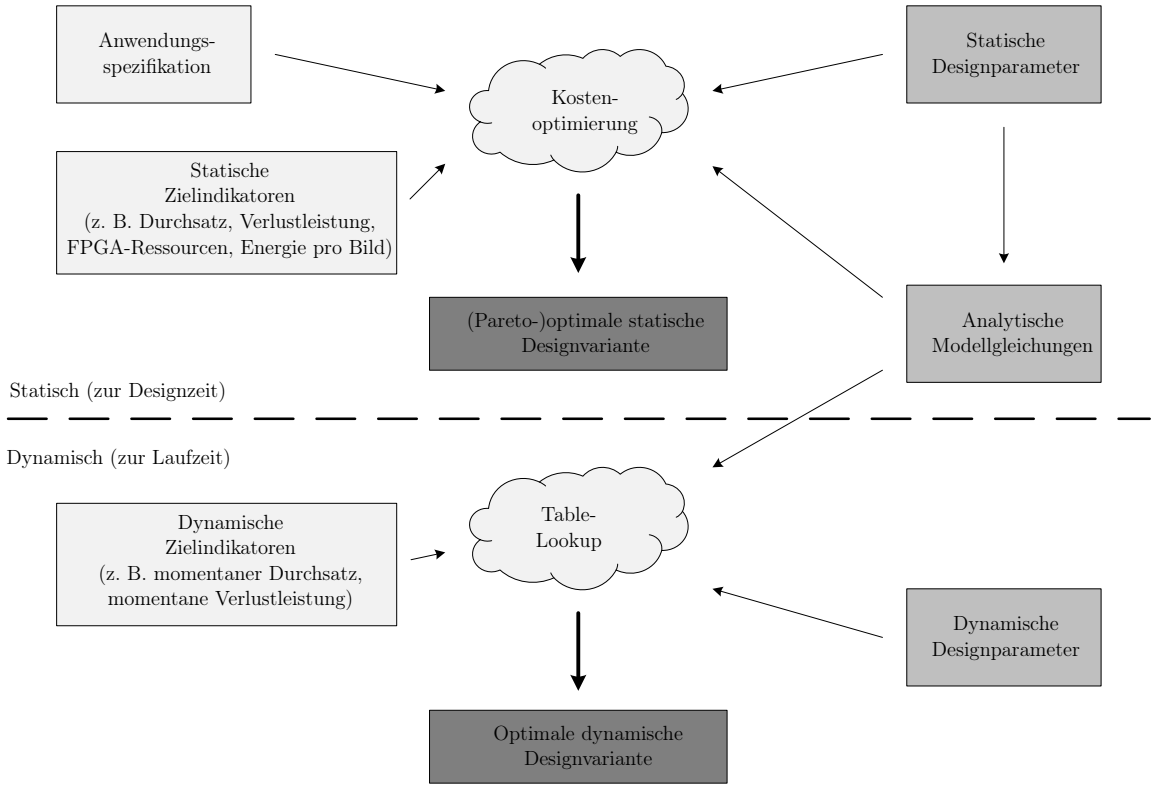



**Abbildung 5.1:** Modellgestützte Systemoptimierung

steht im Zusammenhang mit der zu realisierenden Rechenleistung der Implementierung. Auf Basis der Spezifikation wird eine parametrisierte Implementierung erstellt. *Statische Zielindikatoren* stellen neben der Anwendungsspezifikation weitere Randbedingungen an die Implementierung. Primär sind dies hier die Performance-Anforderung, d. h. der Durchsatz bzw. die Bildrate, die Verlustleistung, die Energie pro Bild und die FPGA-Ressourcen. Die VLIW-Rechenzeit pro Merkmalskandidat spielt eine sekundäre Rolle.

Zur Designzeit können im System verschiedene *statische Designparameter* (vgl. Tab. 4.1) verändert werden. Die Bildauflösung (1024×368 px) ergibt sich aus dem verwendeten Benchmark. Auch die Struktur der Gauß-Pyramide wird hier nicht weiter untersucht. Die hier betrachteten veränderlichen Parameter sind die VLIW-Prozessorkonfiguration, die VLIW-Pipeline-Konfiguration und die TDM-Variante im SDE. Darüber hinaus können die Taktfrequenzen verändert werden. Auch algorithmische Parameter, wie z. B. Schwellwerte, stellen Designparameter dar, welche hier aber nicht betrachtet werden.

Die Designparameter dienen als Eingang in *analytische Modellgleichungen*, welche eine Vorhersage der Zielindikatoren ermöglichen. Die Ableitung dieser Gleichungen ergibt sich entweder direkt aus der Struktur des Systems oder erfordert eine Reihe von Referenzimplementierungen, die dann der Modellbildung dienen.

Im Zentrum steht eine *Kostenoptimierung*, die für gegebene Randbedingungen (Anwendungsspezifikation und vorgegebene Zielindikatoren) die *Pareto-optimalen statischen Designvarianten* mit ihren jeweiligen Designparametern liefert. Das Laufzeit-Modell wird auf inverse Weise genutzt, sodass für eine Performance-Vorgabe die Taktfrequenzen bestimmt werden können. Die finale Auswahl einer passenden Designvariante ist schließlich eine Abtauschentscheidung und eine Frage der Priorisierung der Zielindikatoren.

Die Zielindikatoren, die zur Designzeit gelten, können sich durch dynamische Einflüsse zur Laufzeit verändern, wodurch sich *dynamische Zielindikatoren* ergeben. Im vorliegenden Fall sind die erkannten Merkmale abhängig von den Eingangsdaten, wodurch die benötigte Rechenlast nicht konstant ist und damit die Performance-Anforderung an das System schwankt. Es besteht die Möglichkeit, die *dynamischen Designparameter* zur Laufzeit anzupassen. Dynamische Parameter sind hier die Taktfrequenzen des VLIW-Prozessor und des SDE. Auch eine dynamische Anpassung algorithmischer Parameter kann sinnvoll sein. Beim dynamischen Ansatz bildet den Kern ein *Table-Lookup*, da zur Laufzeit keine komplexe Suche nach Optima möglich ist. Mithilfe der Modelle wird im Vorfeld eine Liste von jeweils Pareto-optimalen Designvarianten mit Sätzen von dynamischen Designparametern generiert. Die jeweilige *optimale dynamische Designvariante* wird in Abhängigkeit der dynamisch vorliegenden Gegebenheiten ausgewählt. In dieser Arbeit werden die Taktfrequenzen in Abhängigkeit von der Anzahl an Merkmalskandidaten und der aktuellen Fahrgeschwindigkeit verändert.

### Bildzyklus und Definition der Messgrößen

Im Folgenden werden Werte für Bildrate, Verlustleistung und Energie verwendet, deren physikalische Bedeutung hier noch einmal ergänzend zu Abschnitt 4.3.1 dargestellt wird. Abb. 5.2 zeigt dazu die Verlustleistung während eines exemplarischen, idealisierten Bildzyklus. Die Zielbildrate $R_{soll}$ ergibt sich aus dem Kehrwert von $t_{R,soll}$. Die tatsächliche Dauer der Berechnung $t_{img}$ (mit der dazugehörigen tatsächlichen Bildrate $R_{img}$) muss zum Erreichen der Vorgabe darunter liegen, sodass ein Zeitbereich $t_{idle}$ (Ruhemodus) entsteht, in dem das System auf den Ablauf der Zeitvorgabe wartet. Die Verlustleistung während der Verarbeitung $P_{img}$ liegt höher als die Verlustleistung im Ruhemodus $P_{idle}$. Diese Werte der Verlustleistung beinhalten die statische Verlustleistung $P_{static}$, die während des Betriebs des Systems anfällt.

Auch die Energie gliedert sich in einen statischen und einen dynamischen Anteil:

$$E_{total} = E_{dyn} + E_{static} \tag{5.1}$$

$$\text{mit} \quad E_{dyn} = E_{dyn,img} + E_{dyn,idle} \tag{5.2}$$

$$\text{und} \quad E_{static} = P_{static} \cdot t_{R,soll} \tag{5.3}$$

$$\text{und} \quad E_{dyn,img} = (P_{img} - P_{static}) \cdot t_{img} \tag{5.4}$$

$$\text{und} \quad E_{dyn,idle} = (P_{idle} - P_{static}) \cdot t_{idle} \tag{5.5}$$

$$\text{und} \quad t_{idle} = t_{R,soll} - t_{img} \tag{5.6}$$

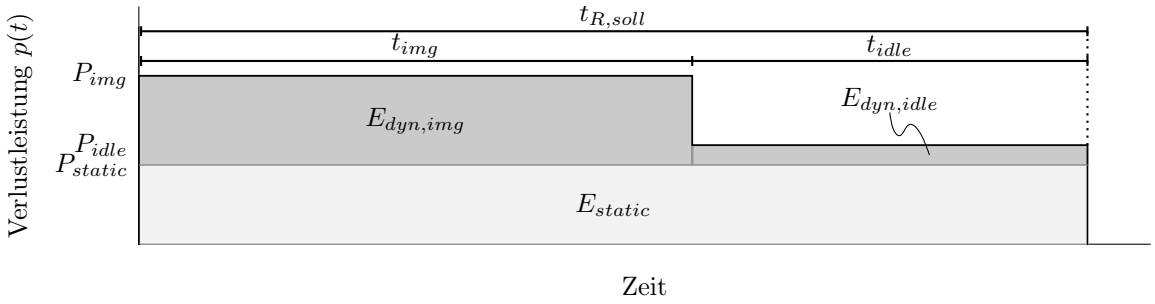



**Abbildung 5.2:** Idealisierter Verlauf der Verlustleistung während eines Bildzyklus

Die Bestimmung der einzelnen Anteile ergibt sich für die reale Messung wie folgt. Die Verlustleistung $p(t)$ wird durchgehend abgetastet und dem jeweiligen Bildzyklus eines Bildes zugeordnet. Jeder Messwert ist mit einem Zeitstempel versehen. Zudem wird die Zeit $t_{img}$ durch Performance-Counter zur Laufzeit gemessen. $t_{idle}$ ergibt sich nach Gl. 5.6. $P_{static}$ resultiert aus dem zugehörigen Verlustleistungsmodell (vgl. Seite 109 in Abschnitt 4.3.3). Die Gesamtenergie $E_{total}$, die während eines Bildzyklus aufgewendet wurde, wird durch grafische Integration aus den Verlustleistungswerten mit ihren Zeitstempeln ermittelt. Nach Subtraktion des statischen Anteils ergibt sich die dynamische Energie $E_{dyn}$. Die Aufteilung der Energie in $E_{dyn,img}$ und $E_{dyn,idle}$ ist anhand des Verhältnisses $t_{img}/t_{R,soll}$ möglich. Die jeweilige Verlustleistung $P_{img}$ bzw. $P_{idle}$ ergibt sich dann durch Division mit $t_{img}$ bzw. $t_{idle}$. Damit entsprechen die Werte der Verlustleistung einem Mittelwert für den Verarbeitungs- bzw. Ruhebereich in einem Bildzyklus.

Betrachtet werden im Folgenden insbesondere die durchschnittliche dynamische Verlustleistung eines Einzelbildes

$$P_{dyn,avg} = \frac{E_{total}}{t_{R,soll}} - P_{static}, \tag{5.7}$$

sowie die maximale dynamische Verlustleistung pro Einzelbild

$$P_{dyn,max} = \max_{0<t<t_{R,soll}} ( \, p(t) \, ) - P_{static}. \tag{5.8}$$

Die Testsequenz 19 umfasst 9.962 Einzelbilder. Der Mittelwert einer Messgröße über alle Einzelbilder der Sequenz wird mit einem Oberstrich $\overline{\ }$ über dem Formelzeichen angezeigt.

Um das Modell gegenüber der Messung zu validieren, wird folgendes relatives Fehlermaß verwendet:

$$\Delta = \left(1 - \frac{Messwert}{Modellwert}\right) \cdot 100 \qquad [\%] \tag{5.9}$$

Ein positiver Fehler bedeutet, dass das Modell die Messung überschätzt.

## 5.2 Statische Architekturoptimierung

### 5.2.1 Optimierung des Betriebspunktes

Es wird zunächst das Modell einer einzelnen Systemkonfiguration betrachtet. Diese besteht aus einem VLIW-Prozessor in der Basiskonfiguration #*b*0 mit optimierter Pipeline ($N_{cand} = 1.025$, $N_{SDE,work} = 570.045$, VLIW-Taktperiode $p_{VLIW,min} = 8{,}75$ ns, vgl. #*b*0_86 in Tab. A.5 im Anhang, $P_{static} = 1,94\ W$) und ohne TDM im SDE. Durch eine Veränderung der beiden Taktfrequenzen $f_{VLIW}$ und $f_{SDE}$ kann das System unterschiedliche Betriebspunkte einnehmen, welche zum einen zu einem veränderten FIFO-Ausnutzungsgrad $\rho$ nach Gl. 4.14 führen und zum anderen den Durchsatz des Systems bestimmen.

Abb. 5.3 zeigt die Verlustleistung während der Bildverarbeitung über der erreichten Bildrate für eine Vielzahl an unterschiedlichen Taktfrequenzkombinationen (Betriebspunkten). Die Darstellung beinhaltet eine Vielzahl von Informationen.

Die Verlustleistung des Systems ist abhängig von der Bildrate. Eine schnellere Verarbeitung erfordert schnellere Schaltvorgänge und damit höhere Ströme zum Umladen der Kapazitäten. Für eine exemplarische Anzahl an Merkmalskandidaten von $n_c = 1.072$ ergibt sich eine klare Pareto-Front, auf der nur bestimmte Betriebspunkte liegen. Die Front hat einen linearen Verlauf und kann durch eine Gerade angenähert werden:

$$P_{img} = E_{dyn,img} \cdot R_{img} + P_{static} = 12{,}5\ \text{mJ/Bild} \cdot R_{img} + 1{,}94\ \text{W} \tag{5.10}$$

Die Steigung der Geraden entspricht dabei der dynamischen Energie, die zur Berechnung der Merkmale eines Bildes *minimal* notwendig ist. Diese Energie ist für alle Bildraten konstant, da der Algorithmus deterministisch arbeitet und für identische Eingangsdaten identische Berechnungen durchführen muss, um dann dieselben Merkmale zu liefern. Wird die Energie bei kleineren Bildraten aufgewendet, so geschieht dies in einem längeren Zeitintervall als bei höheren, wodurch die Verlustleistung geringer ist. Die Gerade, die die Pareto-Front annähert, läuft durch diejenigen Betriebspunkte, die einem optimal balancierten System entsprechen ($\rho \approx 0{,}9$) und bei denen wenig Energie durch Wartezeiten aufgewendet wird. Dies entspricht einem Betriebspunkt im Knick zwischen linearem und Sättigungsbereich aus Abb. 4.8. Das System arbeitet auf der Geraden mit der höchsten Energieeffizienz, da die wenigste Energie pro Einzelbild notwendig ist.

Die Pareto-Front knickt bei hohen Bildraten ab (Sättigungsbereich). Eine Beschleunigung des SDE erhöht die Verlustleistung und sorgt dabei nur für eine geringe Steigerung der Bildrate. Trotzdem liegt der Betriebspunkt mit den maximalen Taktfrequenzen ($f_{VLIW} = 111$ MHz, $f_{SDE} = 200$ MHz, $\rho = 3{,}5$) aufgrund des höchsten Durchsatzes (95 fps) auf der Pareto-Front. Er liegt jedoch nicht auf der Geraden, weil die dynamische Energie pro Bild aufgrund des unbalancierten Betriebspunktes (längeren Wartezeiten von VLIW-Prozessor und SDE) höher ist. Die maximale Bildrate vor dem Knick der Pareto-Front, d. h. bei geringster Energie pro Bild, liegt zwischen 85 fps ($f_{VLIW} = 111$ MHz, $f_{SDE} = 53$ MHz, $\rho = 0{,}9$) und 93 fps ($f_{VLIW} = 111$ MHz, $f_{SDE} = 91$ MHz, $\rho = 1{,}5$).

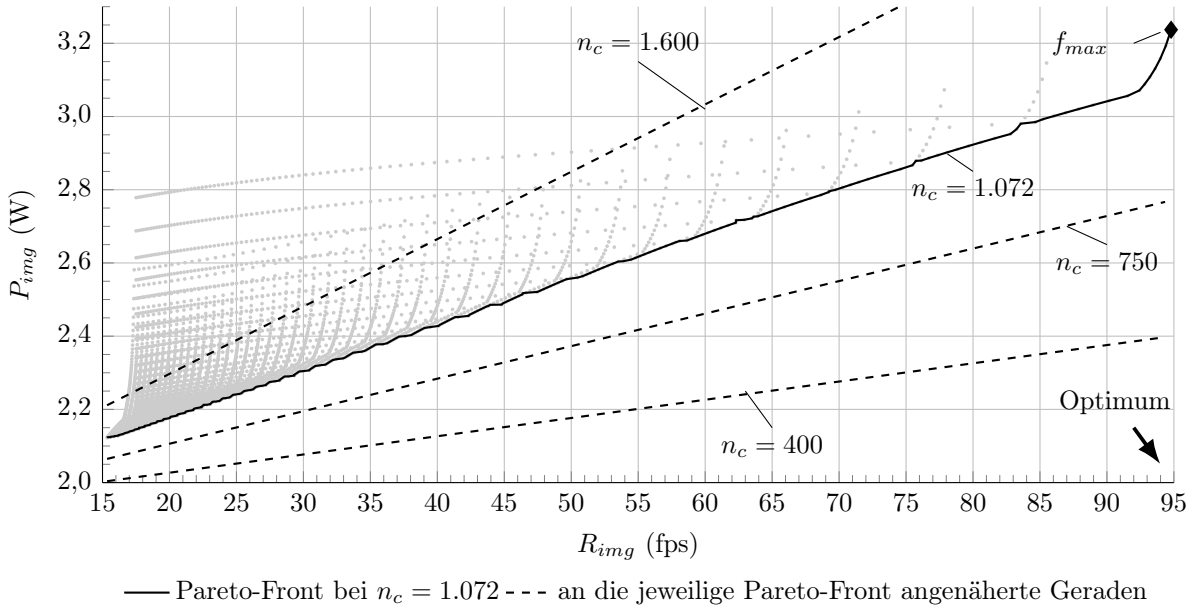



**Abbildung 5.3:** Verlustleistung über der erreichten Bildrate für verschiedene Anzahlen an Merkmalskandidaten (modellbasierte Untersuchung). Die Punkte entsprechen modellierten Taktfrequenzkombinationen (Betriebspunkten) bei $n_c = 1.072$.

Die Steigung der Geraden, also die minimale Energie pro Bild, ist abhängig von der Anzahl an Merkmalskandidaten, die im Bild gefunden werden. Bei einer größeren Anzahl an Kandidaten liegt diese Energie höher. Es ergibt sich jedoch aus dem Modell eine etwa konstante Energie von 11,2 $\mu$J/Kandidat bei einem konstanten Offset von 502,7 $\mu$J.

Wird die dynamische Energie pro Bild $E_{dyn,img}$ über dem Ausnutzungsgrad betrachtet (siehe Abb. 5.4a), so zeigt sich auch hier das Minimum von 12,5 mJ/Bild bei einem Wert von $\rho = 0,9$. Die dynamische Energie bei Nutzung der maximalen Taktfrequenzen liegt etwa 10% höher.

In einer realen Anwendung zur Bildverarbeitung bestimmt oft nicht die tatsächliche, maximal erreichbare Bildrate den Systemdurchsatz, sondern eine Zielvorgabe $R_{soll}$. Verschiedene Anteile der Gesamtenergie sind in Abb. 5.4b über $R_{soll}$ aufgetragen. Das System wird dabei mit maximalen Taktfrequenzen betrieben ($f_{VLIW} = 111$ MHz, $f_{SDE} = 200$ MHz, $n_c = 1.072$, $\rho = 3,5$). Für die sich ergebende Zeitspanne, während sich das System im Ruhemodus befindet, werden zwei Strategien verglichen:

- **wait**: eine Beibehaltung derselben Taktfrequenzen wie während der Bildverarbeitung
- **sleep**: eine Reduzierung der Taktfrequenzen auf 5 MHz

Die tatsächliche Bildrate liegt hier immer bei maximal 95 fps, sodass bei geringeren Zielbildraten das System einen signifikanten Anteil der Zeit im Ruhemodus verbringt. Es fällt auf, dass die Gesamtenergie vom statischen Energiebedarf bestimmt wird. Im

Mittel sind dies 60%, im Fall von kleinen Zielbildraten im *sleep*-Fall sogar bis zu 90% des Gesamtenergiebedarfs. Dies begründet sich im dynamische Energiebedarf im Ruhemodus $E_{dyn,idle}$. Dieser wird im *sleep*-Fall sehr klein, wodurch der statische Anteil relativ gesehen größer ist. Im *wait*-Fall nimmt $E_{dyn,idle}$ bereits einen großen Anteil ein. $E_{dyn,img}$ ist konstant, da die durchgeführten Berechnungen und der Betriebspunkt immer identisch sind. Die Reduktion der Gesamtenergie im *sleep*-Fall liegt abhängig von $R_{soll}$ bei bis zu 30 %, sodass in folgenden Untersuchungen nur noch der *sleep*-Fall betrachtet wird.

Der statische Anteil der Energie kann bei der hier vorliegenden FPGA-Technologie nur in geringem Maße beeinflusst werden, da der zugrundeliegende SRAM-basierte Konfigurationsspeicher einen wesentlichen Anteil an der Gesamtchipfläche hat und mit Energie versorgt werden muss. Für alle betrachteten Architekturkonfigurationen wurde eine statische Verlustleistung zwischen 1,9 W und 2 W durch die Modelle ermittelt. Im Folgenden liegt der Fokus auf dem dynamischen Anteil der Energie, da sich dort durch schaltungstechnische Maßnahmen Optimierungsmöglichkeiten ergeben.

Es wurde bereits deutlich, dass die Wahl des Betriebspunktes Einfluss auf den Energiebedarf zur Verarbeitung eines Bildes hat. Wird nun ein kompletter Bildzyklus der Dauer $t_{R,soll}$ betrachtet, indem auch der Ruhemodus in die Energiebetrachtung miteinbezogen wird, lassen sich daraus in Anlehnung an die Literatur (z. B. [Das16]) zwei Betriebsstrategien ableiten.

- **race-to-idle**: das System wird so schnell wie möglich getaktet, um eine schnellstmögliche Verarbeitung eines Bildes ($t_{img}$ so klein wie möglich) zu erreichen. Danach wird das System durch Heruntertakten für die Dauer von $t_{idle}$ in einen Ruhemodus mit sehr niedrigem Verbrauch versetzt.

- **slowdown**: das System wird so getaktet, dass ein Ruhemodus nach Möglichkeit entfällt ($t_{idle} \approx 0$).

Grundsätzlich sollte die dynamische Energie für beide Varianten gleich sein, wenn für *race-to-idle* von einer dynamischen Verlustleistung von 0 W im Ruhemodus ausgegangen wird. Zwei Aspekte sorgen allerdings dafür, dass *slowdown* hier weniger Energie benötigt. Zum einen ist die dynamische Verlustleistung im Ruhemodus bei noch 5 MHz nicht genau 0 W. Zum anderen sorgt ein unbalancierter Betriebspunkt, hier mit $\rho = 3,5$ bei maximalen Taktfrequenzen, für einen unnötig hohen Energiebedarf durch Wartezeiten im System. Abb. 5.4c stellt dies auf Basis der Modellierung dar. Die Datenreihe $f_{max,sleep}$ steht für die dynamische Energie bei Verwendung der maximalen Frequenzen und eines 5 MHz Ruhemodus (*race-to-idle*). Durch exakte Abstimmung der Taktfrequenzen auf einen bestimmten Betriebspunkt, sodass eine Bildrate von $R_{soll}$ sehr genau erreicht wird (*slowdown*), ergeben sich die Reihen $f_{\rho=0,6}$ und $f_{\rho=0,9}$. Die Zackenform resultiert daraus, dass nur ganzzahlige Periodenlängen betrachtet werden. $f_{\rho=0,6}$ hat genauso wie $f_{max,sleep}$ aufgrund des unbalancierten Betriebspunktes einen 20-60% höheren dynamischen Energiebedarf als $f_{\rho=0,9}$. Aufgrund der maximalen VLIW-Taktfrequenz von 111 MHz reduziert sich allerdings die maximale Bildrate, die für den jeweiligen Betriebspunkt noch erreicht

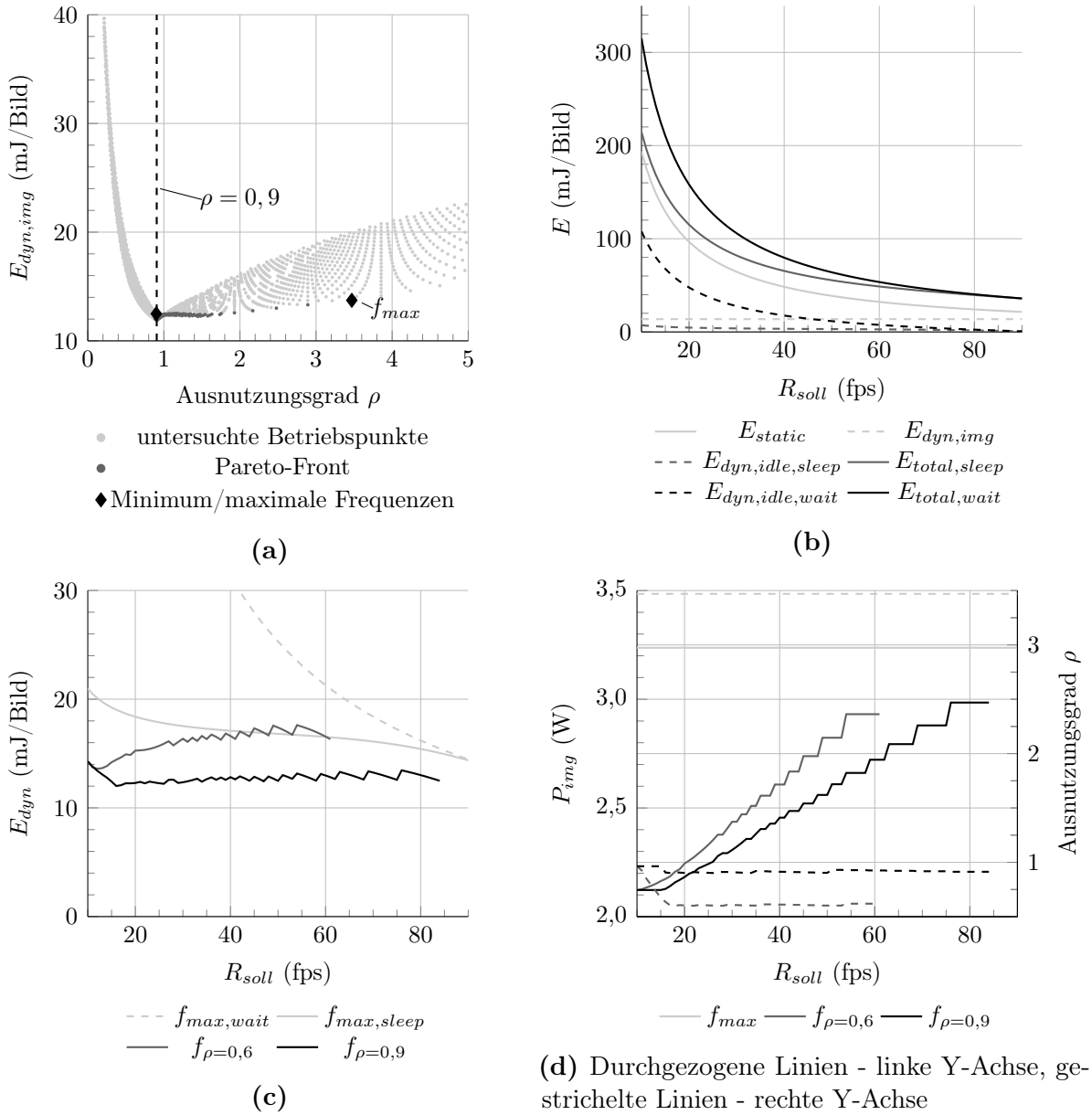



**Abbildung 5.4:** Modellbasierte Betrachtungen der Architektur bei $n_c = 1.072$

werden kann. Abb. 5.4d zeigt links die Verlustleistung und rechts den Ausnutzungsgrad für die drei Szenarien. Der Ausnutzungsgrad zeigt aufgrund der ganzzahligen Perioden bei kleinen $R_{soll}$ eine Abweichung vom Zielbetriebspunkt. Verlustleistung und Ausnutzungsgrad sind bei $f_{max}$ unabhängig von $R_{soll}$. $f_{\rho=0,9}$ erreicht $R_{soll}$ mit der kleinsten Verlustleistung, da die Energie pro Bild dort am geringsten ist.

Der modellbasierte Entwurfsraum, der sich für die zwei Zielindikatoren $R_{img}$, $P_{img}$ und den jeweiligen Ausnutzungsgrad $\rho$ ergibt, wenn die Designparameter $f_{VLIW}$ und $f_{SDE}$

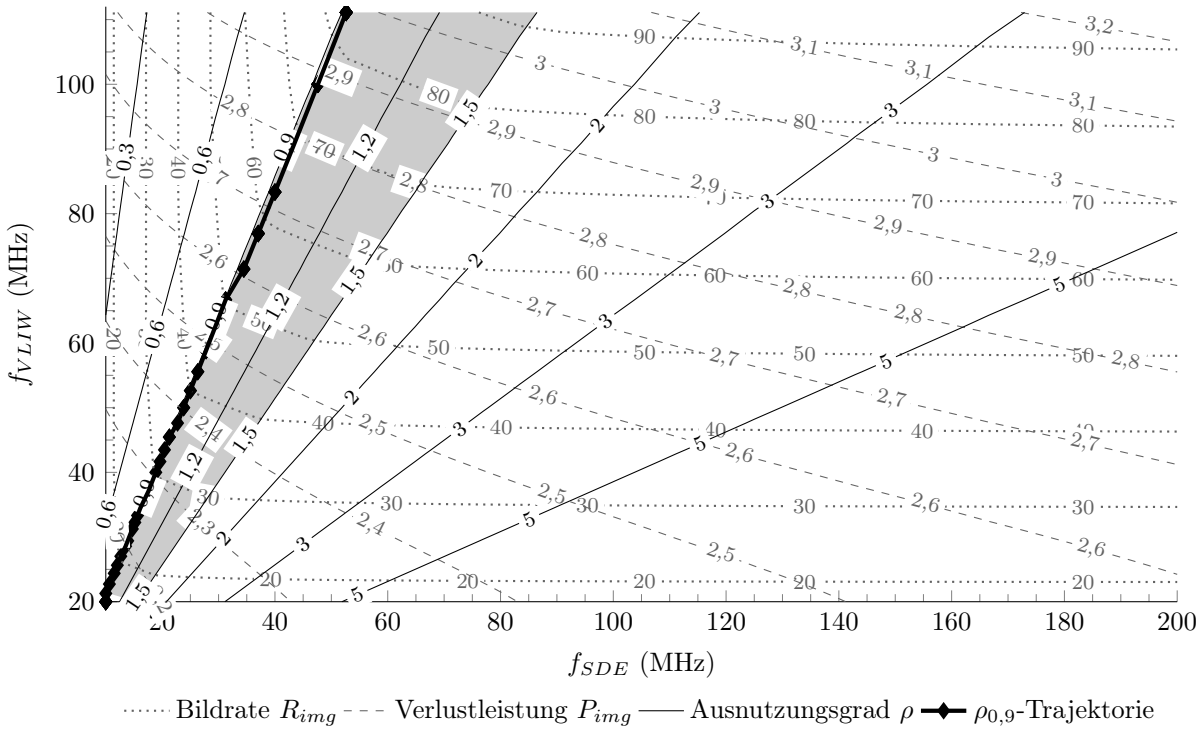



**Abbildung 5.5:** Modellbasierter Entwurfsraum zur Wahl des Betriebspunktes ($n_c = 1.072$). Der optimale Betriebskorridor zwischen $\rho = 0{,}9$ und $\rho = 1{,}5$ ist grau hinterlegt.

innerhalb ihres gültigen Intervalls verändert werden, ist in Abb. 5.5 gezeigt. Die Konturlinien definieren eine Landschaft, die bei Veränderung des Betriebspunktes durchwandert wird. Für die geringste Verlustleistung bei der höchsten Bildrate (entspricht der Energie pro Bild) ergibt sich ein optimaler Betriebskorridor zwischen $\rho = 0{,}9$ und $\rho = 1{,}5$. Die Betriebspunkte, die einzustellen sind, wenn bei konstantem Ausnutzungsgrad die Bildrate variiert wird (bei minimaler Energie pro Bild), sind als $\rho_{0,9}$-Trajektorie eingezeichnet. Insgesamt zeigt sich aus der Modellierung, dass ein balanciertes System immer zu bevorzugen ist. Wird die allerhöchste Bildrate benötigt, so muss der optimale Betriebskorridor verlassen und eine geringere Effizienz in Kauf genommen werden.

### Validierung

Um die modellbasierten Untersuchungen mit Messungen zu validieren, wird die Testsequenz 19 bei verschiedenen Zielbetriebspunkten $\rho_{soll}$ für $R_{soll} = 30$ fps prozessiert. Da die Anzahl an Merkmalskandidaten nicht konstant ist, sondern einer Verteilung entspricht (siehe Abb. A.19 im Anhang), werden die Taktfrequenzen nach Gl. 4.14 für $n_{c,max} = 1.600$ eingestellt, um 30 fps für jedes Einzelbild sicher zu erreichen. Dadurch ergibt sich ein im Mittel geringerer Ausnutzungsgrad $\overline{\rho_{ist}}$, der auch anhand der mittleren Merkmalsanzahl $n_c = 1.052$ errechnet werden kann. Die Ergebnisse der Validierungsläufe sind in Tab. 5.1

**Tabelle 5.1:** Messergebnisse bei Prozessierung von Testsequenz 19 bei verschiedenen Zielbetriebspunkten $\rho_{soll}$ für $R_{soll} = 30$ fps (Modellfehler nach Gl. 5.9 in Klammern)

| | $\rho_{soll}$ | $p_{VLIW}$ (ns) | $p_{SDE}$ (ns) | $\overline{\rho_{ist}}$ | $\overline{R_{img}}$ (fps) | $\overline{P_{dyn,avg}}$ (mW) | $\overline{P_{dyn,max}}$ (mW) | $\overline{E_{dyn}}$ (mJ/Bild) |
|---|---|---|---|---|---|---|---|---|
| | *max,wait* | 9 | 5 | 3,5 | 105 (-5,5%) | 1.369 (-10%) | 1.731 (-33%) | 45,7 (-10%) |
| | *max,sleep* | 9 | 5 | 3,5 | 105 (-5,5%) | 551 (-6,3%) | 1.636 (-26%) | 18,4 (-6,3%) |
| *slowdown* | 0,6 | 12 | 57 | 0,4 | 31 (-0,0%) | 714 (-5,4%) | 789 (-16%) | 23,8 (-5,4%) |
| | 0,9 | 16 | 51 | 0,6 | 34 (-0,8%) | 559 (-11%) | 678 (-28%) | 18,6 (-11%) |
| | 1,2 | 17 | 40 | 0,8 | 41 (-0,8%) | 474 (-8,4%) | 665 (-25%) | 15,8 (-8,4%) |
| | 1,5 | 18 | 34 | 1,0 | 44 (-1,9%) | 471 (-13%) | 682 (-29%) | 15,7 (-13%) |
| | 1,8 | 18 | 28 | 1,2 | 48 (-3,6%) | 463 (-10%) | 708 (-27%) | 15,4 (-10%) |
| | 2,1 | 18 | 24 | 1,4 | 49 (-4,5%) | 465 (-9,0%) | 731 (-27%) | 15,5 (-9,0%) |
| | 2,4 | 18 | 21 | 1,6 | 50 (-4,9%) | 483 (-11%) | 769 (-29%) | 16,1 (-11%) |
| | 2,7 | 18 | 19 | 1,8 | 51 (-5,1%) | 482 (-8,9%) | 777 (-28%) | 16,1 (-8,9%) |
| | 3,0 | 18 | 17 | 2,0 | 51 (-5,3%) | 489 (-8,4%) | 795 (-27%) | 16,3 (-8,4%) |
| | 3,3 | 18 | 15 | 2,3 | 52 (-5,5%) | 500 (-8,0%) | 818 (-27%) | 16,7 (-8,0%) |

zusammengestellt. Die Messwerte sind über alle Einzelbilder der Sequenz gemittelt. Die Abweichung zum Modellwert nach Gl. 5.9, d. h. demjenigen Wert, der durch Prozessierung der Sequenz mit dem Modell erhalten wird, ist in Klammern angegeben.

Die tatsächliche mittlere Bildrate $\overline{R_{img}}$ liegt immer über 30 fps aufgrund der *Worst Case* Parametrierung für $n_{c,max} = 1.600$. Die Zielvorgabe für die Bildrate wird somit erfüllt. *max*, *sleep* erreicht eine viel höhere Bildrate, da das System mit maximalem Takt arbeitet. Der Modellfehler liegt für die Bildrate im Maximum bei -5,5%.

Die mittlere dynamische Verlustleistung $\overline{P_{dyn,avg}}$ beträgt 463 mW bei einem mittleren Ausnutzungsgrad von $\overline{\rho_{ist}} = 1{,}2$. Im Vergleich zum *race-to-idle*-Fall $max, sleep$ mit 551 mW sind dies 16% weniger. Dies ist ein Resultat der System-Balancierung und der *slowdown*-Parametrierung, da die mittlere Bildrate wesentlich geringer ist. Der Modellfehler beträgt maximal -13% und im Mittel -9%.

Für die maximale, dynamische Verlustleistung $\overline{P_{dyn,max}}$ zeigt sich ein hoher Modellfehler, da der Messwert direkt aus den tatsächlichen Abtastwerten der Verlustleistung bestimmt wird. Diese Verlustleistungsspitzen werden durch das Modell nicht abgebildet. Der Modellwert hier entspricht der Verlustleistung während der Bildverarbeitung. Es zeigt sich, dass sich $\overline{P_{dyn,max}}$ durch den *slowdown* signifikant um bis zu 60% gegenüber $max, sleep$ reduzieren lässt.

Wird die mittlere dynamische Energie $\overline{E_{dyn}}$ betrachtet, so zeigt sich mit 15,4 mJ/Bild ebenfalls ein Minimum bei $\overline{\rho_{ist}} = 1{,}2$, welches im Vergleich zu $max, sleep$ (18,4 mJ/Bild) 16% niedriger liegt. Der Modellfehler ist identisch zu $\overline{P_{dyn,avg}}$, da die Datenbasis dieselbe ist.

### 5.2.2 Optimierung der VLIW-Architektur

Der VLIW-Prozessor wird hier als *Application-Specific Instruction-Set Processor (ASIP)* untersucht. In Tab. 3.4 werden dabei elf VLIW-Prozessorkonfigurationen unterschieden. Neben Duplizierung von grundlegenden funktionalen Einheiten und dem X2-Modus wird der Prozessor durch Hinzufügen von neuen funktionalen Einheiten an die Anwendung angepasst. Aus Gründen der Unterscheidbarkeit der Ergebnisse liegt der Fokus im Folgenden auf drei ausgewählten Konfigurationen. Alle weiteren Konfigurationen werden nur am Rande betrachtet, da sie lediglich Zwischenschritte im Entwurfsraum darstellen und lediglich weitere innere Punkte hinzufügen. Die drei ausgewählten Konfigurationen sind:

- $\#b0$: Damit die SIFT-Implementierung überhaupt in einem sinnvollen zeitlichen Rahmen durchzuführen ist, besitzt die hier betrachtete *Basiskonfiguration* des Prozessors bereits drei Spezialinstruktionen sowie zwei Koprozessoren zur Division und vier Koprozessoren zur Histogrammberechnung. Die Konfiguration wird im Folgenden als $\#_{base,pipeX}$ bezeichnet, wobei $X$ die Pipeline-Konfiguration bezeichnet.

- $\#b4x2$: Diese Architektur folgt einem Konzept stärkerer *Parallelisierung*. Durch Duplizierung der AU- und PERM-Unit sowie der Divisionskoprozessoren können bei vorhandener Datenparallelität zwei Instruktionen parallel vom Prozessor ausgeführt werden. Zudem wird der X2-Modus aktiviert, sodass zwei parallele Instruktionen mit konsekutivem Registerzugriff als eine Instruktion ausgeführt werden. Dadurch wird auch ein zeitgleicher Zugriff auf die Histogramm-Koprozessoren ermöglicht. Die Konfiguration wird im Folgenden als $\#_{par,pipeX}$ bezeichnet.

- $\#c2x2$: Zusätzlich zur Parallelisierung verfügt diese Konfiguration über acht *Spezialinstruktionen*, von denen zwei dupliziert werden. Die Konfiguration wird im Folgenden als $\#_{spez,pipeX}$ bezeichnet.

Für jede der drei Konfigurationen wird untersucht, inwieweit sich eine Veränderung der Pipeline auswirkt, mit dem Ziel, den kritischen Pfad des Prozessors soweit wie möglich zu verkürzen. Die dadurch entstehende Penalty aufgrund der Auswirkung auf den IPC wurde in Abschnitt 3.5.3 auf Seite 60 erläutert.

Das Vorgehen dabei ist empirisch, d. h. das Durchsuchen des Entwurfsraums entspricht einer geführten Suche. Für eine erste Auswahl von Pipeline-Konfigurationen werden zehn Bitstreams mit jeweils anderem Placer Cost Table Parameter generiert und der kritische Pfad analysiert. Für den schnellsten wird eine Modellierung durchgeführt, was Lauffähigkeit in Emulation erfordert. Auf Basis dieser Ergebnisse werden weitere Pipeline-Konfigurationen abgeleitet, sodass im Rahmen dieser Arbeit in Summe 86 verschiedene Pipeline-Konfigurationen pro VLIW-Prozessorkonfiguration bewertet wurden. Für alle drei VLIW-Konfigurationen werden dieselben Pipeline-Konfigurationen untersucht. Eine detaillierte Übersicht der Konfigurationen sowie eine Liste der Pipeline-Schalter wird in Abschnitt A.4.2 im Anhang gegeben. Abb. 5.6 zeigt die Ergebnisse dieser Untersuchung.

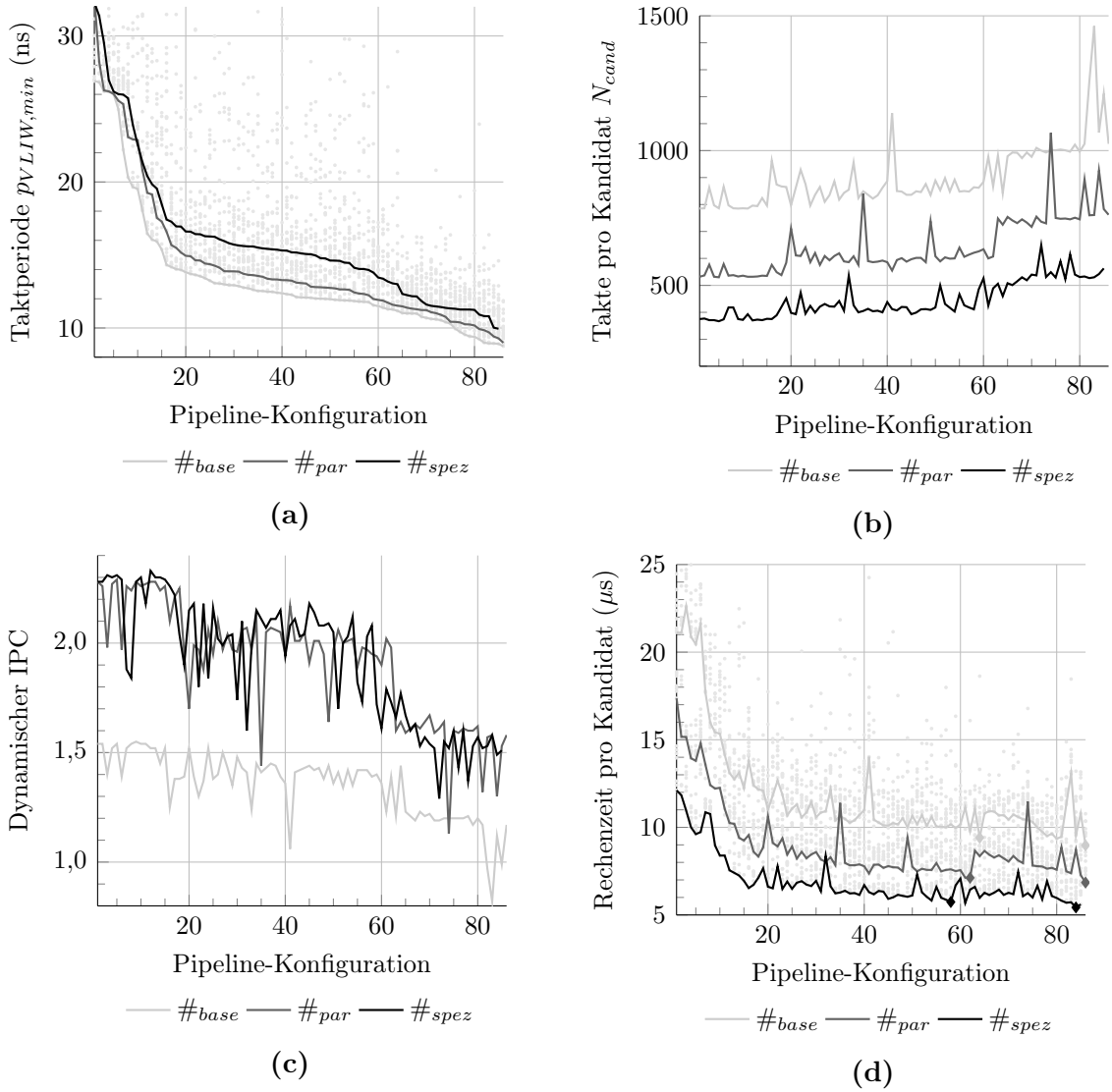



**Abbildung 5.6:** Optimierung der Prozessorpipeline

Abb. 5.6a stellt die Taktperiode in absteigender Reihenfolge sortiert über den Pipeline-Konfigurationen für die drei VLIW-Prozessorkonfigurationen dar. Die Punkte im Hintergrund stehen für je einen der zehn Bitstreams. Die Streuung der Periodenlängen aufgrund der Zufallskomponenten im Xilinx Tool Flow wird deutlich. Insgesamt liefert $\#_{base}$ die Bitstreams mit den geringsten möglichen Taktperioden, gefolgt von $\#_{par}$ und $\#_{spez}$, da die Komplexität der Architektur durch weitere FUs zunimmt, wodurch die Multiplexer wachsen und die Timing-Pfade länger werden. Die Länge der Taktperioden liegt insgesamt zwischen 33 ns und knapp 9 ns. Durch die Methodik kann die Taktfrequenz also um mehr als das Dreifache gesteigert werden.

Um den Performance-Gewinn zu beurteilen, müssen zunächst die Takte pro Merk-

malskandidat $N_{cand}$ betrachtet werden. Diese sind in Abb. 5.6b über derselben Sortierung der Pipeline-Konfigurationen aufgetragen. Insgesamt steigt $N_{cand}$ für alle VLIW-Prozessorkonfigurationen, wobei $\#_{base} > \#_{par} > \#_{spez}$ gilt. Durch Erhöhung der Parallelität und Spezialisierung lässt sich die jeweils gleiche Aufgabe in einer geringeren Anzahl an Takten lösen, wodurch der Energiebedarf insgesamt sinken wird. Die Ausreißer nach oben entsprechen denjenigen Konfigurationen, die eine AU mit einer Latenz von drei Takten besitzen. Die Penalty im Code ist dabei hoch. Die Taktperiode lässt sich durch andere Konfigurationen aber weiter absenken, was darauf hindeutet, dass die AU nicht im kritischen Pfad lag. Ab etwa $\#_{x,pipe60}$ steigt $N_{cand}$ sprunghaft. Dies liegt am Pipeline-Schalter PL_FWD, der hier besonders hervorgehoben werden soll. Da alle FUs eine zusätzliche Latenz erhalten (vgl. Abb. 3.13c), ist die Penalty hoch, jedoch lassen sich die höchstmöglichen Taktfrequenzen nur so erreichen.

Ein Maß für die Effizienz der Ausführung ist der dynamische IPC. Dieser fällt durch zunehmende Pipeline-Stufen ab (siehe Abb. 5.6c). Die Instruktionen können durch die zusätzlichen Latenzen für diese Anwendung nicht so kompakt geschedult werden (nicht ausreichend Datenparallelitäten). Insbesondere durch $PL\_FWD$ sinkt der IPC stark. Allerdings kann durch die Erhöhung der parallelen Rechenressourcen und den X2-Mode der IPC von 1,54 ($\#_{base}$) auf bis zu 2,33 ($\#_{spez}$) gesteigert werden.

Der Performance-Gewinn zeigt sich in der Rechenzeit pro Merkmalskandidat:

$$t_{VLIW,workPerCand} = \frac{N_{cand}}{f_{VLIW,max}} = N_{cand} \cdot p_{VLIW,min} \tag{5.11}$$

Nur, wenn der Zuwachs von $N_{cand}$ durch einen ebenso großen Anstieg von $f_{VLIW,max}$ ausgeglichen werden kann, ist das System in der Lage, einen höheren Durchsatz zu erzielen. Dazu zeigt Abb. 5.6d die Rechenzeit über der Pipeline-Konfiguration. Die Rechenzeit läuft in eine Sättigung und lässt sich für diese Anwendung durch weitere Pipeline-Konfigurationen kaum noch reduzieren. Die jeweils beste Konfiguration mit und ohne PL_FWD ist mit einem Punkt markiert.

Eine aussagekräftigere Darstellung ergibt sich, wenn die Rechenzeit pro Merkmalskandidat über der maximalen Taktfrequenz des Prozessors aufgetragen wird (siehe Abb. 5.7). Die kleinen Punkte im Hintergrund stehen für alle Bitstreams, die großen Punkte für die Bitstreams mit der jeweils geringsten Taktperiode jeder Pipeline-Konfiguration und die Linien stellen die jeweiligen Pareto-Fronten für die drei VLIW-Prozessorkonfigurationen dar. Manche der 86 Pipeline-Konfigurationen können die Rechenzeit trotz Frequenzsteigerung nicht verkürzen und liegen folglich nicht auf der Pareto-Front. Die Datenbasis zu den 86 Konfigurationen ist den Tab. A.4-A.9 im Anhang zu entnehmen.

Ausgehend von der Basiskonfiguration kann für die drei VLIW-Prozessorkonfigurationen die Rechenzeit pro Merkmalskandidat durch Pipeline-Optimierung um etwa 60% reduziert werden, während die Taktfrequenz um den Faktor 3× steigt. Die Rechenzeit für $\#_{par}$ ist um 20% niedriger als bei $\#_{base}$. Für $\#_{spez}$ liegt die Rechenzeit sogar etwa 40% unter der von $\#_{base}$. Der Sättigungseffekt, der unweigerlich irgendwann eintritt, da die Penalties durch weitere Pipeline-Stufen für diese Anwendung zu groß werden, als dass sie ausgeglichen

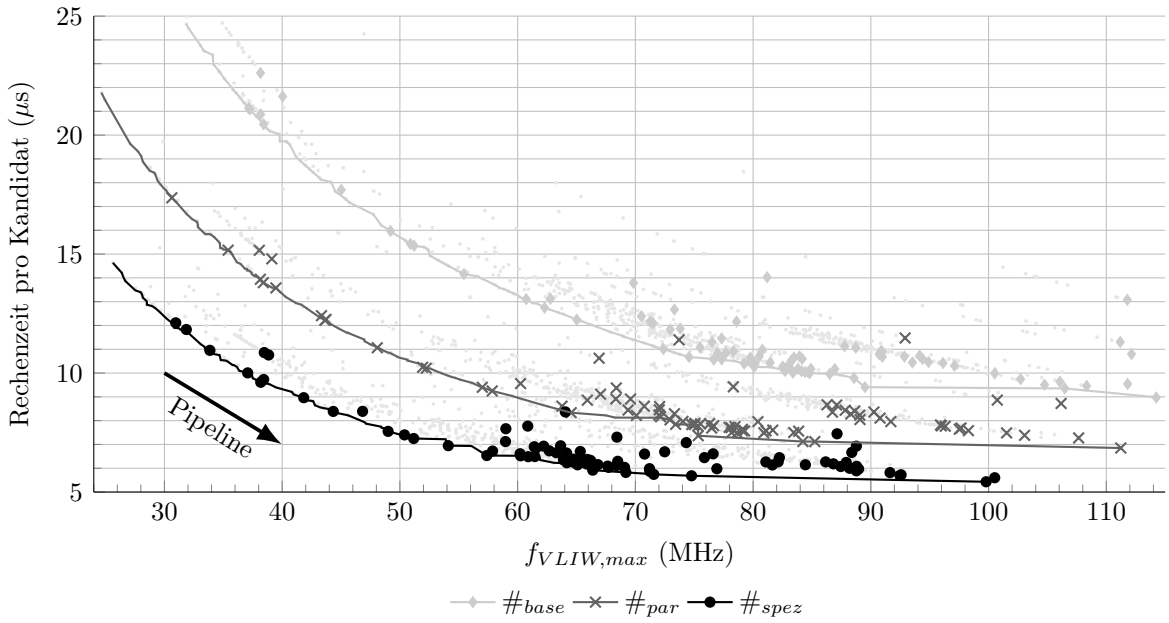



**Abbildung 5.7:** Pareto-Fronten für die Reduktion der Rechenzeit pro Merkmalskandidat bei der Pipeline-Optimierung (drei VLIW-Prozessorkonfigurationen im Vergleich)

werden könnten, beginnt in allen drei Fällen bei unterschiedlichen Taktfrequenzen. Die jeweils schnellsten Konfigurationen nutzen alle den PL_FWD-Schalter. Allerdings beträgt die Reduktion der Rechenzeit nur etwa 5% im Vergleich zur besten Konfiguration ohne PL_FWD. Zudem ist dies mit einem Zuwachs der Taktfrequenz von etwa 30% verbunden. Eine ausführlichere Übersicht, die die Basiskonfiguration und die jeweils beste Pipeline-Konfiguration ohne und mit PL_FWD für alle elf VLIW-Prozessorkonfigurationen auflistet, ist im Anhang in Tab. A.10 gegeben.

Das hier gewählte Vorgehen der empirischen, manuellen Wahl der 86 Pipeline-Konfigurationen dient als Heuristik für die Suche nach den Pareto-Fronten. Eine vollständige Suche wäre hier nicht praktikabel, da es für die Pipeline-Schalter über 300.000 verschiedene Möglichkeiten gäbe. Das Ziel bei der Untersuchung war das Erreichen des Sättigungsbereichs. Die Rechenzeit pro Kandidat wird hier methodisch lediglich als Indikator herangezogen, der auf eine Erhöhung des Durchsatzes des Gesamtsystems hindeutet. Im Detail spielen dort jedoch auch SDE-Konfiguration, tatsächliche Merkmalsanzahl im Bild und der jeweilige Betriebspunkt eine Rolle. Die Rechenzeit pro Kandidat dient jedoch der Eingrenzung des Entwurfsraums.

Tab. 5.2 vergleicht die statische Verlustleistung und die VLIW-Ressourcen zwischen Basispipeline und einer Konfiguration am Ende der Pareto-Front. Während der je zehn Synthesen pro Konfiguration sind Schwankungen der Logic Slices von bis zu 10% um den jeweiligen Mittelwert herum zu beobachten, im Mittel jedoch unter 3%. Die Anzahl LUTs und Register ist jedoch fast konstant.

**Tabelle 5.2:** Vergleich der VLIW-Ressourcen. Der VLIW-Prozessor benötigt zudem noch 16 BRAMs für das Registerfile und keine DSP-Slices.

| Architektur | $p_{VLIW,min}$ (ns) | $N_{cand}$ (Takte/ Kand.) | $P_{static,model}$ (W) | VLIW-Ressourcen | | | |
|---|---|---|---|---|---|---|---|
| | | | | Logic Slices | LUTs | Register | Register pro Logic Slice |
| $\#_{base,pipe5}$ | 26,03 | 786 | 1,92 | 11.580 | 33.480 | 10.450 | 0,9 |
| $\#_{base,pipe86}$ | 8,75 | 1025 | 1,94 | 11.607 | 34.315 | 17.040 | 1,5 |
| $\#_{par,pipe1}$ | 32,64 | 532 | 1,94 | 13.927 | 43.025 | 12.123 | 0,9 |
| $\#_{par,pipe61}$ | 11,87 | 600 | 1,96 | 12.786 | 40.747 | 15.161 | 1,2 |
| $\#_{spez,pipe3}$ | 29,53 | 371 | 1,97 | 13.800 | 44.096 | 12.040 | 0,9 |
| $\#_{spez,pipe58}$ | 13,97 | 411 | 1,99 | 13.415 | 42.880 | 14.800 | 1,1 |

$\#_{base,pipe86}$ benutzt PL_FWD. Im Virtex-6 FPGA besteht eine Logic Slice aus vier LUTs und acht Registern [Xil15]. Trotz wesentlich mehr Registern werden nur geringfügig mehr Logic Slices verwendet, da zu jeder LUT auch Register dediziert zur Verfügung stehen. Diese gesteigerte Hardwareeffizienz ist anhand des Quotienten aus Registern und Logic Slices gut zu erkennen, der von 0,9 auf 1,5 erhöht werden kann. $\#_{par,pipe61}$ und $\#_{spez,pipe58}$ benutzen kein PL_FWD, weshalb der Zuwachs an Registern im Vergleich zur Basiskonfiguration nicht so stark ist.

Es fällt auf, dass für diese beiden Konfigurationen die Anzahl an LUTs und auch die Anzahl an Logic Slices durch die Pipeline-Optimierung sogar leicht sinken. Obwohl die Unterschiede innerhalb der beobachteten Schwankungen der Synthese liegen, ist dieser Trend systematisch auch für andere VLIW-Prozessorkonfigurationen (vgl. Tab. A.10 im Anhang). Aufgrund der durch Pipelining kürzeren Timing-Pfade ist der Aufwand für die Xilinx-Tools nicht so hoch, um das vorgegebene Timing zu erreichen. Auf Reduzierung des Fanouts durch Verdopplung von Logik kann verzichtet werden, was sich im geringeren Ressourcenverbrauch zeigt. Mit PL_FWD lässt sich dieser Effekt nicht beobachten, da das Design dort durch die höhere Anzahl an Forwarding-Pfaden wieder komplexer wird.

Insgesamt benötigt $\#_{par,pipe1}$ 20% mehr Logic Slices als $\#_{base,pipe5}$. $\#_{spez,pipe3}$ ist ungefähr so groß wie $\#_{par,pipe1}$, da die zusätzlichen Spezialinstruktionen trotz geringen Ressourcenverbrauchs einen großen Einfluss auf die Performance haben (vgl. Tab. 3.4). Ein Zusammenhang zwischen den verwendeten Ressourcen und der statischen Verlustleistung zeigt sich hier nicht eindeutig.

Die Konfigurationen der hier ermittelten Pareto-Fronten für Rechenzeit und VLIW-Taktfrequenz werden nun noch hinsichtlich Durchsatz, Verlustleistung und Energie modellgestützt weiter untersucht. Dazu zeigt Abb. 5.8 die Verlustleistung während der Bildverarbeitung $P_{img}$ und die dynamische Energie in einem Bildzyklus $E_{dyn}$ über der maximalen Bildrate $R_{soll,max}$ für 1.072 Merkmalskandidaten. An den kleinen Punkten wird das System mit maximalem Takt betrieben, bei den größeren Symbolen ist ein Zielbetriebspunkt von $\rho = 0,9$ gewählt. Die jeweiligen Pareto-Fronten für $\rho = 0,9$ sind als Linien dargestellt.

Durch die Pipeline-Optimierung kann die maximale Bildrate für alle drei VLIW-

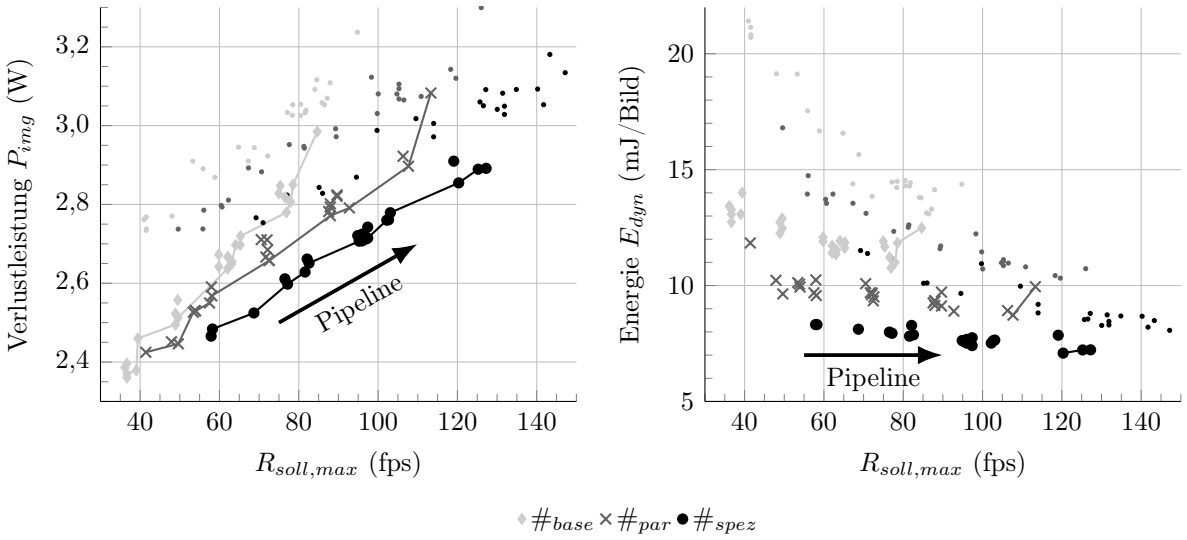



**Abbildung 5.8:** Modellbasierte Betrachtung der Verlustleistung und der Energie pro Bild für $n_c = 1.072$, kein TDM und verschiedene VLIW-Konfigurationen bei Pipeline-Optimierung. Erläuterung der Punkte siehe Text.

Prozessorkonfigurationen um einen Faktor 2-3× gesteigert werden. Durch Parallelisierung liegt die maximale Bildrate mehr als 30% über der Basiskonfiguration des Prozessors. Durch Spezialisierung liegt dieser Wert sogar bei 50%. Werden die maximalen Taktfrequenzen anstatt eines balancierten Betriebspunktes gewählt, so erhöht sich die Bildrate noch einmal um 10%. $\#_{par}$ und $\#_{spez}$ erreichen bei gleicher Verlustleistung eine höhere Bildrate als $\#_{base}$, was auf die höhere Ausführungseffizienz (höherer IPC, weniger und spezialisierte Instruktionen) und dem damit verbundenen geringeren dynamischen Energiebedarf pro Bild zurückzuführen ist.

Die jeweils oberen Pipeline-Konfigurationen der Pareto-Fronten für Verlustleistung und Bildrate sind auch bezüglich dynamischer Energie und maximaler Bildrate Pareto-optimal. Hinsichtlich der Energie ergeben sich weniger Pareto-optimale Konfigurationen, da die Energie bedingt durch weniger Ressourcen zunächst sinkt, aber gleichzeitig sogar ein höherer Durchsatz möglich ist. Die Pipeline-Optimierung reduziert den dynamischen Energiebedarf um etwa 15%. Parallelisierung reduziert die Energie um 20% und Spezialisierung benötigt noch einmal weitere 20% weniger. Die Nutzung der maximalen Taktfrequenzen erhöht den Energiebedarf um bis zu einem Drittel.

Die betrachteten Konfigurationen sind bezüglich Rechenzeit pro Merkmalskandidat und VLIW-Taktfrequenz Pareto-optimal. Die Pareto-Fronten für dynamische Energie und maximale Bildrate beinhalten für $\#_{base}$ die Pipeline-Konfigurationen 86 (kürzeste Rechenzeit mit PL_FWD), 80, 49 und 64 (kürzeste ohne PL_FWD), für $\#_{par}$ 86 (kürzeste mit PL_FWD) und 61 (kürzeste ohne PL_FWD) und für $\#_{spez}$ 54, 58 (kürzeste ohne PL_FWD) und 61. Für $\#_{spez}$ lohnt die Verwendung von PL_FWD für diese Anwendung nicht, da der Performance-Gewinn nicht hoch genug ist. Die Konfiguration 84 (kürzeste

**Tabelle 5.3:** Messergebnisse bei Prozessierung von Testsequenz 19 für verschiedene VLIW-Konfigurationen mit $\rho_{soll} = 1{,}2$ und $R_{soll} = 30$ fps (Modellfehler nach Gl. 5.9 in Klammern)

| Architektur | $p_{VLIW}$ (ns) | $p_{SDE}$ (ns) | $\overline{\rho_{ist}}$ | $\overline{R_{img}}$ (fps) | $\overline{P_{dyn,avg}}$ (mW) | $\overline{P_{dyn,max}}$ (mW) | $\overline{E_{dyn}}$ (mJ/Bild) |
|---|---|---|---|---|---|---|---|
| $\#_{base,pipe5}$ | 27 | 42 | 0,93 | 37 (-1,3%) | 467 (-6,7%) | 583 (-12%) | 15,6 (-6,7%) |
| $\#_{base,pipe86}$ | 17 | 40 | 0,80 | 41 (-0,8%) | 474 (-8,4%) | 665 (-15%) | 15,8 (-8,4%) |
| $\#_{par,pipe1}$ | 35 | 41 | 0,84 | 40 (-1,0%) | 400 (+1,8%) | 543 (-5,1%) | 13,3 (+1,8%) |
| $\#_{par,pipe61}$ | 30 | 42 | 0,80 | 39 (-0,7%) | 368 (-15%) | 503 (-18%) | 12,3 (-15%) |
| $\#_{spez,pipe3}$ | 47 | 40 | 0,80 | 41 (-1,4%) | 297 (-4,4%) | 433 (-9,2%) | 9,9 (-4,4%) |
| $\#_{spez,pipe58}$ | 43 | 41 | 0,80 | 40 (-1,5%) | 302 (-2,7%) | 425 (-6,8%) | 10,1 (-2,7%) |

mit PL_FWD) liegt bei nur 120 fps und einer höheren Energie als Konfigurationen ohne PL_FWD. Da diejenigen Pipeline-Konfigurationen mit der kürzesten Rechenzeit auch zumeist bezüglich Energie und Bildrate Pareto-optimal sind, wird bestätigt, dass die Rechenzeit hier eine gute Analysemetrik darstellt.

### Validierung

Da die bisherigen Betrachtungen auf den Modellen basieren, wird hier eine Validierung auf Basis von Testsequenz 19 für ausgewählte Konfigurationen durchgeführt.

Die Ergebnisse sind in Tab. 5.3 aufgelistet. Die Taktfrequenzen sind auf den *Worst Case* von $n_{c,max} = 1.600$ ausgelegt, um die Zielbildrate von 30 fps sicher zu erreichen. Durch die Pipeline-Optimierung werden höhere Zielbildraten überhaupt erst ermöglicht. Bei 30 fps besteht somit kein Vorteil eines höheren maximalen Prozessortakts, da die Systeme nicht im Grenzbereich der Leistungsfähigkeit betrieben werden. Für die jeweils optimierte Pipeline können $\overline{P_{dyn,avg}}$ und $\overline{E_{dyn}}$ durch Parallelisierung um 22% und durch Spezialisierung um 36% reduziert werden. Zudem sinkt die notwendige VLIW-Taktfrequenz zum Erreichen der Zielbildrate, was sich in einem geringeren Wert für $\overline{P_{dyn,max}}$ widerspiegelt.

Um den Entwurfsraum hinsichtlich des maximalen Durchsatzes aufzuzeigen, ist in Abb. 5.9 die Bildrate über der Anzahl an Merkmalskandidaten dargestellt. Die Messung ergibt sich aus der Verarbeitung der Einzelbilder der Testsequenz 19 bei maximaler Taktfrequenz der Systeme. Mittelwert und Standardabweichung entstehen, da die erreichte Bildrate für $\rho > 0{,}6$ selbst bei gleicher Merkmalsanzahl variiert. Die Messwerte wurden mit in einem über die Rohdaten laufenden Fenster mit einer Breite von 30 Kandidaten geglättet. Die Schwankungen in der Bildrate bei einer hohen bzw. geringen Anzahl an Merkmalskandidaten sind damit zu begründen, dass nur wenige Einzelbilder diese Anzahl an Merkmalskandidaten besitzen, sodass dort die Glättung einen schwächeren Effekt hat. In der modellbasierten Vorhersage ist ein Modellfehler von 5% berücksichtigt.

Durch Verwendung der maximalen Taktfrequenz stellen die Kurven eine obere Schranke für die real erreichbare Bildrate da. Der Betriebspunkt ist nicht balanciert. Wird ein energieoptimiertes System angestrebt, reduziert sich die erreichbare Bildrate (vgl. Abb. 4.9).

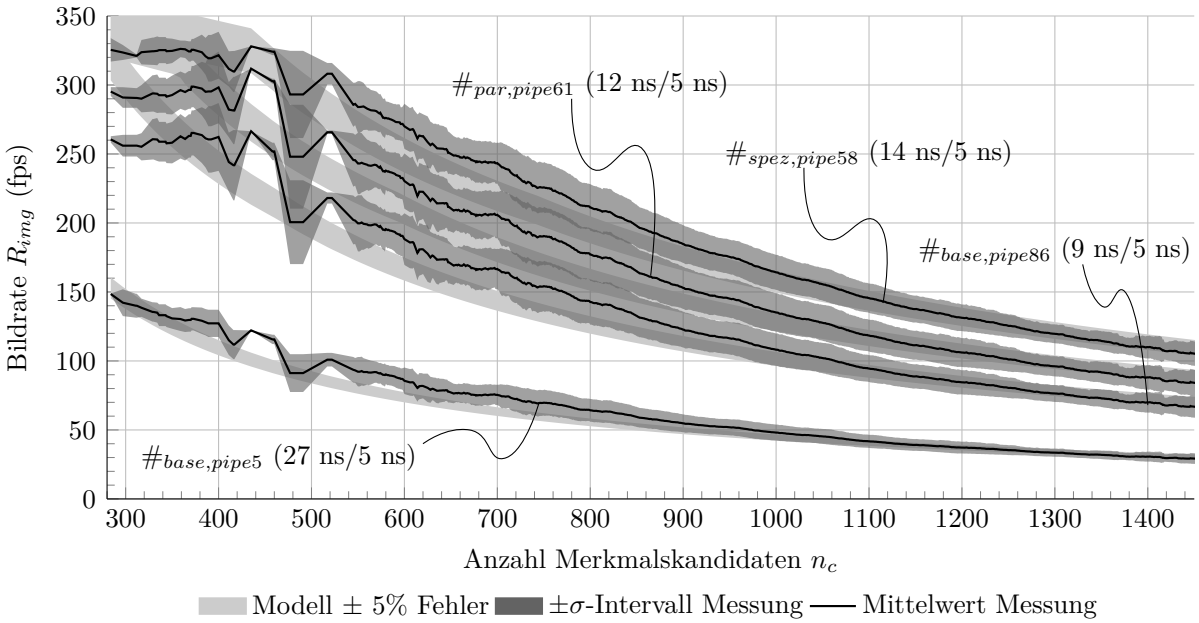



**Abbildung 5.9:** Messung der Bildrate für verschiedene Prozessorkonfigurationen bei Verarbeitung der Einzelbilder aus Testsequenz 19 über der in den Bildern gefundenen Anzahl an Merkmalskandidaten. Das System verwendet maximale Taktfrequenzen (Taktperioden in Klammern für VLIW/SDE), sodass die Bildrate eine in der Praxis erreichbare obere Schranke darstellt.

Insgesamt hängt die Bildrate maßgeblich von der Anzahl an zu verarbeitenden Merkmalen ab, welche zum einen datenabhängig ist, zum anderen aber auch durch algorithmische Parameter beeinflusst wird. Bereits die Optimierung der Pipeline des VLIW-Prozessors in Basiskonfiguration bewirkt einen starken Anstieg der erreichbaren Bildrate, welche durch Parallelisierung und Spezialisierung jeweils noch weiter erhöht werden kann.

### 5.2.3 Optimierung der SDE-Architektur

Der wesentliche Architekturparameter, der hier für den SDE untersucht wird, ist der Grad des Time-Division-Multiplexings in der Filterhardware der Gauß-Pyramide (TDM-Modus). Daraus ergibt sich für den SDE ein globaler Takt und ein Takt nur für die DSP-Slices, auf denen die Filter implementiert sind. Die DSP-Slices können für den hier verwendeten FPGA mit 450 MHz getaktet werden [Xil11a]. In dieser Arbeit wird als maximaler Takt 400 MHz ($p_{SDE,Filter} = 2,5$ ns) genutzt.

Die in der späteren Validierung untersuchte VLIW-Prozessorkonfiguration ist $\#_{spez,pipe54}$. Der VLIW-Prozessor kann in dieser Konfiguration mit maximal 67 MHz ($p_{VLIW,min} = 15$ ns) getaktet werden. Zur Einstellung der verschiedenen Taktfrequenzen im MMCM müssen nach Gl. 3.20 bestimmte Konstanten hinterlegt werden, die den Eingangstakt ins Gesamtsystem von $f_{clkin} = 200$ MHz entsprechend teilen. Dazu zeigt

**Tabelle 5.4:** Einzustellende Konstanten im MMCM für $f_{clkin} = 200$ MHz und verschiedene TDM-Modi (vgl. Gl. 3.20)

| TDM-Modus | $p_{VLIW}$ (ns) | $p_{SDE}$ (ns) | $p_{SDE,Filter}$ (ns) | $M$ | $D$ | $O_{VLIW}$ | $O_{SDE}$ | $O_{SDE,Filter}$ |
|---|---|---|---|---|---|---|---|---|
| - | 15 | 5 | 5 | 5 | 1 | 15 | 5 | 5 |
| ×2 | 15 | 5 | 2,5 | 6 | 1 | 18 | 6 | 3 |
| ×3 | 15 | 7,5 | 2,5 | 6 | 1 | 18 | 9 | 3 |
| ×6 | 15 | 15 | 2,5 | 6 | 1 | 18 | 18 | 3 |
| - | 22 | 20 | 20 | 5 | 1 | 22 | 20 | 20 |
| ×2 | 22 | 20 | 10 | 5 | 1 | 22 | 20 | 10 |
| ×3 | 22 | 20 | 6,7 | 6 | 1 | 27 | 24 | 8 |
| ×6 | 22 | 20 | 3,3 | 6 | 1 | 27 | 24 | 4 |

**Tabelle 5.5:** Vergleich der SDE-Ressourcen. Der SDE benötigt zudem noch 116 BRAMs u. a. für Zeilenspeicher.

| TDM-Modus | $p_{SDE,min}$ (ns) | $P_{static,model}$ (W) | SDE-Ressourcen | | | | |
|---|---|---|---|---|---|---|---|
| | | | Logic Slices | LUTs | Register | LUTRAM | DSP-Slices |
| - | 5 | 1,99 | 3.310 | 10.079 | 7.749 | 1.525 | 156 |
| ×2 | 5 | 1,97 | 3.260 | 10.064 | 7.858 | 1.499 | 86 (-45%) |
| ×3 | 7,5 | 1,98 | 3.213 | 9.963 | 7.536 | 1.651 | 64 (-59%) |
| ×6 | 15 | 1,97 | 3.222 | 9.904 | 7.276 | 1.412 | 42 (-73%) |

Tab. 5.4 zwei Beispiele. Zum einen soll das System mit den maximalen Taktfrequenzen betrieben werden ($p_{VLIW} = 15$ ns, $p_{SDE,Filter} = 2{,}5$ ns), zum anderen soll ein aus der Modellierung heraus ermittelter Betriebspunkt für $R_{soll} = 60$ fps und $\rho_{soll} = 1{,}2$ bei $n_{c,max} = 1.600$ eingestellt werden ($p_{VLIW} = 22$ ns, $p_{SDE} = 20$ ns). Das Verhältnis von $O_{SDE}$ zu $O_{SDE,Filter}$ entspricht dem jeweiligen TDM-Modus.

Durch die Verwendung des TDM-Modus reduziert sich die maximale Frequenz, mit der der SDE arbeitet. Diese beträgt für TDM×6 nur noch 67 MHz, um die 400 MHz in den DSP-Slices nicht zu überschreiten. Somit reduziert sich auch der maximal mögliche Durchsatz des Systems, wie sich weiter unten zeigt. Zunächst zeigt Tab. 5.5 die SDE-Ressourcen bei verschiedenen TDM-Modi. Die Logik-Ressourcen sind für alle Varianten nahezu identisch und entsprechen einem Viertel der Größe des VLIW-Prozessors. Die Anzahl an DSP-Slices kann von 156 auf bis zu 42 reduziert werden, was einer Verringerung von bis zu 73% entspricht. Die Einsparung ist nicht proportional zum TDM-Modus, da immer der größere von zwei Filtern erhalten bleiben muss (vgl. Abschnitt 3.4.1 ab Seite 49). Eine veränderte statische Verlustleistung ist nicht zu erkennen, da selbst bei Nichtverwendung bestimmter DSP-Slices die Logik- und Routing-Strukturen in den betroffenen FPGA-Bereichen genutzt werden und somit die Spannungsversorgung dort aktiviert ist.

Abb. 5.10 zeigt die modellbasierte Betrachtung der Verlustleistung während der Bildverarbeitung $P_{img}$ und der dynamischen Energie in einem Bildzyklus $E_{dyn}$ über der maximalen

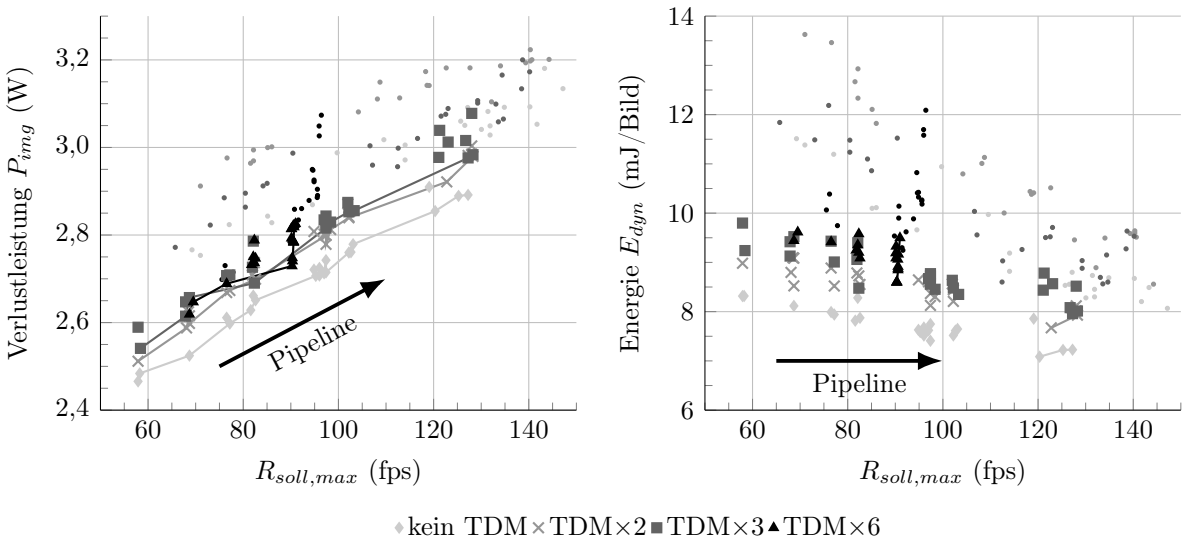



**Abbildung 5.10:** Modellbasierte Betrachtung der Verlustleistung und der Energie pro Bild für $n_c = 1.072$, $\#_{spez}$ und verschiedene TDM-Modi bei Pipeline-Optimierung. Erläuterung der Punkte siehe Text.

Bildrate $R_{soll,max}$ für 1.072 Merkmalskandidaten und einer VLIW-Prozessorkonfiguration von $\#_{spez}$ bei Pipeline-Optimierung. An den kleinen Punkten wird das System mit maximalem Takt betrieben, bei den größeren Symbolen ist ein Zielbetriebspunkt von $\rho = 0{,}9$ gewählt. Die jeweiligen Pareto-Fronten für $\rho = 0{,}9$ sind als Linien dargestellt.

Zwischen keinem TDM, TDM×2 und TDM×3 ist für $\rho = 0,9$ kein Unterschied im Durchsatz zu erkennen. Alle drei Modi können etwa die gleiche maximale Bildrate erreichen, da der Bottleneck hier im VLIW-Prozessor liegt. Für TDM×6 allerdings zahlt sich eine weitere Pipeline-Optimierung ab einer Bildrate von etwa 90 fps nicht mehr aus, da das System dort durch den SDE und die bei TDM×6 geringere maximale Taktrate limitiert ist. Der VLIW-Prozessor würde eine höhere Bildrate zulassen. Die maximale Bildrate ist in diesem exemplarischen Fall für $n_c = 1.072$ und TDM×6 im Vergleich zu den anderen TDM-Modi fast 30% geringer.

TDM×2, TDM×3 und TDM×6 erzeugen bei gleicher maximaler Bildrate eine etwas höhere Verlustleistung als ohne TDM. Durch kontinuierliches, zyklisches Durchtauschen der Filterkoeffizienten findet in diesen Bereichen der DSP-Slices eine höhere Schaltaktivität statt, welche ohne TDM nicht notwendig ist. Zudem ergeben sich zusätzliche Schieberegisterstrukturen zur Realisierung des Multiplexings rund um die DSP-Slices. Als Resultat ergibt sich ein um etwa 10% erhöhter dynamischer Energiebedarf pro Bild für TDM×2 und TDM×3. TDM×6 benötigt sogar knapp 20% mehr dynamische Energie pro Bild, obwohl die eigentlich durchgeführten Bildfilterungen in allen TDM-Modi im Kern dieselben Multiplikationen und Additionen beinhalten.

**Tabelle 5.6:** Messergebnisse bei Prozessierung von Testsequenz 19 bei verschiedenen TDM-Modi mit $\#_{spez,pipe54}$, $\rho_{soll} = 1{,}2$ und $R_{soll} = 60$ fps (Modellfehler nach Gl. 5.9)

| TDM-Modus | $p_{VLIW}$ (ns) | $p_{SDE}$ (ns) | $\overline{\rho_{real}}$ | $\overline{R_{img}}$ (fps) | $\overline{P_{dyn,avg}}$ (mW) | $\overline{P_{dyn,max}}$ (mW) | $\overline{E_{dyn}}$ (mJ/Bild) |
|---|---|---|---|---|---|---|---|
| - | 22 | 20 | 0,82 | 82 (-1,4%) | 592 (-10%) | 827 (-23%) | 9,9 (-10%) |
| ×2 | 22 | 20 | 0,82 | 82 (-1,4%) | 654 (-9,2%) | 907 (-21%) | 10,9 (-9,2%) |
| ×3 | 22 | 20 | 0,84 | 81 (-0,6%) | 656 (-10%) | 908 (-21%) | 10,9 (-10%) |
| ×6 | 22 | 20 | 0,84 | 81 (-0,6%) | 737 (-9,4%) | 1.018 (-19%) | 12,3 (-9,4%) |

### Validierung

Die Validierung der modellbasierten Ergebnisse wird hier für $R_{soll} = 60$ fps und $\rho_{soll} = 1{,}2$ bei $n_{c,max} = 1.600$ durchgeführt. Die Messergebnisse sind in Tab. 5.6 aufgelistet.

Wie aus der Modellierung zu erwarten, liefern TDM×2 und TDM×3 nahezu identische Ergebnisse bezüglich der dynamischen Verlustleistung und Energie. Diese liegen im Vergleich zu keinem TDM auch in der realen Messung um 10% höher. Für TDM×6 ergibt sich hier sogar ein Anstieg um knapp 25%. Abb. 5.11 stellt die angedeutete Reduktion der maximalen Bildrate bei TDM×3 und TDM×6 dar, wenn maximale Taktfrequenzen verwendet werden. TDM×2 kann mit den gleichen Frequenzen wie ohne TDM betrieben werden. Die Reduktion entsteht im Bereich weniger Merkmalskandidaten, in denen das System durch den SDE und nicht durch den VLIW-Prozessor limitiert ist. Für TDM×3 reduziert sich die Bildrate um 30%, für TDM×6 sogar um über 60%, wobei die maximale Bildrate allerdings nur für wenige Merkmalskandidaten überhaupt erreicht werden kann. Insgesamt steht bei Verwendung von TDM×6 dem um 20% größeren Energiebedarf, der damit verbundenen erhöhten Verlustleistung bei gleicher Bildrate und der um über 60% geringeren maximalen Bildrate eine Reduktion der DSP-Slices von 73% gegenüber.

## 5.3 Dynamische Anpassung des Betriebspunktes

Die statische Architekturoptimierung liefert durch modellgestützte Exploration des Entwurfsraums zur Designzeit eine Reihe von Systemkonfigurationen. Diese können zum einen einen vorgegebenen maximalen Durchsatz bei einer vorgegebenen Anzahl an maximal zu verarbeitenden Merkmalskandidaten erreichen und sind zum anderen hinsichtlich Verlustleistung und Energie Pareto-optimal. Dabei ist es nicht optimal, immer mit *maximalen Taktfrequenzen* zu arbeiten. Es ergibt sich stattdessen ein optimaler Zielbetriebspunkt des Systems, woraus sich die Taktfrequenzen für den VLIW-Prozessor und den SDE ableiten. Zur Designzeit festgelegte Designparameter müssen sich bezogen auf die Systemperformance immer am *Worst Case* orientieren.

Die Anzahl an Merkmalskandidaten ist jedoch abhängig von den Eingangsdaten und schwanken in einer realen Sequenz. Testsequenz 19 besitzt im Mittel 1.052 Kandidaten mit einer Standardabweichung von 178. Damit ergibt sich laufend eine veränderte

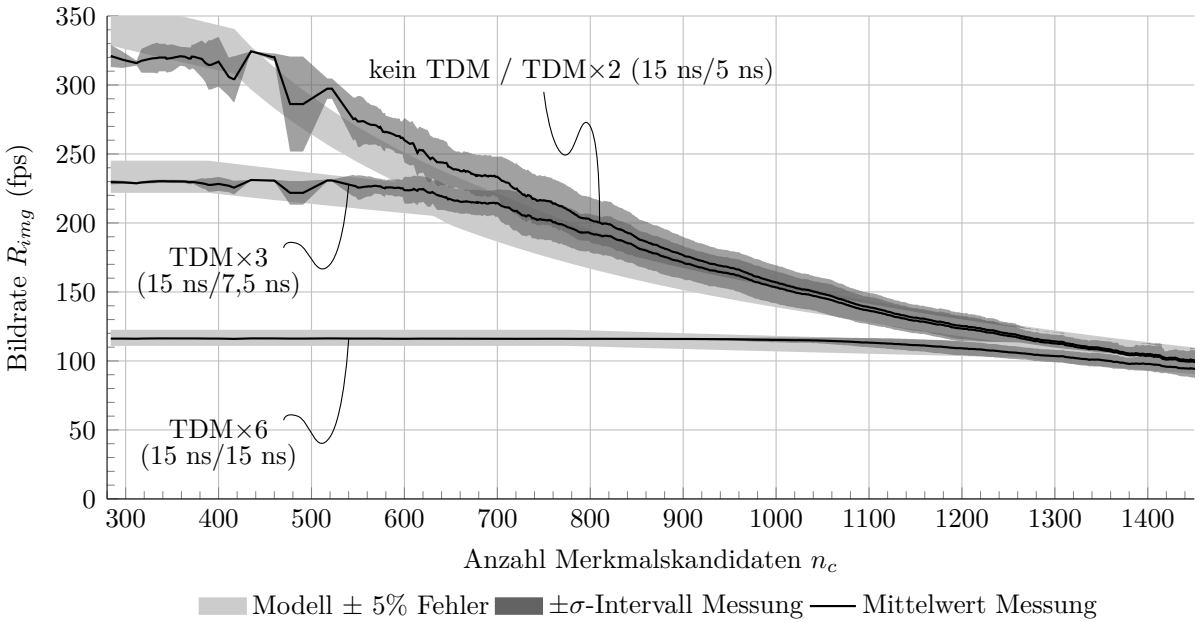



**Abbildung 5.11:** Messung der Bildrate für verschiedene TDM-Modi bei Verarbeitung der Einzelbilder aus Testsequenz 19 über der im Bild gefundenen Anzahl an Merkmalskandidaten. Das System verwendet maximale Taktfrequenzen (Taktperioden in Klammern für VLIW/SDE), sodass die Bildrate eine in der Praxis erreichbare obere Schranke darstellt.

Rechenlast für das System. Im Sinne einer Optimierung wäre es wünschenswert, das System nicht am Worst Case, sondern an tatsächlichen vorherrschenden Bedingungen zu parametrieren. Um während der Verarbeitung einen anvisierten Zielbetriebspunkt mit einer garantierten *konstanten Mindestbildrate* $R_{soll,min}$ beizubehalten, müssen die Taktfrequenzen des Systems zur Laufzeit auf die aktuelle Merkmalsanzahl angepasst werden.

Darüber hinaus ergibt sich im speziellen Anwendungsfall hier ein weiterer dynamischer Eingangsparameter. Die Verarbeitung der Bildsequenz geschieht während der Fahrt eines Autos, sodass sich neben dem Bildinhalt auch die Fahrgeschwindigkeit ändert. Diese kann durch eine der Merkmalsextraktion nachgeschaltete Eigenbewegungsschätzung bildbasiert ermittelt werden. Es ergibt sich ein Abstand zwischen zwei Einzelbildern zu

$$d_{img} = \frac{v_{Auto}}{R_{img}}. \tag{5.12}$$

Fährt das Auto schneller, so ist für den gleichen Abstand zwischen zwei Bildern auch eine höhere Bildrate notwendig. Im Sinne der Fahrsicherheit ist es sinnvoll, anstatt einer konstanten Mindestbildrate einen *konstanten Maximalbildabstand* $d_{soll,max}$ zu garantieren und diesen als Zielindikator zu definieren. Daraus ergibt sich eine zur Fahrgeschwindigkeit proportionale Zielbildrate. Es ist zu berücksichtigen, dass hier Stereobilder, also zwei Bilder zu einem Zeitpunkt, verarbeitet werden. Somit muss die tatsächliche Bildrate

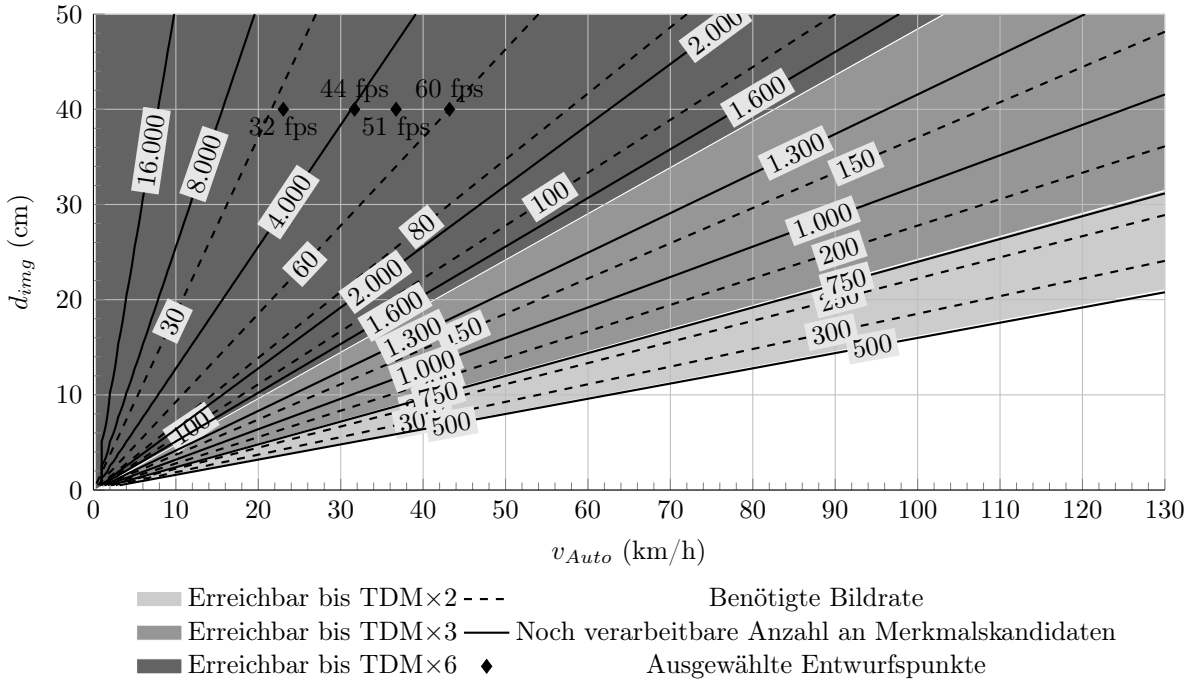



**Abbildung 5.12:** Entwurfsraum zum Bildabstand für VLIW-Prozessorkonfiguration $\#_{spez,pipe58}$ und maximale Taktfrequenzen. Die Schilder an den Geraden geben die benötigte Bildrate, sowie die gerade noch verarbeitbare Anzahl an Merkmalskandidaten an.

doppelt so hoch liegen als für den Abstand nach Gl. 5.12 nötig.

Der Entwurfsraum, der sich für diese Abstandsbetrachtung bei Verwendung von maximalen Taktfrequenzen ergibt, ist in Abb. 5.12 dargestellt. Bei verschiedenen Geschwindigkeiten $v_{Auto}$ sind unterschiedliche Bildraten notwendig, um einen bestimmten Abstand $d_{img}$ zu erreichen. Dabei lassen sich für die TDM-Modi aufgrund der Limitierung im maximalen Durchsatz jeweils unterschiedliche Regionen erreichen. Die Anzahl an Merkmalskandidaten ergibt sich vereinfacht, indem die zur Verfügung stehende Zeit $t_{img}$ durch die Zeit pro Merkmalskandidat, welche hier 14 ns/Takt · 411 Takte/Kand.= 5,76 $\mu$s/Kand. beträgt, geteilt wird. Durch Einhalten eines bestimmten Betriebspunktes reduziert sich die Anzahl an Merkmalskandidaten, was über das Modell bestimmt werden kann.

Das Potential einer dynamischen Anpassung der Taktfrequenzen wird im Folgenden evaluiert. Als Mindestbildrate wird $R_{soll,min} = 60$ fps festgelegt. Der Maximalbildabstand soll $d_{soll,max} = 40$ cm betragen. Die verwendete VLIW-Prozessorkonfiguration ist $\#_{spez,pipe58}$ mit einem Zielbetriebspunkt von $\rho_{soll} = 1{,}2$. Es wird von einer maximalen Anzahl an Merkmalskandidaten von $n_{c,max} = 1.600$ ausgegangen. Bei einer maximalen Geschwindigkeit in dieser Sequenz von $v_{Auto} = 43{,}2$ km/h ist der Maximalbildabstand bei einer Bildrate von 60 fps immer unterschritten. Abb. A.19 im Anhang gibt Informationen

zu Merkmalsanzahl und Fahrgeschwindigkeit der Testsequenz 19.

Für die dynamische Anpassung der Taktfrequenzen des Systems ergeben sich vier Szenarien für die VLIW- und SDE-Taktfrequenzen:

- *max*: Das System verwendet immer maximale Taktfrequenzen und übertrifft die Mindestbildrate bzw. unterschreitet den Maximalbildabstand um ein Vielfaches.

- $worstcase_{60fps}$: Das System wird mithilfe des Laufzeitmodells statisch für einen Worst Case an Merkmalskandidaten und Fahrgeschwindigkeit parametriert und ändert seine Taktfrequenzen zur Laufzeit nicht. Der Zielbetriebspunkt wird somit nur im Worst Case erreicht und die Mindestbildrate außer im Worst Case immer deutlich überschritten.

- $dyn_{cand,60fps}$: Das System passt seine Taktfrequenzen dynamisch in jedem Bild an die zu erwartende Anzahl an Merkmalskandidaten an. Dazu wird hier anhand der Anzahl in den letzten vier Bildern eine Prädiktion für das kommende Bild durchgeführt, indem eine Ausgleichsgerade errechnet und zur Extrapolation genutzt wird. Der so erhaltene Schätzwert wird anschließend noch um einen Offset von 200 erhöht und aus Implementierungsgründen in 31 Bins quantisiert. Anhand des Laufzeitmodells können für den quantisierten Schätzwert und die Mindestbildrate optimale Taktfrequenzen ausgewählt werden (siehe Tab. A.11 im Anhang).

- $dyn_{velo,cand,40cm}$: Zusätzlich zur Prädiktion der Merkmalskandidaten wird die Fahrgeschwindigkeit genutzt, um einen Maximalbildabstand einzuhalten. Die Merkmalskandidaten werden aus Implementierungsgründen in sieben, die Fahrgeschwindigkeit in vier Bins quantisiert (siehe Tab. A.12 im Anhang). Die vier Bins entsprechen den ausgewählten Betriebspunkten in Abb. 5.12.

Die erreichte Bildrate der vier Szenarien bei der Verarbeitung von Testsequenz 19 ist in Abb. 5.13 für die ersten 2.000 Bilder dargestellt.

Für die ersten drei Szenarien gilt $R_{soll,min} = 60$ fps. Szenario $worstcase_{60fps}$ liegt dabei deutlich dichter an 60 fps als *max*, bleibt aber immer darüber, da die 60 fps nur bei 1.600 Merkmalskandidaten erreicht würden. Für $dyn_{cand,60fps}$ lässt sich eine leichte Verbesserung erkennen, da die Prädiktionsmethodik eher konservativ ist und die tatsächliche Anzahl an Merkmalskandidaten meist deutlich überschätzt wird (im Mittel um knapp 20%). Allerdings hat sich bei einem weniger konservativen Schätzansatz gezeigt, dass aufgrund plötzlicher Schwankungen der Kandidatenanzahl von Bild zu Bild die 60 fps oft nicht erreicht werden. Die Zielbildrate $R_{img,soll,40cm}$ für das vierte Szenario ist abhängig von der Fahrgeschwindigkeit, welche durch die Quantisierung um gut 10% überschätzt wird. Die gemessene Bildrate nähert sich dem sich ergebenden Verlauf merkbar an.

Die Messergebnisse für Verlustleistung und Energie sind in Tab. 5.7 zu sehen. Die mittlere Bildrate $\overline{R_{img}}$ bei $dyn_{cand,60fps}$ liegt etwas dichter an der Zielvorgabe als bei $worstcase_{60fps}$. Dadurch kann insbesondere die maximale, dynamische Verlustleistung

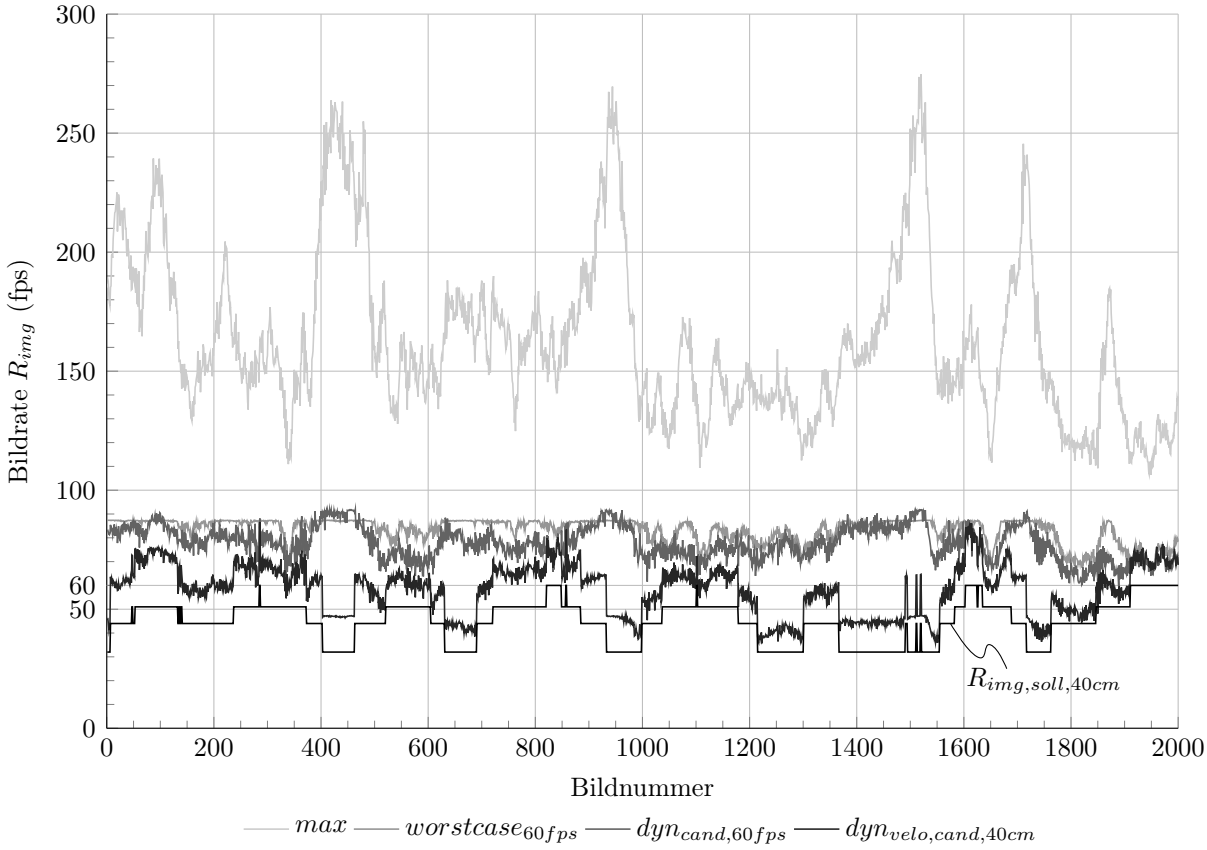



**Abbildung 5.13:** Gemessene Bildrate für die vier Szenarien für die VLIW- und SDE-Taktfrequenzen

$\overline{P_{dyn,max}}$ um etwa 8% reduziert werden. Die Reduktion gegenüber $max$ beträgt hier 50%. $\overline{P_{dyn,avg}}$ ist für $dyn_{cand,60fps}$ etwa 12% geringer als für $max$.

Im vierten Szenario, das anstatt auf eine konstante Mindestbildrate auf einen konstanten maximalen Bildabstand abzielt, reduzieren sich $\overline{P_{dyn,avg}}$ und $\overline{P_{dyn,max}}$ um 20% gegenüber $dyn_{cand,60fps}$, da hier eine kleinere mittlere Bildrate und geringere Taktfrequenzen möglich sind. Der Abstand $\overline{d_{img}}$ liegt durch die Anpassung der Verarbeitungsgeschwindigkeit auf Basis der Fahrgeschwindigkeit deutlich dichter an den vorgegebenen 40 cm.

Die Energie $\overline{E_{dyn}}$ ist für die drei modellgestützten Szenarien etwa gleich, da die durchgeführten Berechnungen identisch sind und der leicht unterschiedliche mittlere Betriebspunkt $\overline{\rho_{real}}$ hier keine sichtbare Auswirkung hat. Im Vergleich zu $max$, welches ein deutlich schlechter balanciertes System darstellt, reduziert sich der Energiebedarf um 11%.

Die vorgestellte dynamische Bild-zu-Bild-Anpassung der Taktfrequenzen kann die dynamische Verlustleistung des Systems unter Einhaltung der Zielindikatoren signifikant reduzieren. Insgesamt sind im gewählten Beispiel die realen Taktfrequenzen mit maximal

**Tabelle 5.7:** Messergebnisse bei Prozessierung von Testsequenz 19 bei verschiedenen Szenarien für die VLIW- und SDE-Taktfrequenzen ($\#_{spez,pipe58}$, $\rho_{soll} = 1{,}2$) und ohne TDM

| Szenario | $p_{VLIW}$ (ns) | $p_{SDE}$ (ns) | $\overline{\rho_{real}}$ | $\overline{R_{img}}$ (fps) | $\overline{d_{img}}$ (cm) | $\overline{P_{dyn,avg}}$ (mW) | $\overline{P_{dyn,max}}$ (mW) | $\overline{E_{dyn}}$ (mJ/Bild) |
|---|---|---|---|---|---|---|---|---|
| $max$ | 14 | 5 | 2,12 | 160 | 11 | 647 | 1507 | 10,8 |
| $worstcase_{60fps}$ | 21 | 20 | 0,80 | 83 | 20 | 575 | 814 | 9,6 |
| $dyn_{cand,60fps}$ | siehe Tab. A.11 | | 1,02 | 76 | 22 | 568 | 746 | 9,5 |
| $dyn_{velo,cand,40cm}$ | siehe Tab. A.12 | | 0,99 | 60 | 35 | 452 | 597 | 9,6 |

48 MHz (VLIW) und 59 MHz (SDE) eher niedrig. Innerhalb der dynamischen Anpassung werden die Taktfrequenzen um einen Faktor bis zu 3× (VLIW) und 2× (SDE) reduziert. Die VLIW-Prozessorkonfiguration im Beispiel könnte mit bis 71 MHz betrieben werden, was eine Erhöhung der Bildrate über 60 fps, eine Verarbeitung von mehr Kandidaten oder eine Erhöhung der Komplexität des Algorithmus ermöglichen würde.

Durch genauere Prädiktionsmechanismen für die Anzahl an Merkmalskandidaten im nächsten Bild und durch eine feinere Quantisierung mittels einer höheren Anzahl an Bins könnten die Einsparungen noch größer werden, da dadurch noch genauer an die Zielvorgaben (Mindestbildrate, Maximalbildabstand) angenähert würden.

## 5.4 Literaturvergleich

Eine ausführliche Übersicht über SIFT-Implementierungen für FPGA und ASIC in der Literatur wurde in Abschnitt 3.1.2 präsentiert. In diesem Abschnitt wird die in Rahmen dieser Arbeit entworfene Architektur mit den anderen Implementierungen quantitativ verglichen. Für einen fairen Vergleich muss berücksichtigt werden, dass sich die Varianten in Technologie, Bildauflösung und insbesondere in ihrer genauen Implementierung des Algorithmus stark unterscheiden. Das Ziel ist zumeist, dass die reale Anwendung, die der Bildmerkmalsextraktion nachgeschaltet ist, mit der Implementierung des Algorithmus gut funktioniert. Eine exakte Implementierung des Referenzalgorithmus steht nicht im Vordergrund. Die Implementierung der Subpixel-Lokalisierung in dieser Arbeit stellt einen wesentlichen Unterschied zur Literatur dar, da dieser algorithmische Schritt meist nicht realisiert wird. Zudem ist der Deskriptor zur günstigeren Implementierung auf dem VLIW-Prozessor hier weniger komplex als in den meisten Arbeiten.

### Detektionszeit

In Abschnitt 3.1.3 wurde eine Normierung der Detektionszeit auf VGA-Auflösung und eine Oktave vorgestellt, um die Vergleichbarkeit zwischen verschiedenen Implementierungen zu erhöhen (vgl. Tab. 3.2). Abb. 5.14 zeigt diese normierte Detektionszeit, d. h. die reine Verarbeitungszeit eines Eingangsbildes, mit der Normierung nach Gl. 3.11 für die

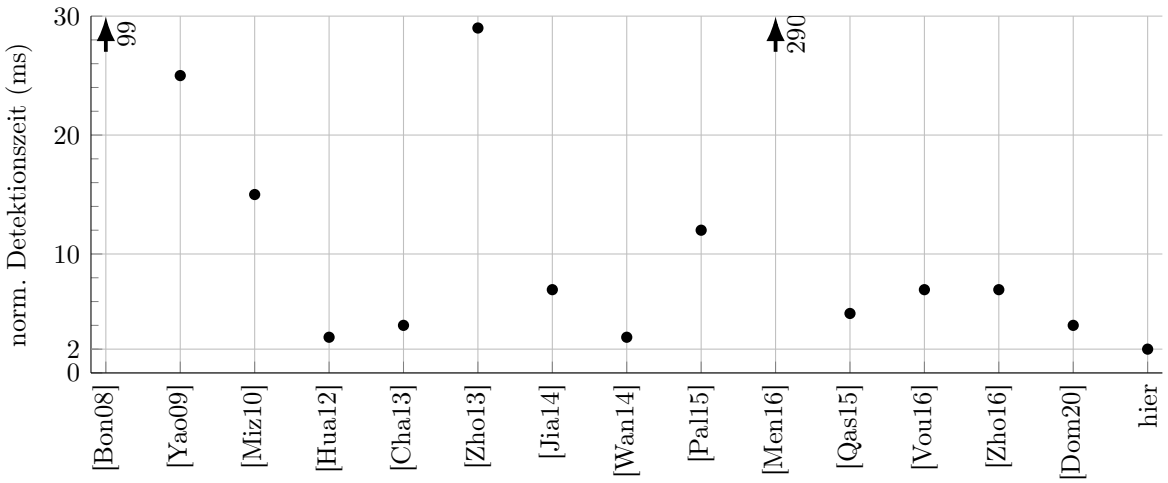



**Abbildung 5.14:** Vergleich der normierten Detektionszeiten nach Abschnitt 3.1.3

Literatur und die im Rahmen dieser Arbeit entworfene Architektur. Dabei liegt hier eine Verarbeitungsdauer von 2,91 ms (ohne TDM) für 1024×368 px und mehr als zwei Oktaven zugrunde. Im Vergleich ist die Architektur hier mit 2 ms etwas schneller als alle anderen Implementierungen, was an der starken Parallelisierung der Gauß-Pyramide und der hohen maximalen SDE-Taktfrequenz liegt.

### Bildrate

Die Bildrate ist für die Anwendung der Merkmalsextraktion immer nur im Vergleich mit der Anzahl an Merkmalen zu beurteilen, die dabei schritthaltend verarbeitet werden können. Dazu ist in Abb. 5.15 die *normierte Bildrate* nach Gl. 3.13 über der Anzahl an *Merkmalen pro Pixel*[1] dargestellt. Auf einem Bild mit hoher Auflösung werden mehr Merkmale gefunden, für die auch die Rechenleistung zur Deskriptorberechnung bereitstehen muss. Hierbei zeigen sich deutliche Unterschiede für die in der Literatur erschienenen Ansätze.

Konfiguration $\#_{spez,pipe58}$ (kein TDM) der in dieser Arbeit vorgestellten FPGA-Architektur benötigt bei maximaler Taktfrequenz 5,76 $\mu$s pro Merkmalskandidat zur Berechnung der Subpixel-Lokalisierung, der Stabilitätsprüfung und des Deskriptors. Dabei liegt die in Abschnitt 4.2.2 beschriebene Statistik zugrunde, nach der eine mittlere Anzahl an Takten $N_{cand}$ ermittelt wird. Die mittlere Anzahl an Takten liegt hier bei 411 Takten pro Merkmalskandidaten. Die Abb. 5.15 zeigt zum einen die Messung der normierten Bildrate für Testsequenz 19. Der kurze Anstieg bei etwa 0,4 % Merkmale pro Pixel ist darin begründet, dass nur wenige Einzelbilder der Testsequenz eine derart hohe Anzahl an Merkmalen aufweisen und Schwankungen in der Bildrate nicht durch

[1] basierend auf den in der jeweiligen Publikation angegebenen Daten für die Anzahl an Merkmalen und die Bildauflösung

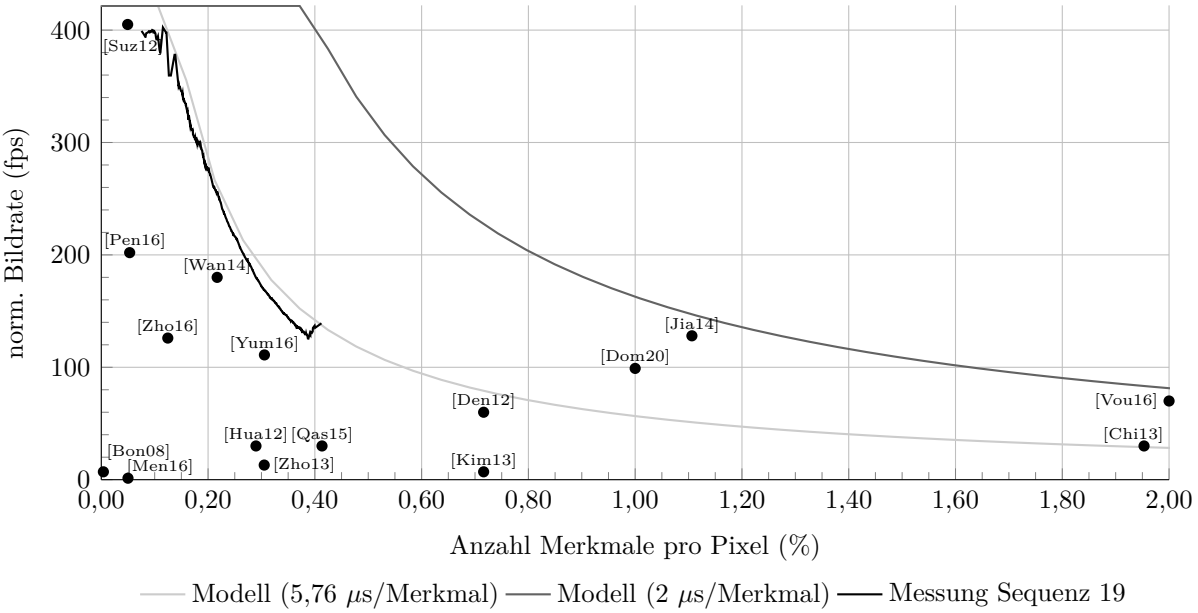



**Abbildung 5.15:** Normierte Bildrate über der normierten Merkmalsanzahl

Mittelwertbildung ausgeglichen werden können. Die Abbildung zeigt zum anderen noch ein vereinfachtes Modell nach folgender Gleichung zur Hochrechnung auf eine höhere Anzahl an Merkmalen, basierend auf der Architekturmodellierung für Konfiguration $\#_{spez,pipe58}$ (kein TDM):

$$R_{img,model,simple,norm}(n_c) = \frac{1}{\max(2{,}91\ \text{ms}; 5{,}76\ \mu s \cdot n_c)} \frac{1024 \cdot 368}{640 \cdot 480} \tag{5.13}$$

mit $n_c$ der Anzahl an Merkmalen. Diese einfache Modellbildung wird durch die in der Abbildung gezeigte Messreihe auf Sequenz 19 bestätigt. Für die hier entworfene Architektur zeigt sich zudem ein sehr gutes Vergleichsergebnis gegenüber den meisten Ansätzen in der Literatur. [Dom20] nehmen durch Puffern eines ganzen Bildes eine hohe Latenz und Speicherbedarf in Kauf. [Jia14] berechnet einen eigenen Deskriptor, der weit von der SIFT-Referenz abweicht. In [Vou16] wird für jedes Pixel auch ein Deskriptor berechnet, wodurch kein höherer Durchsatz als die angegebenen 70 fps möglich ist, unabhängig von der tatsächlichen Merkmalsanzahl.

Zum Vergleich ist außerdem die Modellrechnung nach Gl. 5.13 für 2 $\mu$s (anstatt 5,76 $\mu$s) pro Merkmalskandidaten im VLIW-Prozessor dargestellt, um aufzuzeigen, in welche Richtung die Architektur für einen noch höheren Durchsatz optimiert werden müsste: eine schnellere Abarbeitung im VLIW-Prozessor durch zusätzliche Maßnahmen.

### Verlustleistung und Energie pro Bild

Ein Vergleich der Verlustleistung bzw. des Energiebedarfs mit Werten aus der Literatur ist aus Mangel an verfügbaren Daten nicht möglich.

# 6 Zusammenfassung

Der Entwurfsraum komplexer Systeme vergrößert sich exponentiell mit steigender Anzahl an Designparametern. Aus der sich ergebenden Vielzahl an Designvarianten müssen möglichst früh im Entwurfsprozess gezielt diejenigen Varianten identifiziert werden, die ein Pareto-Optimum hinsichtlich der Optimierungskriterien darstellen. Ein systematisches Vorgehen während der Systemimplementierung ist somit von essentieller Bedeutung.

Diese Arbeit befasst sich mit der Entwurfsraumexploration einer FPGA-basierten, heterogenen Architektur sowie deren Optimierung. Dabei wird die Exploration durch Modellbildung gestützt, um das Durchsuchen des Entwurfsraums zu beschleunigen und zu vereinfachen. Sowohl der Entwurf der Architektur, deren Modellierung, als auch das Konzept zur Entwurfsraumexploration sind dabei die Ziele dieser Arbeit.

Die hier ausgewählte Anwendung der Bildmerkmalsextraktion bildet eine Basiskomponente der Bildverarbeitung. Der verwendete SIFT-Algorithmus ist dabei eine komplexe Variante aus diesem Feld mit hoher algorithmischer Qualität bei gleichzeitigem großen Rechenleistungs- und Speicherbedarf. Der Algorithmus gliedert sich in einen Pixel- und einen Objektteil mit unterschiedlichen Anforderungen. Der Pixelteil erfordert einen hohen Datendurchsatz und bietet Parallelisierungspotential, während der Objektteil neben moderatem Rechenleistungsbedarf auch ein hohes Maß an Kontrollfluss beinhaltet. Es wurde hier eine heterogene Architektur entworfen, die den Pixelteil in einem dedizierten Beschleuniger, in dieser Arbeit als SIFT Detection Engine, kurz SDE, bezeichnet, realisiert und den Objektteil auf einen VLIW-Prozessor abbildet, der in seinem Instruktionssatz auf die Anwendung angepasst ist.

Der SDE implementiert die Gauß-Pyramide und die Extraktion von Merkmalskandidaten. Zudem wird ein Pixelfenster zur Deskriptorberechnung bereitgestellt. Die Filter innerhalb der Gauß-Pyramide können im Time-Division-Multiplexing (TDM-Modus bis zu einem Faktor 6×) betrieben werden. Dabei arbeiten die Filter bei gleichem Durchsatz intern mit einer höheren Taktfrequenz bis zu 400 MHz, um die Anzahl an notwendigen DSP-Slices zu reduzieren. Aufgrund der dadurch limitierten maximalen Taktfrequenz sinkt auch die maximale Bildrate. Zudem hat sich in der Realisierung des TDM-Modus ein erhöhter Energiebedarf ergeben.

Im VLIW-Prozessor werden die Merkmalskandidaten im Subpixelbereich lokalisiert und auf Stabilität geprüft. Zudem wird ein Deskriptor berechnet. Dabei ist der VLIW-Prozessor mit dem SDE über FIFO-Speicher gekoppelt. Im VLIW-Prozessor wurden mit Blick auf die vorliegende Anwendung die Konzepte Parallelität (Duplizieren von Rechenressourcen), Spezialisierung (Hinzufügen neuer funktionaler Einheiten und Koprozessoren) und Pipeline-Optimierung (zusätzliche Register) untersucht. In der Untersuchung konnte

der Durchsatz des Prozessors für diese Anwendung anhand dieser drei Konzepte erhöht werden. Zudem ist durch dedizierte Prozessorerweiterungen der dynamische Energiebedarf pro Bild gesunken, und gleiche Bildraten ließen sich bereits bei einer geringeren Taktfrequenz erreichen. Die Taktfrequenzen des SDE und des VLIW-Prozessor stellen Designparameter dar, die nicht nur zur Designzeit, sondern im entworfenen System auch im laufenden Betrieb zwischen der Berechnung von zwei Bildern verändert werden können. Das Verhältnis der beiden Taktfrequenzen unter Berücksichtigung der Architekturkonfiguration bildet den Betriebspunkt des Systems. Durch Einstellen eines optimalen Betriebspunktes, an dem der Durchsatz der beiden Architekturkomponenten balanciert ist, konnte der dynamische Energiebedarf pro Bild reduziert werden. Zudem wurde gezeigt, dass die maximale dynamische Verlustleistung bei gleichem Durchsatz erheblich sinkt.

Zur systematischen Exploration des Entwurfsraums wurde eine Kombination unterschiedlicher Ansätze verwendet. Die Architekturbeschreibung erfolgte als parametrisierte Implementierung in einer Hardwarebeschreibungssprache. Die Anwendung wurde direkt in Assemblerbefehlen verfasst und mit Schaltern für die Designparameter versehen. Als Parameter wurden die VLIW-Prozessorkonfiguration, VLIW-Pipeline-Konfiguration, TDM-Modus im SDE und die beiden Taktfrequenzen untersucht. Primäre Optimierungskriterien sind die maximale Bildrate an einem balancierten Betriebspunkt, die dynamische Energie pro Bild, sowie die dynamische Verlustleistung. Das System wurde geteilt, sodass zuerst der Entwurfsraum des VLIW-Prozessors ohne TDM-Modus untersucht wurde und dann in einem zweiten Schritt der TDM-Modus im SDE auf bekannten VLIW-Konfigurationen.

Jeder Auswertung einer einzelnen Designvariante des VLIW-Prozessor geht ein Instruction Scheduling und eine Bitstream-Synthese voraus, wodurch die maximale Taktfrequenz und die Anzahl an Takten pro Merkmalskandidat ermittelt werden können. Ein emulationsbasiertes Profiling ist nur einmal pro VLIW-Prozessorkonfiguration nötig und kann für verschiedene Pipeline-Konfigurationen extrapoliert werden. Auf Basis dessen erfolgt die Generierung eines architekturbasierten, analytischen Modells für die Laufzeit des Systems. Zur Durchsuchung des Entwurfsraums wird die Rechenzeit pro Merkmalskandidat als sekundäres Optimierungskriterium verwendet, welches zusammen mit der maximalen Taktfrequenz des Prozessors eine sekundäre Pareto-Front liefert. Durch Auswertung der Ergebnisse einiger initialer Pipeline-Konfigurationen wurden manuell weitere Pipeline-Konfigurationen erzeugt und untersucht, bis ein Abflachen der Pareto-Front erkennbar war. Die VLIW-Konfigurationen der sich ergebenden sekundären Pareto-Front wurden danach in einer emulationsbasierten Messung der Verlustleistung an einigen zuvor ausgewählten Betriebspunkten für einen Benchmark-Datensatz charakterisiert. Darauf aufbauend erfolgte eine regressionsbasierte Modellierung der Verlustleistung. Abschließend wurden die VLIW-Konfigurationen mithilfe der beiden analytischen Modellgleichungen (Laufzeit und Verlustleistung) auf einer großen Anzahl an feingranularen Betriebspunkten ausgewertet, welche in ihrer Anzahl die zuvor in Emulation durchlaufenen Betriebspunkte um ein Vielfaches übersteigen. Dieses modellgestützte Vorgehen ermöglicht eine beinahe vollständige Suche innerhalb eines begrenzten Bereichs des Entwurfsraums. Durch diese

Suche konnte die finale Pareto-Front für die primären Optimierungskriterien gefunden werden. Mit einer Validierung in Emulation auf einem umfangreichen Testdatensatz wurden ausgewählte Konfigurationen abschließend bewertet.

Nachdem diese erste Untersuchung die zu betrachtenden VLIW-Pipeline-Konfigurationen klar definiert, ergibt sich ein identisches Vorgehen für die Untersuchung des TDM-Modus, basierend auf Bitstream-Synthesen, emulationsbasierter Messung der Verlustleistung und modellgestützter Variation des Betriebspunktes.

Die Tatsache, dass die Taktfrequenzen von VLIW-Prozessor und SDE zur Laufzeit angepasst werden können, ermöglicht eine dynamische Anpassung des Systems auf die aktuelle Performance-Anforderung. Da sich die Anzahl an Merkmalen innerhalb einer Bildsequenz stark ändert, ergibt sich für den VLIW-Prozessor eine veränderliche Rechenlast. Im Rahmen dieser Arbeit wurde eine dynamische Anpassung der Taktfrequenzen unter Einhaltung einer Zielbildrate und eines Zielbetriebspunktes untersucht. Dabei wurde das Laufzeitmodell verwendet, um die notwendigen Taktfrequenzen zur Laufzeit zu ermitteln. Es konnte dabei eine Reduktion der maximalen dynamischen Verlustleistung erreicht werden. Als exemplarische Anwendung für Bildmerkmale wird im Rahmen dieser Arbeit merkmalsbasierte Eigenbewegungsschätzung betrachtet. In diesem Kontext kann anstatt einer Minimalbildrate ein maximaler Abstand zwischen zwei Bildern als Zielkriterium vorgegeben werden. Bei adaptiver Taktanpassung sinkt der Verlustleistungsbedarf des Systems dabei zusätzlich noch einmal deutlich.

Im Rahmen dieser Arbeit konnte eine hochleistungsfähige Realisierung des Zielalgorithmus auf dem FPGA konzipiert und implementiert werden, welche eine Weiterentwicklung des aktuellen Stands der Forschung darstellt. Die Aufteilung des Algorithmus in einen Pixel- und einen Objektteil gilt für viele Verfahren der Bildverarbeitung. Das entworfene heterogene System, das einen dedizierten Beschleuniger mit einem VLIW-Prozessor kombiniert, ist somit typisch für diese Klasse von Anwendungen. Folglich lässt sich auch der in dieser Arbeit vorgestellte Ansatz der Modellbildung für andere Anwendungen verallgemeinern. Insbesondere mithilfe des architekturbasierten Laufzeitmodells können durch gezielte Anpassungen der Modellparameter Performance-Vorhersagen auch für zukünftige Algorithmen oder Architekturen gemacht werden. Neben der Möglichkeit zur Vorhersage konnte eine direkte Anwendung der Architekturmodellierung im Rahmen einer umfangreichen Entwurfsraumexploration gezeigt werden. Die Modelle dienen dabei der beschleunigten Suche nach Pareto-optimalen Designvarianten in Teilbereichen des Entwurfsraums entlang bestimmter, diskretisierbarer Designparameter und erhöhen den realisierbaren Umfang der Suche. Außerdem wurde mithilfe der Modelle ein adaptives Systemverhalten zur Laufzeit ermöglicht, indem die Zieltaktfrequenzen bei dynamischer Performance-Anforderung auf Basis der Modelle ermittelt wurden.

Mit dem Abschluss dieser Arbeit steht ein Konzept zur Verfügung, mit dem heterogene Systeme modelliert und in ihrem Entwurfsraum systematisch untersucht werden können. Es konnte gezeigt werden, dass Modellgleichungen dabei einen wertvollen Beitrag zur Vorhersage von Zielindikatoren sowohl während des Entwurfsprozesses als auch dynamisch zur Laufzeit liefern.

# A Anhang

## A.1 Exemplarische Darstellung von SIFT-Merkmalen

Abb. A.1 zeigt zur Verdeutlichung die extrahierten Bildmerkmale nach dem hier verwendeten SIFT-Algorithmus. Der Algorithmus ist in Abschnitt 2.2.4 beschrieben. Das Bild zeigt eine Fahrszene in einem Wohngebiet aus der KITTI Vision Benchmark Suite [Gei13] (vgl. Abschnitt A.5). Die SIFT-Merkmalskandidaten sind als Punkte dargestellt. Verifizierte, stabile SIFT-Merkmale bilden einen Kreis um den Kandidaten. Der Radius dieses Kreises ist proportional zu derjenigen Position innerhalb der Gauß-Pyramide, auf der das Merkmal gefunden wurde. Größere Kreise entsprechen dabei Merkmalen mit größerer Skalierung. Die erste Oktave ist in schwarz und alle weiteren Oktaven sind in weiß gezeichnet. Dabei wird deutlich, dass die meisten Merkmale auf dieser ersten Oktave zu detektieren sind. Wenn ein Punkt von keinem Kreis umgeben wird, dann wurde der entsprechende Merkmalskandidat vom Algorithmus verworfen. Bei manchen Merkmalen ist zu erkennen, dass der Kandidatenpunkt nicht dem Mittelpunkt des umgebenen Kreises entspricht. Dies ist mit dem algorithmischen Schritt der Subpixel-Lokalisierung der Kandidaten zu begründen. Aufgrund der höheren Skalierung ist der Effekt bei größeren Kreisradien etwas besser zu erkennen. Das Bild wird von der Hardware zeilenweise verarbeitet. Da die Merkmale sichtbar nicht gleichverteilt im Bild sind, ist der zeitliche Abstand, mit dem die Kandidaten von der Hardware gefunden werden, nicht konstant.

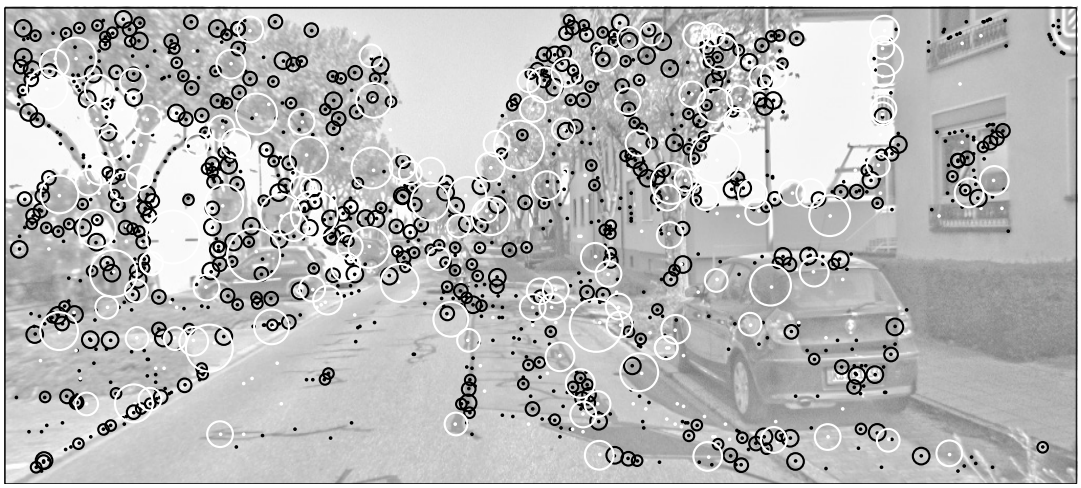
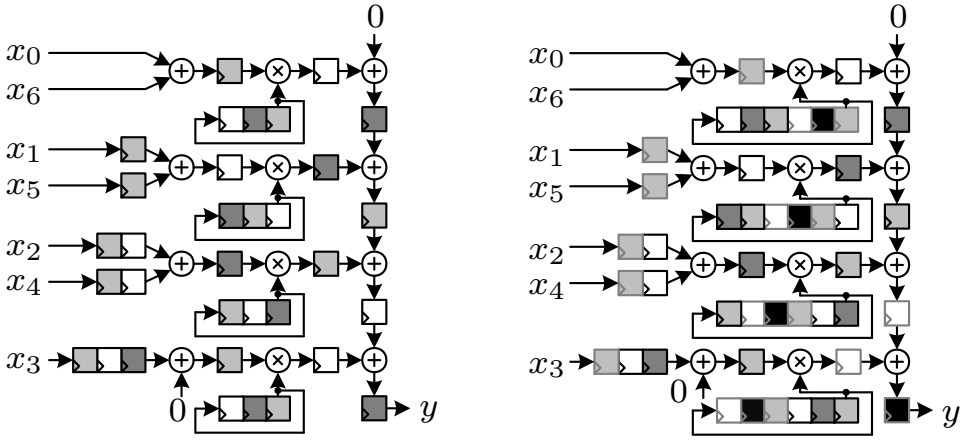


**Abbildung A.1:** Exemplarische Darstellung von SIFT-Merkmalen (Sequenz 00)

## A.2 Weitere Abbildungen

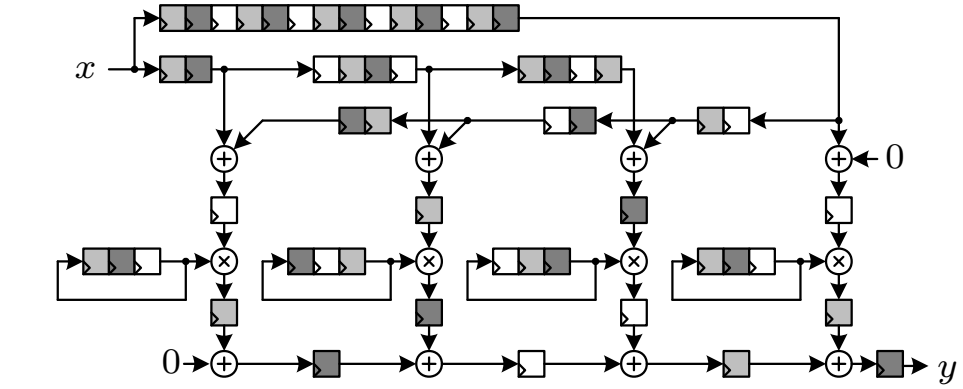
### A.2.1 Aufbau der Gauß-Filter

Zur Diskussion und Erklärung siehe Abschnitt 3.4 auf Seite 49.

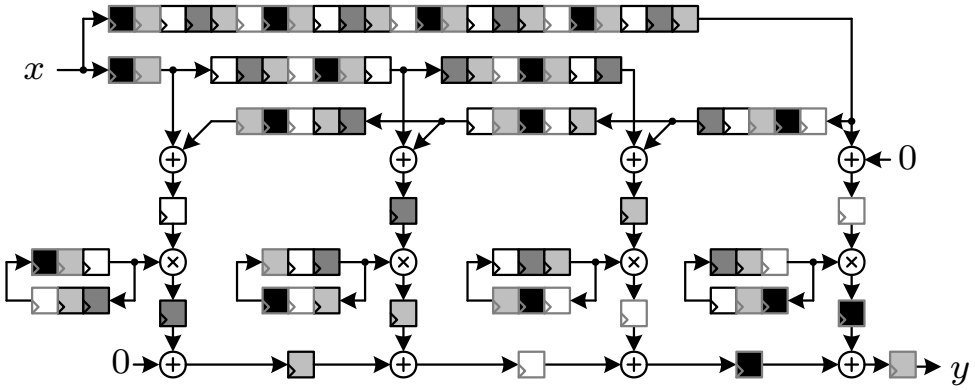



**(a)** TDM×3 **(b)** TDM×6

**Abbildung A.2:** 1D-FIR-Filter zur vertikalen Bildfilterung exemplarisch mit sieben Taps und vorgeschalteten Addierern aufgrund der symmetrischen Filterfunktion. Durch Vervielfachung des internen Filtertakts aller dargestellten Register und zyklisches Schieben der Koeffizienten kann das Filter im Time-Division-Multiplexing (TDM) betrieben werden. Dieselben Eingangsdaten werden mit unterschiedlichen Koeffizienten multipliziert. Da die Filterhardware geteilt wird, muss für gleichen Durchsatz die Filtertaktfrequenz im Vergleich zur umliegenden Hardware um den TDM-Faktor ansteigen. Die verschiedenen Farben der Register sollen die unterschiedlichen Phasen des Filters andeuten. Im Vergleich zur normalen Implementierung ohne TDM ist nur geringer weiterer Hardwareaufwand in Form von Schieberegistern der Koeffizienten erforderlich.




**(a)** TDM×3




**(b)** TDM×6

**Abbildung A.3:** 1D-FIR-Filter zur horizontalen Bildfilterung exemplarisch mit sieben Taps und vorgeschalteten Addierern aufgrund der symmetrischen Filterfunktion. Durch Vervielfachung des internen Filtertakts aller dargestellten Register und zyklisches Schieben der Koeffizienten kann das Filter im Time-Division-Multiplexing (TDM) betrieben werden. Dieselben Eingangsdaten werden mit unterschiedlichen Koeffizienten multipliziert. Da die Filterhardware geteilt wird, muss für gleichen Durchsatz die Filtertaktfrequenz im Vergleich zur umliegenden Hardware um den TDM-Faktor ansteigen. Die verschiedenen Farben der Register sollen die unterschiedlichen Phasen des Filters andeuten. Im Vergleich zur normalen Implementierung ohne TDM ist erheblicher weiterer Hardwareaufwand in Form von Registern erforderlich, sodass die Abbildung auf den DSP-Slices im FPGA weniger effizient ist.

### A.2.2 VLIW-Programmstruktur als Kontrollflussgraph

Zur Verdeutlichung der Programmstruktur der Anwendung auf dem VLIW-Prozessor ist der Kontrollflussgraph in Abb. A.4 gegeben. Aufgrund der Kontrollstrukturen (bedingte Sprünge) ist die Programmlaufzeit pro Merkmalskandidat datenabhängig.

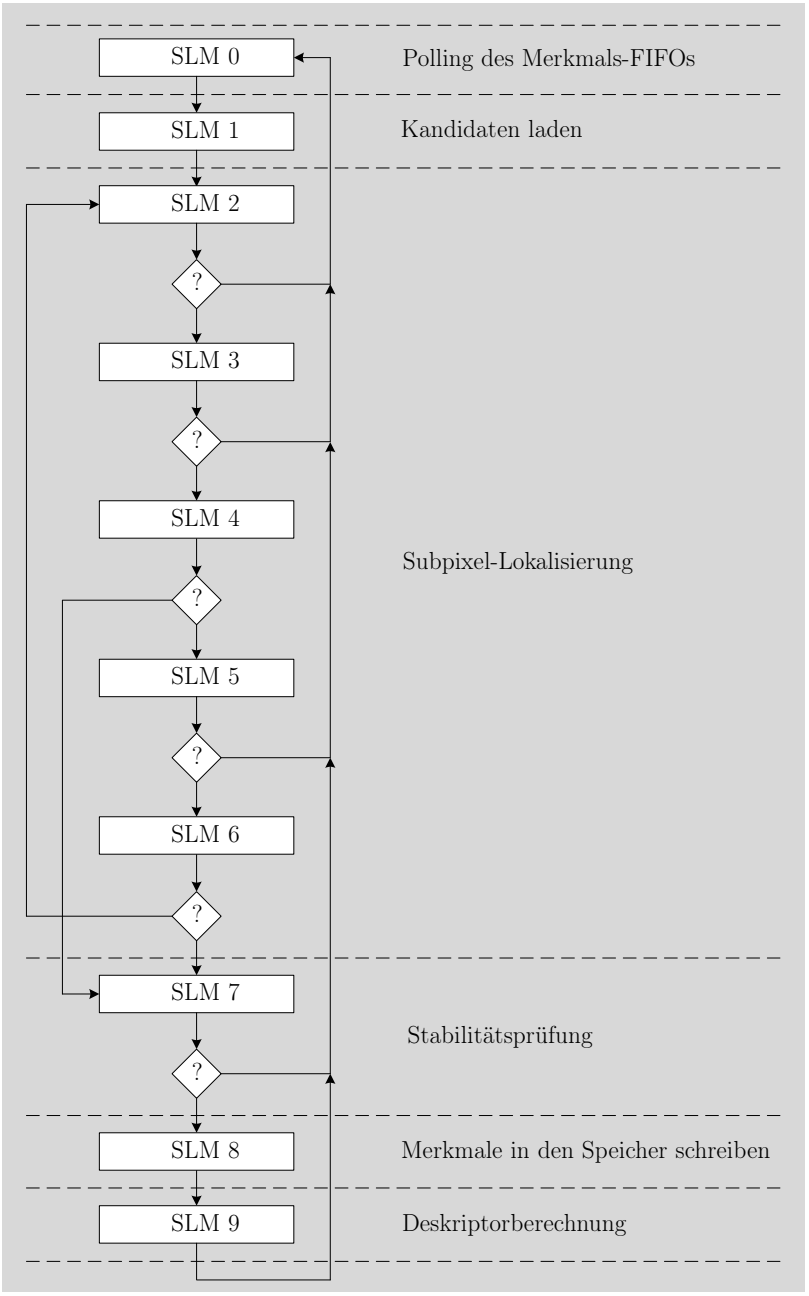



**Abbildung A.4:** VLIW-Programmstruktur als Kontrollflussgraph

### A.2.3 Kommunikationsablauf zwischen VLIW-Prozessor und Host-PC

Abb A.5 zeigt den zeitlichen Ablauf der Kommunikation zwischen VLIW-Prozessor und Host-PC während eines Bildzyklus ohne *double buffering* unter Vorgabe einer Zielbildrate $R_{soll} = 1/t_{R,soll}$. Mit *double buffering* wird zur Steigerung des Durchsatzes in der Wartezeit des Hosts während der Prozessierung auf dem FPGA schon das nächste Bild übertragen. Die Taktrekonfigurationszeit $t_{reconfig}$ ist in der Realität um Größenordnungen kleiner als die tatsächliche Bildverarbeitungsdauer $t_{img}$ und hier nur zur Verdeutlichung länger dargestellt. Bei der zweiten Rekonfiguration können die Taktfrequenzen beliebig verändert werden. Die einzelnen Zeiten werden mit Performance-Countern zur Laufzeit ermittelt.

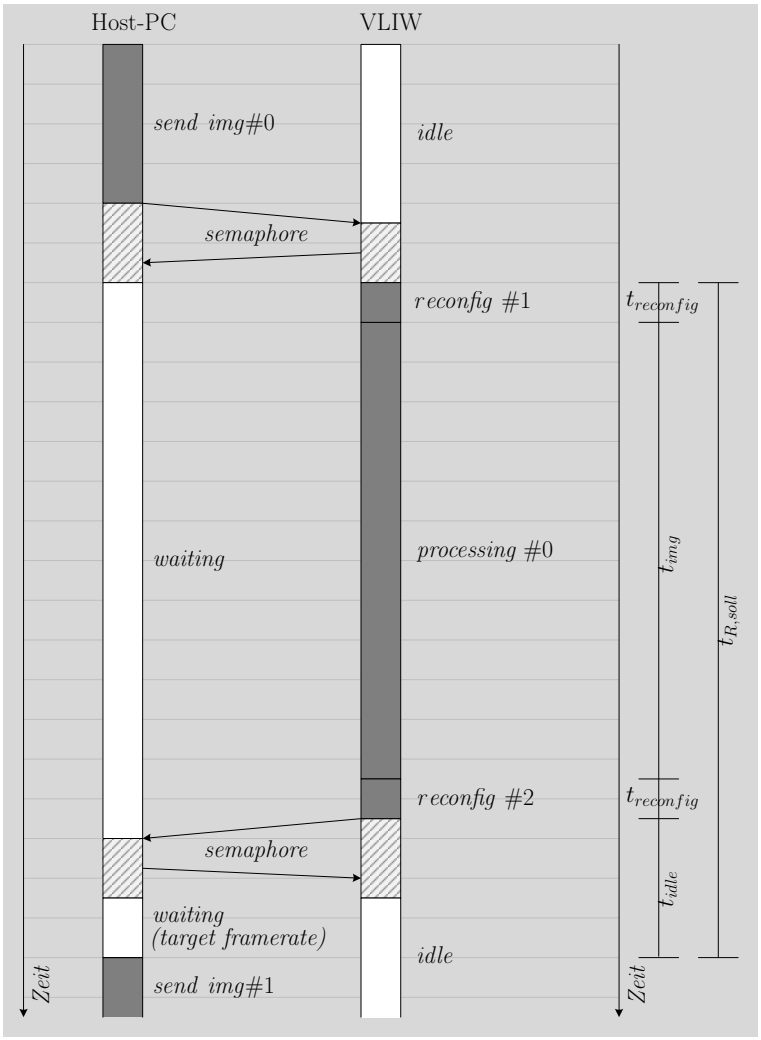



**Abbildung A.5:** Kommunikationsablauf zwischen VLIW-Prozessor und Host-PC

## A.3 Grafische Benutzeroberfläche zur Verlustleistungsmessung

Im Rahmen dieser Arbeit wird die Leistungsaufnahme des verwendeten FPGAs gemessen (vgl. Abschnitt 4.3.2). Das Abfragen und Speichern von Messwerten erfolgt mittels eines selbstgeschriebenen C-Programms, das über die Kommandozeile ausgeführt wird und etwa alle 2 ms einen neuen Wert speichern kann.

Zu Verifikationszwecken und zur Visualisierung der Messwerte wurde ein weiteres Programm entwickelt, welches eine grafische Benutzeroberfläche (GUI) bereitstellt. Dafür wurde das Software-Framework Qt [Qt20] verwendet, welches eine plattformübergreifende Entwicklung von Programmen und grafischen Benutzeroberflächen ermöglicht.

Mit dem GUI-Programm kann ebenfalls eine Messung der Leistung, des Stroms und der Spannung durchgeführt und live beobachtet werden. Außerdem wurde eine Möglichkeit zur Mittelwertbildung über ein selbst wählbares Zeitfenster im Programm integriert. Aufgrund des zusätzlichen Aufwands durch die GUI liegt die Abtastrate für Messwerte jedoch niedriger als beim Kommandozeilen-Programm.

Darüber hinaus können Messwerte als .csv-Datei importiert und visualisiert werden (siehe Abb. A.6). Eine Beschriftung der Messwerte erfolgt durch den zusätzlichen Import einer *Events*-Datei, die von der SIFT-Anwendung auf dem Host-PC geschrieben wird und mit Zeitstempeln versehene Strings enthält. Diese Strings enthalten Informationen zum Systemzustand bzw. Zustandsänderungen während der Bildverarbeitung, wodurch die Messwerte mit der Berechnung auf dem FPGA in Zusammenhang gebracht werden können.

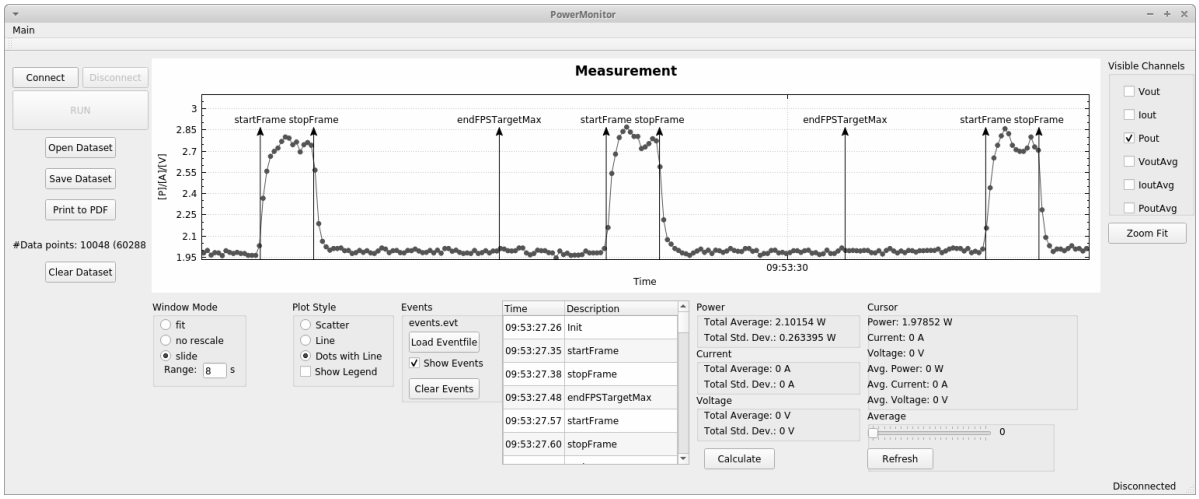



**Abbildung A.6:** Grafische Benutzeroberfläche zur Messung und Live-Visualisierung der Leistung. Die aufgenommenen Messwerte können über ihre Zeitstempel mit Kommentaren annotiert werden, welche dem jeweiligen Zustand der SIFT-Anwendung entsprechen.

# A.4 Rohdaten

## A.4.1 Eigenbewegungsschätzung

Im Folgenden sind die Rohdaten der rotatorischen und translatorischen Fehler aus der Untersuchung in Abschnitt 3.7.2 aufgeführt.

**Tabelle A.1:** Rotatorischer Fehler in ($10^{-4} rad/m$) und translatorische Fehler in (%) über der Pfadlänge

| Pfadlänge | Soft2 | | Stereoscan | | OpenCV | | OpenCVsimple | | FPGA | |
|---|---|---|---|---|---|---|---|---|---|---|
| ($m$) | rot. | tr. | rot. | tr. | rot. | tr. | rot. | tr. | rot. | tr. |
| 100 | 0,46 | 0,92 | 2,41 | 1,42 | 1,76 | 5,62 | 1,75 | 5,36 | 1,75 | 5,59 |
| 200 | 0,32 | 0,76 | 2,19 | 1,71 | 1,53 | 5,24 | 1,51 | 4,96 | 1,50 | 5,16 |
| 300 | 0,25 | 0,68 | 2,07 | 2,11 | 1,42 | 5,01 | 1,40 | 4,70 | 1,38 | 4,89 |
| 400 | 0,21 | 0,62 | 1,95 | 2,48 | 1,34 | 4,89 | 1,32 | 4,57 | 1,29 | 4,77 |
| 500 | 0,18 | 0,57 | 1,85 | 2,81 | 1,28 | 4,84 | 1,27 | 4,53 | 1,24 | 4,73 |
| 600 | 0,16 | 0,52 | 1,78 | 3,09 | 1,24 | 4,80 | 1,23 | 4,49 | 1,19 | 4,68 |
| 700 | 0,15 | 0,49 | 1,73 | 3,27 | 1,20 | 4,72 | 1,20 | 4,41 | 1,15 | 4,64 |
| 800 | 0,14 | 0,46 | 1,70 | 3,45 | 1,18 | 4,68 | 1,18 | 4,39 | 1,14 | 4,65 |

**Tabelle A.2:** Rotatorischer Fehler in ($10^{-4} rad/m$) und translatorische Fehler in (%) über der Geschwindigkeit

| Geschwindigkeit | Soft2 | | Stereoscan | | OpenCV | | OpenCVsimple | | FPGA | |
|---|---|---|---|---|---|---|---|---|---|---|
| ($m/s$) | rot. | tr. | rot. | tr. | rot. | tr. | rot. | tr. | rot. | tr. |
| 4 | 0,70 | 0,99 | 2,44 | 1,48 | 2,18 | 2,63 | 2,08 | 2,58 | 1,92 | 2,45 |
| 6 | 0,34 | 0,90 | 2,07 | 2,62 | 1,63 | 3,24 | 1,60 | 3,00 | 1,53 | 3,14 |
| 8 | 0,26 | 0,69 | 1,85 | 2,43 | 1,44 | 4,32 | 1,44 | 3,97 | 1,39 | 4,09 |
| 10 | 0,22 | 0,56 | 1,76 | 2,22 | 1,25 | 5,32 | 1,26 | 4,95 | 1,23 | 5,09 |
| 12 | 0,29 | 0,69 | 2,01 | 1,92 | 1,16 | 5,68 | 1,14 | 5,34 | 1,11 | 5,41 |
| 14 | 0,35 | 0,75 | 1,81 | 1,76 | 1,10 | 7,51 | 1,06 | 7,10 | 1,14 | 7,05 |
| 16 | 0,30 | 0,68 | 1,53 | 1,78 | 1,43 | 10,92 | 1,33 | 13,60 | 2,15 | 17,11 |
| 18 | 0,17 | 0,46 | 1,62 | 2,70 | 1,78 | 7,56 | 1,82 | 17,43 | 1,95 | 19,96 |
| 20 | 0,17 | 0,46 | 2,46 | 2,77 | 1,95 | 7,10 | 2,08 | 13,64 | 2,04 | 14,15 |
| 22 | 0,18 | 0,47 | 2,50 | 2,75 | 1,92 | 9,65 | 1,97 | 11,47 | 2,26 | 11,89 |
| 24 | 0,16 | 0,71 | 2,03 | 2,42 | 2,11 | 10,73 | 1,84 | 10,01 | 2,18 | 11,71 |

## A.4.2 VLIW-Pipeline-Konfigurationen

Dieser Abschnitt listet die in dieser Arbeit untersuchten Pipeline-Konfigurationen des VLIW-Prozessors auf (vgl. Abb.3.10). Eine Übersicht über die dabei möglichen Pipeline-Schalter ist in Tab. A.3 gegeben.

Die Anzahl an verschiedenen Kombinationen der Pipeline-Schalter beträgt mehr als 300.000. Daher wurde im Rahmen dieser Arbeit empirisch durch schrittweise Untersuchung eine Auswahl von 86 Kombinationen erarbeitet. Die Tab. A.4-A.9 listen diese Pipeline-Konfigurationen für drei VLIW-Prozessorkonfigurationen $\#b0/\#_{base}$, $\#b4x2/\#_{par}$ und $\#c2x2/\#_{spez}$ nach ihrem jeweiligen kritischem Pfad ($p_{VLIW,min}$) sortiert. Alle drei VLIW-Prozessorkonfigurationen nutzen dieselben Pipeline-Konfigurationen, welche durch die Sortierung nur in einer anderen Reihenfolge in den Tabellen stehen.

Die Spalte Σ Latenz summiert alle Pipeline-Schalter auf, wobei PL_FWD eine Gewichtung von vier erhält, da vier FUs von diesem Schalter betroffen sind. Als Basiskonfiguration wird diejenige Konfiguration definiert, die eine summierte Latenz von eins besitzt. Insgesamt sollte die Summe über alle Konfigurationen stetig steigen. Spitzen in der Summe entstehen immer dann, wenn der entsprechende Pipeline-Schalter die Taktperiode nicht senken konnte, d. h. der entsprechende Schaltungsteil nicht im kritischen Pfad war.

Konfigurationen, die mit einem *(P)* markiert sind, sind Pareto-optimal bezüglich des kritischem Pfades ($p_{VLIW,min}$) und der Rechenzeit pro Merkmalskandidat ($t_{VLIW,workPerCand}$). Eine *-Markierung entspricht denjenigen Konfigurationen mit dem kleinsten Wert für $t_{VLIW,workPerCand}$ sowohl ohne als auch mit PL_FWD.

Abschließend in diesem Abschnitt gibt Tab. A.10 eine Übersicht der Konfigurationen mit *-Markierung für alle in dieser Arbeit betrachteten VLIW-Prozessorkonfigurationen sowie einen Überblick über die Anzahl an Logic Slices für VLIW, SDE und das Gesamtsystem.

**Tabelle A.3:** Übersicht VLIW-Pipeline-Schalter

| Kürzel | Zugehörigkeit | Werte | Beschreibung |
|---|---|---|---|
| <PL_DE> | Prozessor | 0-1 | Decode-Stage (+1 Takt Latenz auf Branches) |
| <PL_RFBYP> | Prozessor | 0-1 | Registerfile Bypassing (Wenn die gleiche Adresse im selben Takt gelesen und geschrieben wird, sorgt ein Multiplexer dafür, das Registerfile zu umgehen. Das Register nach dem Multiplexer wird mit diesem Schalter vor dem Multiplexer platziert und im BRAM absorbiert. Der Bypass wird dadurch ins Forwarding geschoben, erzeugt aber einen kürzeren Pfad am Registerfile-Ausgang. Es gibt keine Latenz-Penalty.) |
| <PL_BUSW> | Prozessor | 0-1 | Bus Write (+1 Takt Latenz auf alle STOREs) |
| <PL_FLOW> | Prozessor | 0-1 | Flow Unit (+1 Takt Latenz auf alle Sprünge) |
| <PL_MV> | Prozessor | 0-1 | Move (Der Pfad von der EX-Stage zur DE-Stage zum Schreiben in ein spezielles Registerfile wird zusätzlich registriert, wodurch beim Schreiben in dieses Registerfile +1 Takt Latenz entsteht.) |
| <PL_FWD> | Prozessor | 0-1 | Forwarding Multiplexer (+1 Takt Latenz auf alle FUs) |
| <PL_AU> | FU | 0-3 | Arithmetic Unit (+0-3 Takte Latenz auf AU-Operationen) |
| <PL_MMU> | FU | 0-2 | Mul-MAC-Unit (+0-2 Takte Latenz auf MMU-Operationen) |
| <PL_PER> | FU | 0-2 | Permute-Unit (+0-2 Takte Latenz auf PER-Operationen) |
| <PL_SRU> | FU | 0-2 | Shit-Round-Unit (+0-2 Takte Latenz auf SRU-Operationen) |
| <PL_RFU> | FU | 0-2 | applikationsspez. Spezialinstruktion *getIdxPatch* (+0-2 Takte Latenz) |
| <PL_DIV1> | CU | 0-3 | erster Divisionskoprozessor (+7/11/19/35 Takte Latenz) |
| <PL_DIV2> | CU | 0-3 | zweiter Divisionskoprozessor (+5/7/11/19 Takte Latenz) |

**Tabelle A.4:** Pipeline-Konfigurationen für die VLIW-Prozessorkonfiguration $\#b0/\#_{base}$

| Konfiguration | $p_{VLIW,min}$ (ns) | $N_{cand}$ (Takte) | dynamischer IPC | $t_{VLIW,workPerCand}$ ($\mu$s) | <PL_AU> | <PL_MMU> | <PL_MV> | <PL_PER> | <PL_SRU> | <PL_RFBYP> | <PL_FWD> | <PL_BUSW | <PL_FLOW | <PL_DE | <PL_RFU> | <PL_DIV1> | <PL_DIV2> | Σ Latenz |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 (P) | 26,91 | 785 | 1,54 | 21,12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 (P) | 26,84 | 786 | 1,54 | 21,09 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 3 | 26,20 | 863 | 1,40 | 22,61 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 4 |
| 4 (P) | 26,19 | 797 | 1,52 | 20,88 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 |
| 5 (P) | 26,02 | 786 | 1,54 | 20,46 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 6 | 24,97 | 866 | 1,39 | 21,62 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |
| 7 (P) | 22,21 | 797 | 1,52 | 17,70 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 3 |
| 8 (P) | 20,32 | 786 | 1,54 | 15,97 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 |
| 9 (P) | 19,64 | 785 | 1,55 | 15,42 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 3 |
| 10 (P) | 19,53 | 786 | 1,54 | 15,35 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 3 |
| 11 (P) | 18,04 | 785 | 1,54 | 14,16 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 4 |
| 12 (P) | 16,46 | 796 | 1,52 | 13,11 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 4 |
| 13 (P) | 16,05 | 794 | 1,52 | 12,75 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 3 |
| 14 | 15,93 | 824 | 1,45 | 13,12 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 6 |
| 15 (P) | 15,38 | 796 | 1,52 | 12,24 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 5 |
| 16 | 14,32 | 963 | 1,25 | 13,79 | 2 | 2 | 1 | 2 | 2 | 1 | 0 | 1 | 0 | 1 | 2 | 2 | 3 | 19 |
| 17 | 14,18 | 874 | 1,38 | 12,39 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 7 |
| 18 (P) | 14,03 | 867 | 1,38 | 12,17 | 1 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 3 | 14 |
| 19 (P) | 13,98 | 865 | 1,39 | 12,10 | 1 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 2 | 1 | 3 | 14 |
| 20 (P) | 13,81 | 797 | 1,52 | 11,01 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 5 |
| 21 | 13,71 | 863 | 1,39 | 11,83 | 1 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 2 | 12 |
| 22 | 13,64 | 929 | 1,30 | 12,67 | 1 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 2 | 3 | 3 | 17 |
| 23 | 13,55 | 876 | 1,37 | 11,87 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 8 |
| 24 (P) | 13,41 | 796 | 1,52 | 10,68 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 6 |
| 25 | 13,24 | 836 | 1,38 | 11,07 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 9 |
| 26 | 13,07 | 865 | 1,40 | 11,30 | 1 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 3 | 13 |
| 27 (P) | 13,04 | 810 | 1,50 | 10,57 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | 7 |
| 28 | 12,99 | 835 | 1,43 | 10,85 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 8 |
| 29 | 12,93 | 886 | 1,35 | 11,46 | 1 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 14 |
| 30 | 12,93 | 818 | 1,48 | 10,58 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 2 | 8 |
| 31 | 12,87 | 843 | 1,43 | 10,85 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | 8 |
| 32 | 12,77 | 855 | 1,41 | 10,91 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 2 | 10 |
| 33 | 12,72 | 956 | 1,24 | 12,16 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 10 |
| 34 (P) | 12,62 | 824 | 1,45 | 10,40 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 8 |
| 35 | 12,54 | 849 | 1,41 | 10,64 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | 9 |
| 36 | 12,53 | 836 | 1,43 | 10,48 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 2 | 9 |
| 37 (P) | 12,48 | 822 | 1,45 | 10,26 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 7 |
| 38 | 12,48 | 834 | 1,44 | 10,40 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 2 | 8 |
| 39 | 12,42 | 862 | 1,39 | 10,70 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 2 | 11 |
| 40 | 12,38 | 887 | 1,36 | 10,98 | 1 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 2 | 2 | 3 | 16 |
| 41 | 12,32 | 1139 | 1,06 | 14,03 | 3 | 2 | 1 | 2 | 2 | 1 | 0 | 1 | 1 | 1 | 2 | 3 | 3 | 22 |
| 42 | 12,28 | 850 | 1,41 | 10,44 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 2 | 9 |
| 43 | 12,12 | 848 | 1,42 | 10,28 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 11 |

**Tabelle A.5:** Pipeline-Konfigurationen für die VLIW-Prozessorkonfiguration $\#b0/\#_{base}$ (Fortsetzung)

| Konfiguration | $p_{VLIW,min}$ (ns) | $N_{cand}$ (Takte) | dynamischer IPC | $t_{VLIW,workPerCand}$ ($\mu$s) | <PL_AU> | <PL_MMU> | <PL_MV> | <PL_PER> | <PL_SRU> | <PL_RFBYP> | <PL_FWD> | <PL_BUSW | <PL_FLOW | <PL_DE | <PL_RFU> | <PL_DIV1> | <PL_DIV2> | Σ Latenz |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 44 (P) | 12,10 | 836 | 1,43 | 10,11 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 2 | 8 |
| 45 (P) | 12,09 | 836 | 1,44 | 10,10 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 9 |
| 46 | 12,07 | 849 | 1,42 | 10,24 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 2 | 10 |
| 47 | 12,02 | 846 | 1,43 | 10,17 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 10 |
| 48 | 12,01 | 876 | 1,38 | 10,52 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 9 |
| 49 (P) | 11,99 | 835 | 1,44 | 10,01 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 9 |
| 50 | 11,98 | 889 | 1,36 | 10,65 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 2 | 10 |
| 51 | 11,93 | 853 | 1,41 | 10,18 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 2 | 11 |
| 52 | 11,91 | 898 | 1,34 | 10,70 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 2 | 11 |
| 53 | 11,89 | 848 | 1,42 | 10,08 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 10 |
| 54 | 11,88 | 892 | 1,35 | 10,60 | 1 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 17 |
| 55 | 11,85 | 848 | 1,42 | 10,04 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 2 | 9 |
| 56 | 11,81 | 850 | 1,42 | 10,04 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 2 | 10 |
| 57 (P) | 11,79 | 848 | 1,42 | 10,00 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 2 | 9 |
| 58 | 11,78 | 864 | 1,39 | 10,17 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 3 | 12 |
| 59 (P) | 11,56 | 862 | 1,40 | 9,96 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 2 | 10 |
| 60 | 11,50 | 884 | 1,36 | 10,17 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 2 | 10 |
| 61 | 11,39 | 977 | 1,23 | 11,13 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 2 | 14 |
| 62 (P) | 11,29 | 867 | 1,39 | 9,79 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 12 |
| 63 | 11,27 | 982 | 1,21 | 11,07 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 2 | 14 |
| 64* (P) | 11,17 | 842 | 1,44 | 9,41 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 2 | 9 |
| 65 | 11,03 | 979 | 1,23 | 10,79 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 2 | 14 |
| 66 | 11,00 | 993 | 1,21 | 10,93 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 16 |
| 67 | 10,99 | 992 | 1,19 | 10,90 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 17 |
| 68 | 10,99 | 972 | 1,23 | 10,68 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 2 | 13 |
| 69 | 10,77 | 993 | 1,21 | 10,69 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 2 | 15 |
| 70 | 10,69 | 977 | 1,22 | 10,45 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 2 | 15 |
| 71 | 10,62 | 1010 | 1,19 | 10,73 | 2 | 2 | 1 | 2 | 2 | 1 | 0 | 1 | 1 | 1 | 2 | 3 | 3 | 21 |
| 72 | 10,59 | 1000 | 1,19 | 10,59 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 2 | 16 |
| 73 | 10,54 | 993 | 1,20 | 10,46 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 2 | 15 |
| 74 | 10,40 | 1002 | 1,20 | 10,42 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 2 | 1 | 3 | 18 |
| 75 | 10,30 | 1001 | 1,20 | 10,31 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 3 | 17 |
| 76 | 9,95 | 1004 | 1,20 | 9,99 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 3 | 18 |
| 77 | 9,73 | 1001 | 1,20 | 9,74 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 18 |
| 78 | 9,55 | 996 | 1,21 | 9,51 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 3 | 16 |
| 79 | 9,43 | 1022 | 1,17 | 9,64 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 2 | 3 | 18 |
| 80 (P) | 9,39 | 996 | 1,20 | 9,35 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 17 |
| 81 | 9,23 | 1025 | 1,17 | 9,46 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 2 | 2 | 3 | 20 |
| 82 | 8,99 | 1257 | 0,97 | 11,30 | 2 | 2 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 2 | 3 | 3 | 25 |
| 83 | 8,95 | 1462 | 0,82 | 13,08 | 3 | 2 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 2 | 3 | 3 | 26 |
| 84 | 8,94 | 1067 | 1,12 | 9,54 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 2 | 3 | 3 | 21 |
| 85 | 8,92 | 1211 | 0,99 | 10,80 | 2 | 2 | 1 | 2 | 2 | 1 | 1 | 1 | 0 | 1 | 2 | 2 | 3 | 23 |
| 86* (P) | 8,75 | 1025 | 1,17 | 8,97 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 3 | 21 |

**Tabelle A.6:** Pipeline-Konfigurationen für die VLIW-Prozessorkonfiguration $\#b4x2/\#_{par}$

| Konfiguration | $p_{VLIW;min}$ (ns) | $N_{cand}$ (Takte) | dynamischer IPC | $t_{VLIW;workPerCand}$ ($\mu$s) | <PL_AU> | <PL_MMU> | <PL_MV> | <PL_PER> | <PL_SRU> | <PL_RFBYP> | <PL_FWD> | <PL_BUSW | <PL_FLOW | <PL_DE | <PL_RFU> | <PL_DIV1> | <PL_DIV2> | Σ Latenz |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 (P) | 32,64 | 532 | 2,28 | 17,36 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 (P) | 28,24 | 537 | 2,26 | 15,16 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 |
| 3 (P) | 26,27 | 577 | 1,98 | 15,16 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 4 |
| 4 (P) | 26,21 | 532 | 2,27 | 13,94 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 5 (P) | 26,04 | 530 | 2,29 | 13,80 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 25,56 | 579 | 1,97 | 14,80 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |
| 7 (P) | 25,33 | 536 | 2,26 | 13,58 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 3 |
| 8 (P) | 23,07 | 538 | 2,24 | 12,41 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 5 |
| 9 (P) | 22,92 | 533 | 2,28 | 12,22 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 3 |
| 10 | 22,86 | 536 | 2,26 | 12,26 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 4 |
| 11 (P) | 20,80 | 532 | 2,27 | 11,06 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 |
| 12 (P) | 19,25 | 532 | 2,28 | 10,24 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 3 |
| 13 (P) | 19,14 | 533 | 2,28 | 10,20 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 4 |
| 14 (P) | 17,54 | 536 | 2,24 | 9,40 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 5 |
| 15 (P) | 17,29 | 535 | 2,26 | 9,25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 3 |
| 16 | 16,60 | 576 | 2,10 | 9,56 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 6 |
| 17 (P) | 15,68 | 549 | 2,21 | 8,61 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | 7 |
| 18 (P) | 15,49 | 538 | 2,25 | 8,33 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 6 |
| 19 | 15,17 | 584 | 1,95 | 8,86 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 7 |
| 20 | 14,94 | 711 | 1,70 | 10,62 | 2 | 2 | 1 | 2 | 2 | 1 | 0 | 1 | 0 | 1 | 2 | 2 | 3 | 19 |
| 21 | 14,91 | 611 | 1,99 | 9,11 | 1 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 2 | 12 |
| 22 | 14,63 | 609 | 1,97 | 8,91 | 1 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 3 | 13 |
| 23 | 14,62 | 641 | 1,88 | 9,37 | 1 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 2 | 3 | 3 | 17 |
| 24 | 14,41 | 586 | 2,06 | 8,45 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 2 | 8 |
| 25 | 14,37 | 620 | 1,95 | 8,91 | 1 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 2 | 2 | 3 | 16 |
| 26 (P) | 14,27 | 576 | 2,10 | 8,22 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 8 |
| 27 | 14,13 | 609 | 1,97 | 8,61 | 1 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 3 | 14 |
| 28 | 13,89 | 593 | 2,04 | 8,23 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 2 | 9 |
| 29 | 13,88 | 609 | 1,97 | 8,45 | 1 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 2 | 1 | 3 | 14 |
| 30 | 13,88 | 619 | 1,96 | 8,59 | 1 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 14 |
| 31 (P) | 13,86 | 589 | 2,06 | 8,16 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 9 |
| 32 (P) | 13,71 | 586 | 2,07 | 8,03 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 2 | 9 |
| 33 | 13,64 | 608 | 1,97 | 8,29 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 2 | 11 |
| 34 (P) | 13,62 | 576 | 2,10 | 7,84 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 7 |
| 35 | 13,57 | 840 | 1,44 | 11,39 | 3 | 2 | 1 | 2 | 2 | 1 | 0 | 1 | 1 | 1 | 2 | 3 | 3 | 22 |
| 36 | 13,51 | 590 | 2,05 | 7,97 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 8 |
| 37 (P) | 13,37 | 586 | 2,07 | 7,83 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | 8 |
| 38 | 13,33 | 588 | 2,06 | 7,84 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 9 |
| 39 | 13,32 | 592 | 2,05 | 7,89 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 2 | 9 |
| 40 (P) | 13,28 | 586 | 1,95 | 7,78 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 9 |
| 41 (P) | 13,27 | 555 | 2,17 | 7,37 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 2 | 8 |
| 42 | 13,24 | 594 | 2,01 | 7,86 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 2 | 10 |
| 43 | 13,11 | 601 | 2,01 | 7,88 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 10 |

**Tabelle A.7:** Pipeline-Konfigurationen für die VLIW-Prozessorkonfiguration $\#b4x2/\#_{par}$ (Fortsetzung)

| Konfiguration | $p_{VLIW,min}$ (ns) | $N_{cand}$ (Takte) | dynamischer IPC | $t_{VLIW,workPerCand}$ ($\mu$s) | <PL_AU> | <PL_MMU> | <PL_MV> | <PL_PER> | <PL_SRU> | <PL_RFBYP> | <PL_FWD> | <PL_BUSW | <PL_FLOW | <PL_DE | <PL_RFU> | <PL_DIV1> | <PL_DIV2> | Σ Latenz |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 44 | 13,07 | 588 | 2,05 | 7,69 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 2 | 8 |
| 45 | 13,04 | 597 | 1,91 | 7,79 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 2 | 10 |
| 46 | 12,83 | 602 | 2,01 | 7,72 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 10 |
| 47 | 12,80 | 602 | 2,01 | 7,71 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 2 | 9 |
| 48 | 12,79 | 586 | 1,95 | 7,50 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 8 |
| 49 | 12,77 | 738 | 1,64 | 9,42 | 2 | 2 | 1 | 2 | 2 | 1 | 0 | 1 | 1 | 1 | 2 | 3 | 3 | 21 |
| 50 | 12,74 | 608 | 1,97 | 7,75 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 2 | 10 |
| 51 | 12,71 | 589 | 2,06 | 7,49 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 2 | 9 |
| 52 | 12,64 | 604 | 2,00 | 7,64 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 11 |
| 53 | 12,64 | 599 | 2,01 | 7,57 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | 9 |
| 54 | 12,61 | 600 | 2,02 | 7,57 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 2 | 10 |
| 55 | 12,44 | 640 | 1,89 | 7,96 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 2 | 11 |
| 56 | 12,36 | 606 | 1,98 | 7,49 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 3 | 12 |
| 57 | 12,33 | 614 | 1,97 | 7,57 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 12 |
| 58 | 12,24 | 621 | 1,95 | 7,60 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 9 |
| 59 | 11,98 | 626 | 1,94 | 7,50 | 1 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 17 |
| 60 | 11,92 | 634 | 1,90 | 7,56 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 2 | 10 |
| 61* (P) | 11,87 | 600 | 2,02 | 7,12 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 2 | 10 |
| 62* (P) | 11,73 | 607 | 1,98 | 7,12 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 2 | 11 |
| 63 | 11,59 | 748 | 1,60 | 8,67 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 2 | 15 |
| 64 | 11,53 | 725 | 1,64 | 8,36 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 2 | 13 |
| 65 | 11,48 | 754 | 1,59 | 8,66 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 2 | 16 |
| 66 | 11,39 | 741 | 1,61 | 8,44 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 2 | 15 |
| 67 | 11,32 | 731 | 1,63 | 8,28 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 2 | 14 |
| 68 | 11,28 | 745 | 1,61 | 8,41 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 16 |
| 69 | 11,23 | 733 | 1,64 | 8,23 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 2 | 15 |
| 70 | 11,22 | 717 | 1,67 | 8,05 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 10 |
| 71 | 11,08 | 755 | 1,59 | 8,36 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 17 |
| 72 | 11,02 | 736 | 1,61 | 8,11 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 2 | 14 |
| 73 | 10,90 | 730 | 1,64 | 7,96 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 2 | 14 |
| 74 | 10,76 | 1066 | 1,13 | 11,47 | 3 | 2 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 2 | 3 | 3 | 26 |
| 75 | 10,43 | 750 | 1,60 | 7,82 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 3 | 17 |
| 76 | 10,42 | 747 | 1,59 | 7,78 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 18 |
| 77 | 10,38 | 748 | 1,61 | 7,76 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 17 |
| 78 | 10,26 | 745 | 1,60 | 7,64 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 3 | 18 |
| 79 | 10,24 | 751 | 1,60 | 7,69 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 2 | 1 | 3 | 18 |
| 80 | 10,17 | 745 | 1,62 | 7,58 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 3 | 16 |
| 81 | 9,93 | 893 | 1,32 | 8,86 | 2 | 2 | 1 | 2 | 2 | 1 | 1 | 1 | 0 | 1 | 2 | 2 | 3 | 23 |
| 82 | 9,85 | 760 | 1,57 | 7,49 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 2 | 3 | 18 |
| 83 | 9,70 | 761 | 1,58 | 7,38 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 3 | 21 |
| 84 | 9,42 | 926 | 1,30 | 8,72 | 2 | 2 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 2 | 3 | 3 | 25 |
| 85 | 9,29 | 783 | 1,53 | 7,27 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 2 | 3 | 3 | 21 |
| 86* (P) | 8,99 | 762 | 1,58 | 6,85 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 2 | 2 | 3 | 20 |

**Tabelle A.8:** Pipeline-Konfigurationen für die VLIW-Prozessorkonfiguration $\#c2x2/\#_{spez}$

| Konfiguration | $p_{VLIW;min}$ (ns) | $N_{cand}$ (Takte) | dynamischer IPC | $t_{VLIW;workPerCand}$ ($\mu$s) | <PL_AU> | <PL_MMU> | <PL_MV> | <PL_PER> | <PL_SRU> | <PL_RFBYP> | <PL_FWD> | <PL_BUSW | <PL_FLOW | <PL_DE | <PL_RFU> | <PL_DIV1> | <PL_DIV2> | Σ Latenz |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 (P) | 32,30 | 375 | 2,28 | 12,11 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 |
| 2 (P) | 31,38 | 377 | 2,28 | 11,83 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 5 |
| 3 (P) | 29,53 | 371 | 2,31 | 10,96 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 4 (P) | 26,97 | 371 | 2,30 | 10,01 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 5 (P) | 26,19 | 367 | 2,31 | 9,61 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 26,02 | 374 | 2,29 | 9,73 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 3 |
| 7 | 25,98 | 418 | 1,88 | 10,86 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |
| 8 | 25,74 | 418 | 1,84 | 10,76 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 4 |
| 9 (P) | 23,91 | 375 | 2,28 | 8,96 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 4 |
| 10 (P) | 22,55 | 372 | 2,30 | 8,39 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 |
| 11 | 21,35 | 393 | 2,18 | 8,39 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 6 |
| 12 (P) | 20,40 | 370 | 2,33 | 7,55 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 3 |
| 13 (P) | 19,84 | 373 | 2,30 | 7,40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 3 |
| 14 (P) | 19,54 | 371 | 2,30 | 7,25 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 4 |
| 15 (P) | 18,48 | 376 | 2,29 | 6,95 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 6 |
| 16 (P) | 17,43 | 375 | 2,26 | 6,54 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 5 |
| 17 | 17,28 | 389 | 2,22 | 6,72 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | 7 |
| 18 | 16,96 | 420 | 2,04 | 7,12 | 1 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 3 | 13 |
| 19 | 16,95 | 452 | 1,90 | 7,66 | 1 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 2 | 3 | 3 | 17 |
| 20 | 16,61 | 398 | 2,15 | 6,61 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 2 | 8 |
| 21 (P) | 16,60 | 393 | 2,18 | 6,52 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 7 |
| 22 | 16,43 | 473 | 1,80 | 7,77 | 2 | 2 | 1 | 2 | 2 | 1 | 0 | 1 | 0 | 1 | 2 | 2 | 3 | 19 |
| 23 (P) | 16,42 | 395 | 2,18 | 6,49 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 2 | 8 |
| 24 | 16,28 | 424 | 1,84 | 6,90 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 7 |
| 25 | 16,27 | 399 | 2,16 | 6,49 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 9 |
| 26 | 16,25 | 421 | 2,02 | 6,84 | 1 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 2 | 1 | 3 | 14 |
| 27 | 16,07 | 431 | 1,99 | 6,93 | 1 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 2 | 2 | 3 | 16 |
| 28 | 15,95 | 422 | 2,03 | 6,73 | 1 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 2 | 12 |
| 29 | 15,80 | 421 | 2,04 | 6,65 | 1 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 3 | 14 |
| 30 | 15,71 | 442 | 1,74 | 6,94 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 2 | 10 |
| 31 (P) | 15,65 | 406 | 2,10 | 6,35 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 2 | 9 |
| 32 | 15,61 | 536 | 1,60 | 8,37 | 3 | 2 | 1 | 2 | 2 | 1 | 0 | 1 | 1 | 1 | 2 | 3 | 3 | 22 |
| 33 | 15,59 | 426 | 1,99 | 6,64 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 12 |
| 34 (P) | 15,58 | 400 | 2,15 | 6,23 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 9 |
| 35 | 15,51 | 405 | 2,11 | 6,28 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 10 |
| 36 | 15,45 | 413 | 2,07 | 6,38 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 11 |
| 37 (P) | 15,42 | 404 | 2,11 | 6,23 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 2 | 9 |
| 38 | 15,39 | 406 | 2,11 | 6,25 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 2 | 9 |
| 39 (P) | 15,36 | 400 | 2,15 | 6,14 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 8 |
| 40 | 15,31 | 438 | 1,94 | 6,70 | 1 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 17 |
| 41 | 15,30 | 412 | 2,08 | 6,30 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 2 | 10 |
| 42 | 15,18 | 407 | 2,11 | 6,18 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 10 |
| 43 | 15,17 | 420 | 2,03 | 6,37 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 2 | 10 |

**Tabelle A.9:** Pipeline-Konfigurationen für die VLIW-Prozessorkonfiguration $\#c2x2/\#_{spez}$ (Fortsetzung)

| Konfiguration | $p_{VLIW,min}$ (ns) | $N_{cand}$ (Takte) | dynamischer IPC | $t_{VLIW,workPerCand}$ ($\mu$s) | <PL_AU> | <PL_MMU> | <PL_MV> | <PL_PER> | <PL_SRU> | <PL_RFBYP> | <PL_FWD> | <PL_BUSW | <PL_FLOW | <PL_DE | <PL_RFU> | <PL_DIV1> | <PL_DIV2> | Σ Latenz |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 44 | 15,11 | 419 | 2,04 | 6,33 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 2 | 11 |
| 45 (P) | 15,06 | 393 | 2,18 | 5,92 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 8 |
| 46 | 15,05 | 399 | 2,14 | 6,00 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 2 | 8 |
| 47 | 14,96 | 411 | 2,08 | 6,15 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 2 | 10 |
| 48 | 14,78 | 412 | 2,08 | 6,09 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 2 | 9 |
| 49 | 14,76 | 409 | 2,09 | 6,04 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 2 | 9 |
| 50 | 14,62 | 418 | 2,05 | 6,11 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 2 | 10 |
| 51 | 14,61 | 500 | 1,70 | 7,31 | 2 | 2 | 1 | 2 | 2 | 1 | 0 | 1 | 1 | 1 | 2 | 3 | 3 | 21 |
| 52 | 14,60 | 431 | 2,00 | 6,29 | 1 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 14 |
| 53 | 14,47 | 417 | 2,03 | 6,04 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 2 | 11 |
| 54 (P) | 14,45 | 403 | 2,12 | 5,82 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | 8 |
| 55 | 14,13 | 467 | 1,82 | 6,60 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 2 | 10 |
| 56 | 14,04 | 426 | 1,73 | 5,98 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 8 |
| 57 | 14,04 | 418 | 2,03 | 5,87 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 3 | 12 |
| 58* (P) | 13,97 | 411 | 2,08 | 5,74 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | 9 |
| 59 | 13,79 | 485 | 1,72 | 6,69 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 2 | 11 |
| 60 | 13,45 | 526 | 1,61 | 7,08 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 2 | 15 |
| 61 (P) | 13,37 | 425 | 1,79 | 5,68 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 9 |
| 62 | 13,18 | 489 | 1,73 | 6,45 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 10 |
| 63 | 13,05 | 506 | 1,66 | 6,61 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 2 | 14 |
| 64 | 13,00 | 460 | 1,77 | 5,98 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 9 |
| 65 | 12,33 | 508 | 1,67 | 6,27 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 2 | 14 |
| 66 | 12,25 | 501 | 1,60 | 6,14 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 2 | 13 |
| 67 | 12,18 | 515 | 1,63 | 6,27 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 2 | 15 |
| 68 | 12,16 | 530 | 1,53 | 6,44 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 2 | 16 |
| 69 | 11,84 | 519 | 1,57 | 6,15 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 2 | 14 |
| 70 | 11,61 | 540 | 1,54 | 6,27 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 2 | 3 | 18 |
| 71 | 11,52 | 537 | 1,52 | 6,18 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 18 |
| 72 | 11,48 | 649 | 1,29 | 7,45 | 3 | 2 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 2 | 3 | 3 | 26 |
| 73 | 11,44 | 531 | 1,55 | 6,07 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 17 |
| 74 | 11,38 | 549 | 1,52 | 6,24 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 3 | 21 |
| 75 | 11,34 | 529 | 1,60 | 6,00 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 2 | 1 | 3 | 18 |
| 76 | 11,31 | 589 | 1,43 | 6,66 | 2 | 2 | 1 | 2 | 2 | 1 | 1 | 1 | 0 | 1 | 2 | 2 | 3 | 23 |
| 77 | 11,27 | 523 | 1,60 | 5,89 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 3 | 16 |
| 78 | 11,26 | 615 | 1,37 | 6,93 | 2 | 2 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 2 | 3 | 3 | 25 |
| 79 | 11,26 | 540 | 1,53 | 6,08 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 17 |
| 80 | 11,24 | 531 | 1,57 | 5,97 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 3 | 18 |
| 81 | 10,91 | 533 | 1,52 | 5,82 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 16 |
| 82 | 10,81 | 527 | 1,53 | 5,70 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 2 | 15 |
| 83 | 10,80 | 530 | 1,58 | 5,73 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 3 | 17 |
| 84* (P) | 10,02 | 542 | 1,49 | 5,43 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 2 | 2 | 3 | 20 |
| 85 | 9,95 | 563 | 1,51 | 5,60 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 2 | 3 | 3 | 21 |

**Tabelle A.10:** Pipeline-Konfigurationen (Basiskonfiguration sowie Konfigurationen mit der geringsten Rechenzeit pro Merkmalskandidat ohne und mit PL_FWD) für alle unterschiedenen VLIW-Prozessorkonfigurationen. Zudem sind die Logic Slices für VLIW, SDE und das Gesamtsystem gegeben. Das Gesamtsystem benötigt insgesamt 172 DSP-Slices (ohne TDM) und 189 BRAMs. Auf dem verwendeten Xilinx Virtex-6 FPGA (XC6VLX240T-1 / 40 nm) stehen 37.680 Logic Slices, 768 DSP-Slices und 416 BRAMs zur Verfügung. Die Ressourcen werden aus dem Report nach Technology Mapping angegeben, das Timing ergibt sich aus der statischen Timing-Analyse nach Place&Route.

| | Architektur (Pipelinekonfiguration) | PL_FWD | $p_{VLIW,min}$ (ns) | $freq_{VLIW,max}$ (MHz) | $N_{cand}$ (Takte) | Rechenzeit pro Kandidat ($\mu$s) | Logic Slices (VLIW) | Logic Slices (SDE) | Logic Slices (Gesamtsystem) |
|---|---|---|---|---|---|---|---|---|---|
| Basispipeline | #b0 (5) / $\#_{base,pipe5}$ | nein | 26,03 | 38,42 | 786 | 20,46 | 11.580 | 3.646 | 23.394 |
| | #b1 | nein | 26,26 | 38,08 | 667 | 17,53 | 11.165 | 3.473 | 22.562 |
| | #b2 | nein | 26,63 | 37,55 | 771 | 20,53 | 11.573 | 3.579 | 23.311 |
| | #b3 | nein | 25,95 | 38,54 | 652 | 16,92 | 11.029 | 3.425 | 22.444 |
| | #b4 | nein | 26,12 | 38,28 | 652 | 17,03 | 12.836 | 3.572 | 24.646 |
| | #c1 | nein | 27,25 | 36,70 | 475 | 12,94 | 11.617 | 3.398 | 22.960 |
| | #c2 | nein | 30,66 | 32,62 | 475 | 14,56 | 13.002 | 3.351 | 24.551 |
| | #b3x2 | nein | 28,52 | 35,06 | 545 | 15,54 | 12.063 | 3.494 | 23.784 |
| | #b4x2 (1) / $\#_{par,pipe1}$ | nein | 32,64 | 30,64 | 532 | 17,36 | 13.927 | 3.322 | 25.451 |
| | #c1x2 | nein | 27,61 | 36,22 | 383 | 10,57 | 12.315 | 3.469 | 23.662 |
| | #c2x2 (3) / $\#_{spez,pipe3}$ | nein | 29,53 | 33,86 | 371 | 10,96 | 13.800 | 3.491 | 25.334 |
| Beste ohne PL_FWD | #b0 (64) / $\#_{base,pipe64}$ | nein | 11,17 | 89,50 | 842 | 9,41 | 10.314 | 3.185 | 21.625 |
| | #b1 | nein | 11,24 | 88,95 | 707 | 7,95 | 10.810 | 3.388 | 22.501 |
| | #b2 | nein | 11,45 | 87,36 | 839 | 9,60 | 10.963 | 3.392 | 22.256 |
| | #b3 | nein | 11,09 | 90,20 | 704 | 7,80 | 10.536 | 3.144 | 21.448 |
| | #b4 | nein | 11,26 | 88,80 | 695 | 7,83 | 12.677 | 3.290 | 24.263 |
| | #c1 | nein | 12,65 | 79,04 | 507 | 6,41 | 11.207 | 3.359 | 22.819 |
| | #c2 | nein | 11,68 | 85,58 | 514 | 6,01 | 13.200 | 3.358 | 24.990 |
| | #b3x2 | nein | 11,95 | 83,68 | 624 | 7,46 | 11.141 | 3.376 | 22.807 |
| | #b4x2 (62) / $\#_{par,pipe62}$ | nein | 11,73 | 85,24 | 607 | 7,12 | 12.821 | 3.249 | 24.099 |
| | #c1x2 | nein | 13,78 | 72,55 | 421 | 5,80 | 12.018 | 3.434 | 23.762 |
| | #c2x2 (58) / $\#_{spez,pipe58}$ | nein | 13,97 | 71,56 | 411 | 5,74 | 13.415 | 3.174 | 24.942 |
| Beste mit PL_FWD | #b0 (86) / $\#_{base,pipe86}$ | ja | 8,75 | 114,24 | 1025 | 8,97 | 11.607 | 3.345 | 22.986 |
| | #b1 | ja | 8,43 | 118,66 | 951 | 8,01 | 11.823 | 3.359 | 23.331 |
| | #b2 | ja | 9,12 | 109,69 | 1007 | 9,18 | 10.300 | 3.055 | 20.405 |
| | #b3 | ja | 8,36 | 119,60 | 939 | 7,85 | 12.196 | 3.555 | 24.085 |
| | #b4 | ja | 11,75 | 85,11 | 687 | 8,07 | 12.849 | 3.309 | 24.285 |
| | #c1 | ja | 9,35 | 107,00 | 634 | 5,92 | 12.236 | 3.264 | 23.218 |
| | #c2 | ja | 11,81 | 84,68 | 499 | 5,89 | 13.147 | 3.424 | 24.636 |
| | #b3x2 | ja | 9,53 | 104,96 | 765 | 7,29 | 11.566 | 3.206 | 22.561 |
| | #b4x2 (86) / $\#_{par,pipe86}$ | ja | 8,99 | 111,23 | 762 | 6,85 | 13.356 | 3.318 | 24.195 |
| | #c1x2 | ja | 10,48 | 95,43 | 545 | 5,71 | 12.851 | 3.333 | 24.518 |
| | #c2x2 (84) / $\#_{spez,pipe84}$ | ja | 10,02 | 99,79 | 542 | 5,43 | 14.148 | 3.306 | 25.492 |

### A.4.3 Betriebspunkte für die dynamische Taktanpassung

Die Tabellen A.11 und A.12 zeigen die Betriebspunkte sowie die Quantisierungsbins für Merkmalskandidaten und Fahrgeschwindigkeit, die sich aus der Untersuchung in Abschnitt 5.3 ergeben. Aufgrund der Implementierung der Konfigurationen in einem einzelnen BRAM stehen genau 32 Konfigurationen zur Verfügung, von denen eine für den 5 MHz Ruhemodus reserviert ist. Es wird hier exemplarisch von einer maximalen Anzahl an Merkmalskandidaten von $n_{c,max} = 1.600$ und einer maximalen Fahrgeschwindigkeit von $v_{Auto,max} = 43,2$ km/h ausgegangen.

**Tabelle A.11:** Betriebspunkte für das Szenario $dyn_{cand,60fps}$ mit $\rho_{soll} = 1{,}2$ und $R_{soll,min} = 60$ fps

| Anzahl Merkmalskandidaten | $p_{VLIW}$ (ns) | $p_{SDE}$ (ns) | $f_{VLIW}$ (MHz) | $f_{SDE}$ (MHz) |
|---|---|---|---|---|
| <401 | 73 | 17 | 14 | 59 |
| 401 - 440 | 68 | 17 | 15 | 59 |
| 441 - 480 | 63 | 18 | 16 | 56 |
| 481 - 520 | 59 | 18 | 17 | 56 |
| 521 - 560 | 55 | 18 | 18 | 56 |
| 561 - 600 | 52 | 18 | 19 | 56 |
| 601 - 640 | 49 | 18 | 20 | 56 |
| 641 - 680 | 47 | 19 | 21 | 53 |
| 681 - 720 | 44 | 19 | 23 | 53 |
| 721 - 760 | 42 | 19 | 24 | 53 |
| 761 - 800 | 40 | 19 | 25 | 53 |
| 801 - 840 | 39 | 19 | 26 | 53 |
| 841 - 880 | 37 | 19 | 27 | 53 |
| 881 - 920 | 36 | 19 | 28 | 53 |
| 921 - 960 | 34 | 19 | 29 | 53 |
| 961 - 1000 | 33 | 19 | 30 | 53 |
| 1001 - 1040 | 32 | 19 | 31 | 53 |
| 1041 - 1080 | 31 | 20 | 32 | 50 |
| 1081 - 1120 | 30 | 20 | 33 | 50 |
| 1121 - 1160 | 29 | 20 | 34 | 50 |
| 1161 - 1200 | 28 | 20 | 36 | 50 |
| 1201 - 1240 | 27 | 20 | 37 | 50 |
| 1241 - 1280 | 26 | 19 | 38 | 53 |
| 1281 - 1320 | 25 | 19 | 40 | 53 |
| 1321 - 1360 | 25 | 20 | 40 | 50 |
| 1361 - 1400 | 24 | 20 | 42 | 50 |
| 1401 -1440 | 23 | 19 | 43 | 53 |
| 1441 - 1480 | 23 | 20 | 43 | 50 |
| 1481 - 1520 | 22 | 20 | 45 | 50 |
| 1521 - 1560 | 22 | 20 | 45 | 50 |
| >1561 | 21 | 20 | 48 | 50 |

**Tabelle A.12:** Betriebspunkte für das Szenario $dyn_{velo,cand,40cm}$ mit $\rho_{soll}$ = 1,2 und $d_{soll,max}$ = 40 cm

| Geschwindigkeit (km/h) | $R_{soll}$ (fps) | Anzahl Merkmalskandidaten | $p_{VLIW}$ (ns) | $p_{SDE}$ (ns) | $f_{VLIW}$ (MHz) | $f_{SDE}$ (MHz) |
|---|---|---|---|---|---|---|
| 36,74 - 43,20 | 60 | < 1041 | 32 | 19 | 31 | 53 |
| 36,74 - 43,20 | 60 | 1041 - 1120 | 30 | 20 | 33 | 50 |
| 36,74 - 43,20 | 60 | 1121 - 1200 | 28 | 20 | 36 | 50 |
| 36,74 - 43,20 | 60 | 1201 - 1280 | 26 | 19 | 38 | 53 |
| 36,74 - 43,20 | 60 | 1281 - 1360 | 25 | 20 | 40 | 50 |
| 36,74 - 43,20 | 60 | 1361 - 1480 | 23 | 20 | 43 | 50 |
| 36,74 - 43,20 | 60 | >1481 | 21 | 20 | 48 | 50 |
| 31,68 - 36,74 | 51 | < 1041 | 38 | 23 | 26 | 43 |
| 31,68 - 36,74 | 51 | 1041 - 1120 | 35 | 23 | 29 | 43 |
| 31,68 - 36,74 | 51 | 1121 - 1200 | 33 | 23 | 30 | 43 |
| 31,68 - 36,74 | 51 | 1201 - 1280 | 31 | 23 | 32 | 43 |
| 31,68 - 36,74 | 51 | 1281 - 1360 | 29 | 23 | 34 | 43 |
| 31,68 - 36,74 | 51 | 1361 - 1480 | 27 | 24 | 37 | 42 |
| 31,68 - 36,74 | 51 | >1481 | 25 | 24 | 40 | 42 |
| 23,04 - 31,68 | 44 | < 1041 | 44 | 27 | 23 | 37 |
| 23,04 - 31,68 | 44 | 1041 - 1120 | 41 | 27 | 24 | 37 |
| 23,04 - 31,68 | 44 | 1121 - 1200 | 38 | 27 | 26 | 37 |
| 23,04 - 31,68 | 44 | 1201 - 1280 | 36 | 27 | 28 | 37 |
| 23,04 - 31,68 | 44 | 1281 - 1360 | 34 | 27 | 29 | 37 |
| 23,04 - 31,68 | 44 | 1361 - 1480 | 31 | 27 | 32 | 37 |
| 23,04 - 31,68 | 44 | >1481 | 29 | 27 | 34 | 37 |
| < 23,04 | 32 | < 1041 | 60 | 37 | 17 | 27 |
| < 23,04 | 32 | 1041 - 1120 | 56 | 37 | 18 | 27 |
| < 23,04 | 32 | 1121 - 1200 | 53 | 38 | 19 | 26 |
| < 23,04 | 32 | 1201 - 1280 | 50 | 38 | 20 | 26 |
| < 23,04 | 32 | 1281 - 1360 | 47 | 38 | 21 | 26 |
| < 23,04 | 32 | 1361 - 1480 | 43 | 38 | 23 | 26 |
| < 23,04 | 32 | >1481 | 40 | 38 | 25 | 26 |

## A.5 Verwendete Bildsequenzen

Dieser Abschnitt zeigt exemplarisch das jeweils erste Einzelbild aus den in dieser Arbeit verwendeten Bildsequenzen. Als Benchmark wird der öffentlich zugängliche Visual Odometry Datensatz für Fahrerassistenzanwendungen aus der KITTI Vision Benchmark Suite [Gei13] verwendet. Dieser besteht aus 22 Referenzfahrten (Wohngebiet, Stadt, Autobahn), wovon bei elf auch Ground Truth Informationen der Trajektorie (rotatorische und translatorische Bewegungsparameter) sowie GPS-Positionen bekannt sind. Der Datensatz enthält Bildsequenzen, die mit einer Stereokamera bei einer Basislinie von 54 cm und einer Bildrate von zehn Bildern pro Sekunde aufgenommen wurden. Die Bilder stehen als synchronisierte, rektifizierte Stereobildpaare in 8-bit Graustufen und einer Auflösung von 1240×376 bzw. 1226×370 Pixeln zur Verfügung.

Der Datensatz wurde in einen Trainingsdatensatz *train* (Sequenz 00 bis 10, insgesamt 46.402 Einzelbilder, 44.202 Einzelbilder ohne Sequenz 01) und einen Testdatensatz *test* (Sequenz 11-21, 40.702 Einzelbilder) geteilt. Die merkmalsbasierte Eigenbewegungsschätzung in Abschnitt 3.7.2 wird auf dem Trainingsdatensatz durchgeführt, da nur dort Ground Truth Bewegungsdaten zur Verfügung stehen. Zur Modellgenerierung wird der Trainingsdatensatz ohne Sequenz 01 verwendet. Die Modellvalidierung und Evaluation der Architektur geschieht auf dem Testdatensatz und dabei insbesondere auf Sequenz 19, welche die längste Sequenz im gesamten Datensatz ist.

Informationen über Merkmalsanzahl und Fahrgeschwindigkeit der Sequenz 19 sind in Abb. A.19 gegeben.

**Abbildung A.7:** Sequenz 00 (*train*), Wohngebiet, 4.541 Stereo-Bildpaare

**Abbildung A.8:** Sequenz 01 (*train*), Autobahn, 1.101 Stereo-Bildpaare

**Abbildung A.9:** Sequenz 02 (*train*), Stadt und Wohngebiet, 4.661 Stereo-Bildpaare

**Abbildung A.10:** Sequenz 03 (*train*), Wohngebiet, 801 Stereo-Bildpaare

**Abbildung A.11:** Sequenz 04 (*train*), Stadt, 271 Stereo-Bildpaare

**Abbildung A.12:** Sequenz 05 (*train*), Wohngebiet, 2.761 Stereo-Bildpaare

**Abbildung A.13:** Sequenz 06 (*train*), Wohngebiet, 1.101 Stereo-Bildpaare


**Abbildung A.14:** Sequenz 07 (*train*), Wohngebiet, 1.101 Stereo-Bildpaare

**Abbildung A.15:** Sequenz 08 (*train*), Wohngebiet, 4.071 Stereo-Bildpaare

**Abbildung A.16:** Sequenz 09 (*train*), Wohngebiet, 1.591 Stereo-Bildpaare

**Abbildung A.17:** Sequenz 10 (*train*), Wohngebiet, 1.201 Stereo-Bildpaare

**Abbildung A.18:** Sequenz 19 (*test*), Stadt, 4.981 Stereo-Bildpaare

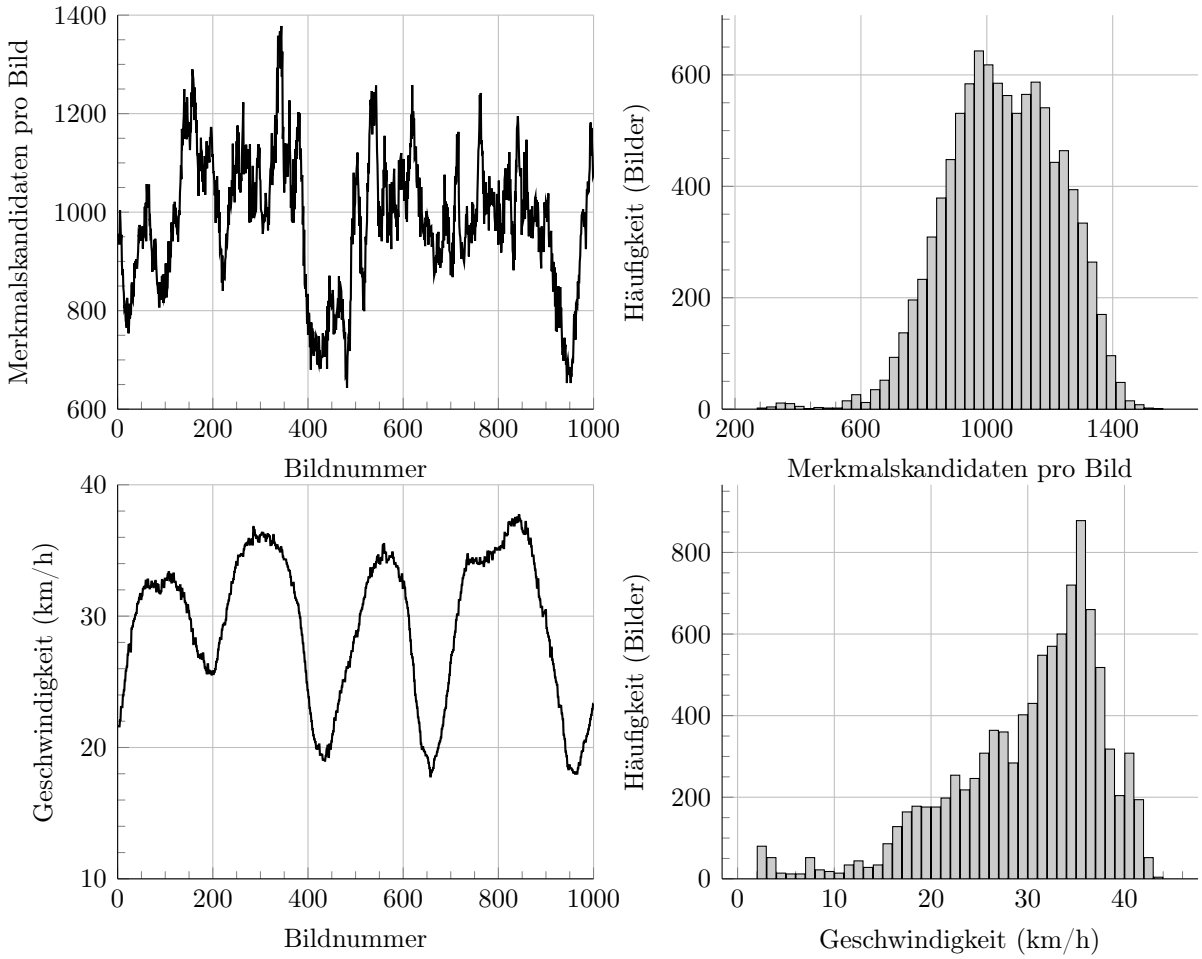



**Abbildung A.19:** Testsequenz 19 besteht aus 4.981 Stereo-Bildpaaren mit einer mittleren Anzahl an Merkmalskandidaten von 1.052 Merkmalen (Standardabweichung 178 Kandidaten). Die Abbildung zeigt die Merkmalsanzahl der ersten 1.000 Einzelbilder sowie das Histogramm über die gesamte Sequenz. Die Bild-zu-Bild Schwankung der Merkmalsanzahl wird deutlich. In dieser Sequenz wird für etwa 44% der Kandidaten auch ein Deskriptor berechnet. Darüber hinaus stellt die Abbildung die Fahrgeschwindigkeit über die ersten 1.000 Einzelbilder und als Histogramm über die gesamte Sequenz dar. Die mittlere Fahrgeschwindigkeit beträgt etwa 30 km/h (Standardabweichung 8 km/h).

# Literaturverzeichnis


[Aan12] H. Aanæs, A. L. Dahl und K. Steenstrup Pedersen. „Interesting Interest Points“. *International Journal of Computer Vision* 97 (2012). Springer, S. 18–35. DOI: 10.1007/s11263-011-0473-8 (siehe S. 12, 21).

[Alc12] P. F. Alcantarilla, A. Bartoli und A. J. Davison. „KAZE Features“. *Computer Vision – ECCV 2012*. Springer, Berlin, Heidelberg, 2012. DOI: 10.1007/978-3-642-33783-3_16 (siehe S. 14).

[Alc13] P. F. Alcantarilla und J. Nuevo. „Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces“. *Proceedings of the British Machine Vision Conference*. BMVA Press, 2013. DOI: 10.5244/C.27.13 (siehe S. 14, 15, 21).

[Awa16] A. I. Awad und M. Hassaballah. „Image Feature Detectors and Descriptors“. *Studies in Computational Intelligence* (2016). DOI: 10.1007/978-3-319-28854-3 (siehe S. 10, 12, 14, 31).

[Ban13] C. Banz. „Design and Analysis of Architectures for Stereo Vision“. Shaker Verlag. Diss. Leibniz Universität Hannover, Institut für Mikroelektronische Systeme, 2013 (siehe S. 29).

[Bay06] H. Bay, T. Tuytelaars und L. Van Gool. „SURF: Speeded up Robust Features“. *Computer Vision – ECCV 2006*. Springer, Berlin, Heidelberg, 2006, S. 404–417. DOI: 10.1007/11744023_32 (siehe S. 13, 15, 21).

[Bay08] H. Bay, A. Ess, T. Tuytelaars und L. Van Gool. „Speeded-up Robust Features (SURF)“. *Computer vision and image understanding* 110.3 (2008), S. 346–359. DOI: 10.1016/j.cviu.2007.09.014 (siehe S. 13, 15, 21).

[Bin09] „Aperture Problem“. *Encyclopedia of Neuroscience*. Hrsg. von M. D. Binder, N. Hirokawa und U. Windhorst. Springer, Berlin, Heidelberg, 2009, S. 159–159. DOI: 10.1007/978-3-540-29678-2_310 (siehe S. 12, 26).

[Blu08] H. Blume. „Modellbasierte Exploration des Entwurfsraumes für heterogene Architekturen zur digitalen Videosignalverarbeitung“. *RWTH Aachen. Habilitationsschrift* (2008) (siehe S. 2, 7).

[Bon08] V. Bonato, E. Marques und G. A. Constantinides. „A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection“. *IEEE Transactions on Circuits and Systems for Video Technology* 18.12 (2008), S. 1703–1712. DOI: 10.1109/TCSVT.2008.2004936 (siehe S. 38, 41, 44).

[Cai03] L. Cai und D. Gajski. „Transaction Level Modeling: an Overview“. *First IEEE/ACM/IFIP International Conference on Hardware/ Software Codesign and Systems Synthesis (IEEE Cat. No.03TH8721)*. 2003, S. 19–24. DOI: 10.1109/CODESS.2003.1275250 (siehe S. 8).

[Cal10] M. Calonder, V. Lepetit, C. Strecha und P. Fua. „BRIEF: Binary Robust Independent Elementary Features“. *Computer Vision – ECCV 2010*. Springer, Berlin, Heidelberg. 2010, S. 778–792. DOI: 10.1007/978-3-642-15561-1_56 (siehe S. 15, 21).

[Cal12] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha und P. Fua. „BRIEF: Computing a Local Binary Descriptor Very Fast“. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.7 (2012), S. 1281–1298. DOI: 10.1109/TPAMI.2011.222 (siehe S. 15, 21).

[Cha07] H. D. Chati, F. Muhlbauer, T. Braun, C. Bobda und K. Berns. „Hardware/-Software Co-Design of a Key Point Detector on FPGA“. *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007)*. 2007, S. 355–356. DOI: 10.1109/FCCM.2007.61 (siehe S. 38, 44).

[Cha13] L. Chang, J. Hernández-Palancar, L. E. Sucar und M. Arias-Estrada. „FPGA-based Detection of SIFT Interest Keypoints“. *Machine Vision and Applications* 24 (2013). Springer, S. 371–392. DOI: 10.1007/s00138-012-0430-8 (siehe S. 39, 44).

[Chi13] L.-C. Chiu, T.-S. Chang, J.-Y. Chen und N. Y.-C. Chang. „Fast SIFT Design for Real-Time Visual Feature Extraction“. *IEEE Transactions on Image Processing* 22.8 (2013), S. 3158–3167. DOI: 10.1109/TIP.2013.2259841 (siehe S. 39, 44).

[Chi16] H.-J. Chien, C.-C. Chuang, C.-Y. Chen und R. Klette. „When to Use What Feature? SIFT, SURF, ORB, or A-KAZE Features for Monocular Visual Odometry“. *2016 International Conference on Image and Vision Computing New Zealand (IVCNZ)*. IEEE. 2016, S. 1–6. DOI: 10.1109/IVCNZ.2016.7804434 (siehe S. 22).

[Chu06] P. P. Chu. *RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability*. Wiley-IEEE Press, 2006. ISBN: 0471720925 (siehe S. 74).

[Cvi17] I. Cvišić, J. Ćesić, I. Marković und I. Petrović. „Soft-SLAM: Computationally Efficient Stereo Visual SLAM for Autonomous Unmanned Aerial Vehicles“. *Journal of Field Robotics* 35.4 (2017). Wiley Online Library, S. 578–595. DOI: 10.1002/rob.21762 (siehe S. 79).

[Dah11] A. L. Dahl, H. Aanæs und K. S. Pedersen. „Finding the Best Feature Detector-Descriptor Combination“. *2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*. IEEE, 2011, S. 318–325. DOI: 10.1109/3DIMPVT.2011.47 (siehe S. 12).

[Das16] A. Das, G. V. Merrett und B. M. Al-Hashimi. „The slowdown or race-to-idle question: Workload-aware energy optimization of SMT multicore platforms under process variation“. *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2016, S. 535–538 (siehe S. 120).

[Den12] W. Deng, Y. Zhu, H. Feng und Z. Jiang. „An Efficient Hardware Architecture of the Optimised SIFT Descriptor Generation“. *22nd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE. 2012, S. 345–352. DOI: 10.1109/FPL.2012.6339228 (siehe S. 38, 44).

[Dha18] M. Dhana Lakshmi, P. Mirunalini, R. Priyadharsini und T. T. Mirnalinee. „Review of Feature Extraction and Matching Methods for Drone Image Stitching“. *Proceedings of the International Conference on ISMAC in Computational Vision and Bio-Engineering 2018 (ISMAC-CVB)*. Springer, Cham, 2018, S. 595–602. DOI: 10.1007/978-3-030-00665-5_59 (siehe S. 22).

[Dom20] G. Doménech-Asensi, J. Zapata-Pérez, R. Ruiz-Merino, J. A. López-Alcantud, J. Á. Díaz-Madrid, V. M. Brea und P. López. „All-Hardware SIFT Implementation for Real-Time VGA Images Feature Extraction“. *Journal of Real-Time Image Processing* 17.2 (2020). Springer, S. 371–382. DOI: 10.1007/s11554-018-0781-0 (siehe S. 41, 44, 141).

[Don20] X. Dong, X. Dong, J. Dong und H. Zhou. „Monocular Visual-IMU Odometry: A Comparative Evaluation of Detector-Descriptor-Based Methods“. *IEEE Transactions on Intelligent Transportation Systems* 21.6 (2020), S. 2471–2484. DOI: 10.1109/TITS.2019.2919003 (siehe S. 22).

[Eec10] L. Eeckhout. „Computer Architecture Performance Evaluation Methods“. *Synthesis Lectures on Computer Architecture* 5.1 (2010). Morgan & Claypool Publishers, S. 1–145. DOI: 10.2200/S00273ED1V01Y201006CAC010 (siehe S. 8).

[Fan19] B. Fan, Q. Kong, X. Wang, Z. Wanga, S. Xiang, C. Pan und P. Fua. „A Performance Evaluation of Local Features for Image-Based 3D Reconstruction“. *IEEE Transactions on Image Processing* 28.10 (2019), S. 4774–4789. DOI: 10.1109/TIP.2019.2909640 (siehe S. 12, 14, 22).

[Fej19] A. Fejér, Z. Nagy, J. Benois-Pineau, P. Szolgay, A. de Rugy und J. Domenger. „FPGA-based SIFT Implementation for Wearable Computing“. *2019 IEEE 22nd International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*. 2019, S. 1–4. DOI: 10.1109/DDECS.2019.8724653 (siehe S. 41, 44).

[Fis79] J. A. Fisher. *Optimization of Horizontal Microcode Within and Beyond Basic Blocks: an Application of Processor Scheduling with Resources*. Techn. Ber. New York Univ., NY (USA). Courant Mathematics und Computing Lab., 1979. DOI: 10.2172/5752434 (siehe S. 57).

[Fis81] M. A. Fischler und R. C. Bolles. „Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". *Commun. ACM* 24.6 (1981), S. 381–395. DOI: 10.1145/358669.358692 (siehe S. 20).

[Gau11] S. Gauglitz, T. Höllerer und M. Turk. „Evaluation of Interest Point Detectors and Feature Descriptors for Visual Tracking". *International Journal of Computer Vision* 94.3 (2011). Springer, S. 335–360. DOI: 10.1007/s11263-011-0431-5 (siehe S. 18, 21, 22).

[Gei11] A. Geiger, J. Ziegler und C. Stiller. „StereoScan: Dense 3D Reconstruction in Real-Time". *2011 IEEE Intelligent Vehicles Symposium (IV)*. 2011, S. 963–968. DOI: 10.1109/IVS.2011.5940405 (siehe S. 3, 11, 19, 79).

[Gei12] A. Geiger, P. Lenz und R. Urtasun. „Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite". *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, S. 3354–3361. DOI: 10.1109/CVPR.2012.6248074 (siehe S. 20, 80).

[Gei13] A. Geiger, P. Lenz, C. Stiller und R. Urtasun. „Vision Meets Robotics: The KITTI Dataset". *The International Journal of Robotics Research* 32.11 (2013). Sage Publications Sage UK: London, England, S. 1231–1237. DOI: 10.1177/0278364913491297 (siehe S. 11, 79, 84, 147, 164).

[Gie17] F. Giesemann, G. Payá Vayá, L. Gerlach, H. Blume, F. Pflug und G. von Voigt. „Using a Genetic Algorithm Approach to Reduce Register File Pressure During Instruction Scheduling". *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. 2017, S. 179–187. DOI: 10.1109/SAMOS.2017.8344626 (siehe S. 57, 59, 86).

[Gie20] F. Giesemann, L. Gerlach und G. Payá Vayá. „Evolutionary Algorithms for Instruction Scheduling, Operation Merging, and Register Allocation in VLIW Compilers". *Journal of Signal Processing Systems* 92.7 (2020). Springer, S. 655–678. DOI: 10.1007/s11265-019-01493-2 (siehe S. 57).

[Gri04] M. Gries. „Methods for Evaluating and Covering the Design Space During Early Design Development". *Integration, the VLSI Journal* 38.2 (2004). Elsevier, S. 131–183. DOI: 10.1016/j.vlsi.2004.06.001 (siehe S. 5–9).

[Har03] R. Hartley und A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003 (siehe S. 11, 17, 19).

[Har17b] J. Hartig, G. Payá Vayá, N. Mentzer und H. Blume. „Balanced Application-Specific Processor System for Efficient SIFT-Feature Detection (Stamatis Vassiliadis Best Paper Award)“. *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE. 2017, S. 78–87. DOI: 10.1109/SAMOS.2017.8344614 (siehe S. 48).

[Har88] C. Harris und M. Stephens. „A Combined Corner and Edge Detector“. *In Proc. of Fourth Alvey Vision Conference*. Citeseer. 1988, S. 147–151 (siehe S. 13, 14, 21, 26).

[Hei09] M. Heikkilä, M. Pietikäinen und C. Schmid. „Description of Interest Regions with Local Binary Patterns“. *Pattern Recognition* 42.3 (2009). Elsevier, S. 425–436. DOI: 10.1016/j.patcog.2008.08.014 (siehe S. 14).

[Hei12] J. Heinly, E. Dunn und J.-M. Frahm. „Comparative Evaluation of Binary Features“. *Lecture Notes in Computer Science, Computer Vision – ECCV 2012* (2012). Springer Berlin Heidelberg, S. 759–773. DOI: 10.1007/978-3-642-33709-3_54 (siehe S. 22).

[Hen83] J. Hennessy und T. Gross. „Postpass Code Optimization of Pipeline Constraints“. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 5.3 (1983). ACM New York, NY, USA, S. 422–448. DOI: 10.1145/2166.357217 (siehe S. 57).

[Hes10] R. Hess. „An Open-source SIFTLibrary“. *Proceedings of the 18th ACM International Conference on Multimedia*. ACM, 2010, S. 1493–1496. DOI: 10.1145/1873951.1874256 (siehe S. 27).

[Hou13] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing und C. Igel. „Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark“. *The 2013 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2013, S. 1–8. DOI: 10.1109/IJCNN.2013.6706807 (siehe S. 45).

[Hua12] F.-C. Huang, S.-Y. Huang, J.-W. Ker und Y.-C. Chen. „High-Performance SIFT Hardware Accelerator for Real-Time Image Feature Extraction“. *IEEE Transactions on Circuits and Systems for Video Technology* 22.3 (2012), S. 340–351. DOI: 10.1109/TCSVT.2011.2162760 (siehe S. 30, 38, 44).

[Jan17] J. Janai, F. Güney, A. Behl und A. Geiger. „Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art“. *eprint arXiv:1704.05519* (2017) (siehe S. 2).

[Jia14] J. Jiang, X. Li und G. Zhang. „SIFT Hardware Implementation for Real-Time Image Feature Extraction“. *IEEE Transactions on Circuits and Systems for Video Technology* 24.7 (2014), S. 1209–1220. DOI: 10.1109/TCSVT.2014.2302535 (siehe S. 39, 44, 141).

[Kal17] L. Kalms, K. Mohamed und D. Göhringer. „Accelerated Embedded AKAZE Feature Detection Algorithm on FPGA“. *Proceedings of the 8th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*. 2017, S. 1–6. DOI: 10.1145/3120895.3120898 (siehe S. 14).

[Keu00] K. Keutzer, A. R. Newton, J. M. Rabaey und A. Sangiovanni-Vincentelli. „System-level Design: Orthogonalization of Concerns and Platform-based Design“. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19.12 (2000), S. 1523–1543. DOI: 10.1109/43.898830 (siehe S. 7).

[Kha15] N. Khan, B. McCane und S. Mills. „Better than SIFT?“ *Machine Vision and Applications* 26.6 (2015). Springer, S. 819–836. DOI: 10.1007/s00138-015-0689-7 (siehe S. 22).

[Kie97] B. Kienhuis, E. Deprettere, K. Vissers und P. Van Der Wolf. „An Approach for Quantitative Analysis of Application-specific Dataflow Architectures“. *Proceedings IEEE International Conference on Application-Specific Systems, Architectures and Processors*. 1997, S. 338–349. DOI: 10.1109/ASAP.1997.606839 (siehe S. 7).

[Kim07] D. Kim, K. Kim, J. Kim, S. Lee und H. Yoo. „An 81.6 GOPS Object Recognition Processor Based on NoC and Visual Image Processing Memory“. *2007 IEEE Custom Integrated Circuits Conference*. 2007, S. 443–446. DOI: 10.1109/CICC.2007.4405769 (siehe S. 37, 44).

[Kim13] E. S. Kim und H.-J. Lee. „A Novel Hardware Design for SIFT Generation with Reduced Memory Requirement“. *Journal of Semiconductor Technology and Science* 13.2 (2013), S. 157–169 (siehe S. 39, 44).

[Kit19] Kitti Vision Benchmark Suite. *Visual Odometry / SLAM Evaluation 2012*. http://www.cvlibs.net/datasets/kitti/eval_odometry.php. [Online; besucht am 02.09.2020]. 2019 (siehe S. 79).

[Koc14] M. Kock, S. Hesselbarth, M. Pfitzner und H. Blume. „Hardware-accelerated Design Space Exploration Framework for Communication Systems“. *Analog Integrated Circuits and Signal Processing* 78.3 (2014). Springer, S. 557–571. DOI: 10.1007/s10470-013-0127-6 (siehe S. 47).

[Koe84] J. J. Koenderink. „The Structure of Images“. *Biological Cybernetics* 50.5 (1984). Springer, S. 363–370. DOI: 10.1007/BF00336961 (siehe S. 13).

[Küm09] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss und A. Kleiner. „On Measuring the Accuracy of SLAM Algorithms“. *Autonomous Robots* 27.4 (2009). Springer, S. 387. DOI: 10.1007/s10514-009-9155-6 (siehe S. 20).

[Lag98] J. C. Lagarias, J. A. Reeds, M. H. Wright und P. E. Wright. „Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions“. *SIAM Journal on Optimization* 9.1 (1998). SIAM, S. 112–147. DOI: 10.1137/S1052623496303470 (siehe S. 104).

[Len16] G. Lentaris, I. Stamoulias, D. Soudris und M. Lourakis. „HW/SW Codesign and FPGA Acceleration of Visual Odometry Algorithms for Rover Navigation on Mars“. *IEEE Transactions on Circuits and Systems for Video Technology* 26.8 (2016), S. 1563–1577. DOI: 10.1109/TCSVT.2015.2452781 (siehe S. 39).

[Leu11] S. Leutenegger, M. Chli und R. Y. Siegwart. „BRISK: Binary Robust Invariant Scalable Keypoints“. *Computer Vision (ICCV), 2011 International Conference on*. IEEE. 2011, S. 2548–2555. DOI: 10.1109/ICCV.2011.6126542 (siehe S. 14, 15, 21).

[Li08] J. Li und N. M. Allinson. „A Comprehensive Review of Current Local Features for Computer Vision“. *Neurocomputing* 71.10-12 (2008). Elsevier, S. 1771–1787. DOI: 10.1016/j.neucom.2007.11.032 (siehe S. 12, 22).

[Lin10] Y.-M. Lin, C.-H. Yeh, S.-H. Yen, C.-H. Ma, P.-Y. Chen und C.-C. J. Kuo. „Efficient VLSI Design for SIFT Feature Description“. *2010 International Symposium on Next Generation Electronics*. IEEE. 2010, S. 48–51. DOI: 10.1109/ISNE.2010.5669202 (siehe S. 38, 44).

[Lin94] T. Lindeberg. *Scale-space Theory in Computer Vision*. Bd. 256. Springer, Boston, MA, 1994. DOI: 10.1007/978-1-4757-6465-9 (siehe S. 13, 25).

[Low04a] D. G. Lowe. *Method and Apparatus for Identifying Scale Invariant Features in an Image and Use of Same for Locating an Object in an Image*. US Patent 6,711,293. 2004 (siehe S. 23).

[Low04b] D. G. Lowe. „Distinctive Image Features from Scale-Invariant Keypoints“. *International Journal of Computer Vision* 60.2 (2004). Springer, S. 91–110. DOI: 10.1023/B:VISI.0000029664.99615.94 (siehe S. 11, 13, 14, 21, 23–27, 29, 34, 75).

[Low99] D. G. Lowe. „Object Recognition from Local Scale-invariant Features“. *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Bd. 2. 1999, S. 1150–1157. DOI: 10.1109/ICCV.1999.790410 (siehe S. 13, 21, 23).

[Mat04] J. Matas, O. Chum, M. Urban und T. Pajdla. „Robust Wide-baseline Stereo from Maximally Stable Extremal Regions“. *Image and Vision Computing* 22.10 (2004). Elsevier, S. 761–767. DOI: https://doi.org/10.1016/j.imavis.2004.02.006 (siehe S. 12).

[Men16] N. Mentzer, G. Payá-Vayá und H. Blume. „Analyzing the Performance-Hardware Trade-off of an ASIP-based SIFT Feature Extraction“. *Journal of Signal Processing Systems* 85.1 (2016). Springer, S. 83–99. DOI: 10.1007/s11265-015-0986-4 (siehe S. 30, 44).

[Men18] N. Mentzer. „Applikationsspezifische Prozessoren zur Punktkorrespondenzsuche in der Stereo-Bildverarbeitung für Automotive Anwendungen“. Dr. Hut. Diss. Leibniz Universität Hannover, Institut für Mikroelektronische Systeme, 2018 (siehe S. 21, 34, 75).

[Mik05a] K. Mikolajczyk. *Affine Covariant Regions*. http://www.robots.ox.ac.uk/~vgg/research/affine/index.html. [Online; besucht am 02.09.2020]. 2005 (siehe S. 22, 75, 77, 78).

[Mik05b] K. Mikolajczyk und C. Schmid. „A Performance Evaluation of Local Descriptors“. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.10 (2005), S. 1615–1630. DOI: 10.1109/TPAMI.2005.188 (siehe S. 14, 18, 21, 22).

[Mik05c] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir und L. V. Gool. „A Comparison of Affine Region Detectors“. *International Journal of Computer Vision* 65.1 (2005). Springer, S. 43–72. DOI: 10.1007/s11263-005-3848-x (siehe S. 11, 12).

[Mit06] S. K. Mitra und Y. Kuo. *Digital Signal Processing: a Computer-based Approach*. Bd. 2. McGraw-Hill New York, 2006 (siehe S. 34).

[Miz10] K. Mizuno, H. Noguchi, G. He, Y. Terachi, T. Kamino, H. Kawaguchi und M. Yoshimoto. „Fast and Low-Memory-Bandwidth Architecture of SIFT Descriptor Generation with Scalability on Speed and Accuracy for VGA Video“. *2010 International Conference on Field Programmable Logic and Applications*. 2010, S. 608–611. DOI: 10.1109/FPL.2010.119 (siehe S. 38, 44).

[Miz11] K. Mizuno, H. Noguchi, G. He, Y. Terachi, T. Kamino, T. Fujinaga, S. Izumi, Y. Ariki, H. Kawaguchi und M. Yoshimoto. „A Low-power Real-Time SIFT Descriptor Generation Engine for Full-HDTV Video Recognition“. *IEICE Transactions on Electronics* E94.C.4 (2011), S. 448–457. DOI: 10.1587/transele.E94.C.448 (siehe S. 38, 44).

[Mor77] H. P. Moravec. „Towards Automatic Visual Obstacle Avoidance“. *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI) - Volume 2*. 1977, S. 584–584. URL: http://dl.acm.org/citation.cfm?id=1622943.1622947 (siehe S. 12).

[Nik14] F. Nikolaus. „Implementierung und Evaluation einer FPGA-basierten heterogenen ASIP-Architektur für die Extraktion von SIFT-Merkmalen“. *Masterarbeit, Institut für Mikroelektronische Systeme, Leibniz Universität Hannover*. Nov. 2014 (siehe S. 33–35, 45).

[Nol04] T. Noll. „Application Domain Specific Embedded FPGAs for SoC Platforms“. *Irish Signals and Systems Conference 2004 (ISSC'04) - eingeladener Übersichtsvortrag*. Juni 2004, S. 3–3. DOI: 10.1049/cp:20040506 (siehe S. 2).

[Ope15] OpenCV. *Open Source Computer Vision Library 3.0*. https://opencv.org/opencv-3-0/. [Online; besucht am 02.09.2020]. 2015 (siehe S. 75, 79).

[Pal13] H. Palm, J. Holzmann, S.-A. Schneider und H.-M. Koegeler. „Die Zukunft im Fahrzeugentwurf Systems-Engineering-basierte Optimierung“. *ATZ - Automobiltechnische Zeitschrift* 115.6 (2013). Springer, S. 512–517. DOI: 10.1007/s35148-013-0142-z (siehe S. 5).

[Pal15] C. Pal, P. Das, S. B. Mandal, A. Chakrabarti, S. Basu und R. Ghosh. „An Efficient Hardware Design of SIFT Algorithm Using Fault Tolerant Reversible Logic“. *2015 IEEE 2nd International Conference on Recent Trends in Information Systems (ReTIS)*. 2015, S. 514–519. DOI: 10.1109/ReTIS.2015.7232933 (siehe S. 40, 44).

[Pay07] G. Payá Vayá, J. Martín-Langerwerf, P. Taptimthong und P. Pirsch. „Design Space Exploration of Media Processors: A Parameterized Scheduler“. *2007 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. 2007, S. 41–49. DOI: 10.1109/ICSAMOS.2007.4285732 (siehe S. 57).

[Pay09] G. Payá Vayá, J. Martín-Langerwerf, F. Giesemann, H. Blume und P. Pirsch. „Instruction Merging to Increase Parallelism in VLIW Architectures“. *2009 International Symposium on System-on-Chip*. 2009, S. 143–146. DOI: 10.1109/SOCC.2009.5335660 (siehe S. 55).

[Pay11] G. Payá Vayá. „Design and Analysis of a Generic VLIW Processor for Multimedia Applications“. Shaker Verlag. Diss. Leibniz Universität Hannover, Institut für Mikroelektronische Systeme, 2011 (siehe S. 55).

[Pay12] G. Payá Vayá, R. Burg und H. Blume. „Dynamic Data-path Self-reconfiguration of a VLIW-SIMD Soft-Processor Architecture“. *Workshop on Self-Awareness in Reconfigurable Computing Systems (SRCS)*. 2012, S. 26 (siehe S. 45, 47, 55, 56).

[Pen16] J. Q. Peng, Y. H. Liu, C. Y. Lyu, Y. H. Li, W. G. Zhou und K. Fan. „FPGA-based Parallel Hardware Architecture for SIFT Algorithm“. *2016 IEEE International Conference on Real-Time Computing and Robotics (RCAR)*. 2016, S. 277–282. DOI: 10.1109/RCAR.2016.7784039 (siehe S. 40, 44).

[Pet05] N. Pettersson und L. Petersson. „Online Stereo Calibration using FPGAs“. *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.* 2005, S. 55–60. DOI: 10.1109/IVS.2005.1505077 (siehe S. 37, 44).

[Pim17] A. D. Pimentel. „Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration“. *IEEE Design & Test* 34.1 (2017), S. 77–90. DOI: 10.1109/MDAT.2016.2626445 (siehe S. 5–9).

[Pir98] P. Pirsch. *Architectures for Digital Signal Processing*. New York, NY, USA: John Wiley & Sons, Inc., 1998. ISBN: 978-0-471-97145-0 (siehe S. 34, 62).

[Pro15] H. Proença. „Performance Evaluation of Keypoint Detection and Matching Techniques on Grayscale Data“. *Signal, Image and Video Processing* 9.5 (2015). Springer, S. 1009–1019. DOI: 10.1007/s11760-013-0535-1 (siehe S. 22).

[Qas15] M. Qasaimeh, A. Sagahyroon und T. Shanableh. „FPGA-Based Parallel Hardware Architecture for Real-Time Image Classification“. *IEEE Transactions on Computational Imaging* 1.1 (2015), S. 56–70. DOI: 10.1109/TCI.2015.2424077 (siehe S. 40, 44).

[Qiu10] J. Qiu, T. Huang und T. Ikenaga. „A 7-Round Parallel Hardware-Saving Accelerator for Gaussian and DoG Pyramid Construction Part of SIFT“. *Computer Vision – ACCV 2009*. Springer Berlin Heidelberg, 2010, S. 75–84. DOI: 10.1007/978-3-642-12297-2_8 (siehe S. 38).

[Qt20] Qt. *Cross-platform Software Development for Embedded and Desktop*. https://www.qt.io/. [Online; besucht am 02.09.2020]. 2020 (siehe S. 152).

[Ros06] E. Rosten und T. Drummond. „Machine Learning for High-Speed Corner Detection“. *Computer Vision – ECCV 2006*. Springer Berlin Heidelberg, 2006, S. 430–443. DOI: 10.1007/11744023_34 (siehe S. 13, 18, 21).

[Ros10] E. Rosten, R. Porter und T. Drummond. „Faster and Better: A Machine Learning Approach to Corner Detection“. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.1 (2010), S. 105–119. DOI: 10.1109/TPAMI.2008.275 (siehe S. 11, 13, 21).

[Rub11] E. Rublee, V. Rabaud, K. Konolige und G. Bradski. „ORB: An Efficient Alternative to SIFT or SURF“. *2011 International Conference on Computer Vision (ICCV)*. 2011, S. 2564–2571. DOI: 10.1109/ICCV.2011.6126544 (siehe S. 14, 15, 21).

[Rub18] P. Rubio-Ibáñez, R. Ruiz-Merino, G. Doménech-Asensi, J. J. Martínez-Álvarez, J. Zapata-Pérez, J. Á. Díaz-Madrid und J. A. López-Alcantud. „An All-Hardware Implementation of the Subpixel Refinement Stage in SIFT Algorithm“. *International Journal of Circuit Theory and Applications* 46.9 (2018). Wiley Online Library, S. 1690–1702. DOI: 10.1002/cta.2482 (siehe S. 41, 45).

[Sch00] C. Schmid, R. Mohr und C. Bauckhage. „Evaluation of Interest Point Detectors“. *International Journal of Computer Vision* 37.2 (2000). Springer, S. 151–172. DOI: 10.1023/A:1008199403446 (siehe S. 17, 18).

[Se04] S. Se, H.-K. Ng, P. Jasiobedzki und T.-J. Moyung. „Vision-based Modeling and Localization for Planetary Exploration Rovers“. *55th International Astronautical Congress of the International Astronautical Federation, the International Academy of Astronautics, and the International Institute of Space Law*. 2004, S. 434–440. DOI: 10.2514/6.IAC-04-U.2.09 (siehe S. 37, 44).

[Suz12] T. Suzuki und T. Ikenaga. „SIFT-based Low Complexity Keypoint Extraction and its Real-Time Hardware Implementation for full-HD Video“. *Proceedings of The 2012 Asia Pacific Signal and Information Processing Association Annual Summit and Conference*. IEEE. 2012, S. 1–6 (siehe S. 39, 44).

[Sze11] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer London, UK, 2011 (siehe S. 3, 10, 12, 14, 16, 17, 19, 39).

[Tar18] S. A. K. Tareen und Z. Saleem. „A Comparative Analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK“. *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*. IEEE. 2018, S. 1–10. DOI: 10.1109/ICOMET.2018.8346440 (siehe S. 22).

[Tex06] Texas Instruments. *USB Interface Adapter Evaluation Module*. http://www.ti.com/lit/ml/sllu093/sllu093.pdf. [Online; besucht am 02.09.2020]. 2006 (siehe S. 103).

[Tex08] Texas Instruments. *Digital PWM System Controller*. http://www.ti.com/lit/ds/slus766c/slus766c.pdf. [Online; besucht am 02.09.2020]. 2008 (siehe S. 103, 104, 107).

[Tuy08] T. Tuytelaars, K. Mikolajczyk u. a. „Local Invariant Feature Detectors: a Survey“. *Foundations and Trends® in Computer Graphics and Vision* 3.3 (2008). Now Publishers, Inc., S. 177–280. DOI: 10.1561/0600000017 (siehe S. 10, 12, 21).

[Vee17] H. J. Veendrick. *Nanometer CMOS ICs*. Springer, 2017. DOI: 10.1007/978-3-319-47597-4 (siehe S. 103).

[Vou16] J. Vourvoulakis, J. Kalomiros und J. Lygouras. „Fully Pipelined FPGA-based Architecture for Real-Time SIFT Extraction“. *Microprocessors and Microsystems* 40 (2016). Elsevier, S. 53–73. DOI: 10.1016/j.micpro.2015.11.013 (siehe S. 40–42, 44, 46, 141).

[Wan14] J. Wang, S. Zhong, L. Yan und Z. Cao. „An Embedded System-on-Chip Architecture for Real-Time Visual Detection and Matching“. *IEEE Transactions on Circuits and Systems for Video Technology* 24.3 (2014), S. 525–538. DOI: 10.1109/TCSVT.2013.2280040 (siehe S. 40, 44).

[Xie17] S. Xie, M. Lin, M. Hang und R. Ding. „An Adaptive Strategy For Monocular Visual Odometry“. *ICRAI 2017: Proceedings of the 2017 International Conference on Robotics and Artificial Intelligence*. ACM. New York, NY, USA, 2017, S. 46–50. DOI: 10.1145/3175603.3175619 (siehe S. 22).

[Xil10] Xilinx Inc. *MMCM Dynamic Reconfiguration*. https://www.xilinx.com/support/documentation/application_notes/xapp878.pdf. [Online; besucht am 02.09.2020]. 2010 (siehe S. 47, 73).

[Xil11a] Xilinx Inc. *Virtex-6 FPGA DSP48E1 Slice*. https://www.xilinx.com/support/documentation/user_guides/ug369.pdf. [Online; besucht am 02.09.2020]. 2011 (siehe S. 34, 49, 131).

[Xil11b] Xilinx Inc. *Virtex-6 FPGA ML605 Evaluation Kit*. https://www.xilinx.com/products/boards-and-kits/ek-v6-ml605-g.html. [Online; besucht am 02.09.2020]. 2011 (siehe S. 47, 103).

[Xil13] Xilinx Inc. *Xilinx Command Line Tools User Guide*. https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/devref.pdf. [Online; besucht am 02.09.2020]. 2013 (siehe S. 113).

[Xil14a] Xilinx Inc. *Virtex-6 FPGA Clocking Resources*. https://www.xilinx.com/support/documentation/user_guides/ug362.pdf. [Online; besucht am 02.09.2020]. 2014 (siehe S. 72).

[Xil14b] Xilinx Inc. *Virtex-6 FPGA DC and Switching Characteristics*. https://www.xilinx.com/support/documentation/data_sheets/ds152.pdf. [Online; besucht am 02.09.2020]. 2014 (siehe S. 104).

[Xil14c] Xilinx Inc. *Virtex-6 FPGA Memory Resources*. https://www.xilinx.com/support/documentation/user_guides/ug363.pdf. [Online; besucht am 02.09.2020]. 2014 (siehe S. 74).

[Xil15] Xilinx Inc. *Virtex-6 FPGA Family Overview*. https://www.xilinx.com/support/documentation/data_sheets/ds150.pdf. [Online; besucht am 02.09.2020]. 2015 (siehe S. 37, 128).

[Yao09] L. Yao, H. Feng, Y. Zhu, Z. Jiang, D. Zhao und W. Feng. „An Architecture of Optimised SIFT Feature Detection for an FPGA Implementation of an Image Matcher“. *2009 International Conference on Field-Programmable Technology*. IEEE. 2009, S. 30–37. DOI: 10.1109/FPT.2009.5377651 (siehe S. 38, 44).

[Yum16] J. Yum, C. Lee, J. Kim und H. Lee. „A Novel Hardware Architecture With Reduced Internal Memory for Real-Time Extraction of SIFT in an HD Video“. *IEEE Transactions on Circuits and Systems for Video Technology* 26.10 (2016), S. 1943–1954. DOI: 10.1109/TCSVT.2015.2489458 (siehe S. 40, 44).

[Zho13] S. Zhong, J. Wang, L. Yan, L. Kang und Z. Cao. „A Real-Time Embedded Architecture for SIFT“. *Journal of Systems Architecture* 59.1 (2013). Elsevier, S. 16–29. DOI: 10.1016/j.sysarc.2012.09.002 (siehe S. 39, 44).

[Zho16] Z. Zhou, R. Ying, R. Sun, Z. Wei, K. Jin und P. Liu. „A Hardware Accelerated Scale Invariant Feature Detector for Real-Time Visual Localization and Mapping“. *2016 Fourth International Conference on Ubiquitous Positioning, Indoor Navigation and Location Based Services (UPINLBS)*. IEEE. 2016, S. 162–169. DOI: 10.1109/UPINLBS.2016.7809965 (siehe S. 40, 44).

[Zim08] H.-J. Zimmermann. *Operations Research: Methoden und Modelle. Für Wirtschaftsingenieure, Betriebswirte, Informatiker*. Vieweg+Teubner Verlag, 2008. DOI: 10.1007/978-3-8348-9461-8 (siehe S. 89, 91).

## Veröffentlichungen des Autors

[Har13] J. Hartig, G. Payá Vayá und H. Blume. „Design and Analysis of a Structured-ASIC Architecture for Implementing Generic VLIW-SIMD Processors“. *ICT. OPEN* (2013).

[Har14] J. Hartig, L. Gerlach, G. Payá Vayá und H. Blume. „Customizing a VLIW-SIMD Application-Specific Instruction-Set Processor for Hearing Aid Devices“. *2014 IEEE Workshop on Signal Processing Systems (SiPS)*. 2014, S. 1–6. DOI: 10.1109/SiPS.2014.6986072.

[Har17a] J. Hartig, G. Payá Vayá, H. Heymann und H. Blume. „Tool-Supported Design Space Exploration of a Processor System for SIFT-Feature Detection“. *2017 IEEE 7th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. IEEE. 2017, S. 168–169. DOI: 10.1109/ICCE-Berlin.2017.8210619.

[Har17b] J. Hartig, G. Payá Vayá, N. Mentzer und H. Blume. „Balanced Application-Specific Processor System for Efficient SIFT-Feature Detection (Stamatis Vassiliadis Best Paper Award)“. *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE. 2017, S. 78–87. DOI: 10.1109/SAMOS.2017.8344614 (siehe S. 48).

[Nol17] S. Nolting, F. Giesemann, J. Hartig, A. Schmider und G. Payá Vayá. „Application-Specific Soft-Core Vector Processor for Advanced Driver Assistance Systems“. *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE. 2017, S. 1–2. DOI: 10.23919/FPL.2017.8056836.

[Now15] R. Nowosielski, J. Hartig, G. Payá Vayá, H. Blume und A. Garcıa-Ortiz. „Exploring Different Approximate Adder Architecture Implementations in a 250◦C SOI Technology“. *1st Workshop On Approximate Computing (WAPCO), HiPEAC*. Citeseer. 2015.

# Lebenslauf

## Zur Person

**Julian Hartig**
geboren am 26. Juni 1989 in Hannover, Deutschland

## Ausbildung

| | |
|---|---|
| 06/2008 | **Gymnasium Lutherschule Hannover**<br>Abschluss mit Abitur (Mathematik, Physik, Chemie, Politik, Englisch) |
| 10/2008-11/2013 | **Studium der Elektrotechnik**<br>Leibniz Universität Hannover<br>Abschluss Diplom-Ingenieur |

## Berufliche Laufbahn

| | |
|---|---|
| 2009-2010 | **Studentische Hilfskraft**<br>Institut für Grundlagen der Elektrotechnik und Messtechnik<br>Leibniz Universität Hannover |
| 2011-2012 | **Studentische Hilfskraft**<br>Institut für Mikroelektronische Systeme<br>Fachgebiet Architekturen und Systeme<br>Leibniz Universität Hannover |
| 12/2013-02/2020 | **Wissenschaftlicher Mitarbeiter**<br>Institut für Mikroelektronische Systeme<br>Fachgebiet Architekturen und Systeme<br>Leibniz Universität Hannover<br>(unterstützt durch ein Stipendium der Bosch-Forschungsstiftung) |
| seit 05/2020 | **Entwicklungsingenieur**<br>Dream Chip Technologies GmbH, Garbsen, Deutschland |