TECHNISCHE UNIVERSITÄT DRESDEN
FAKULTÄT INFORMATIK

Institut für Technische Informatik (TeI)
Professur für VLSI-Entwurfssysteme, Diagnostik und Architektur

und

FRAUNHOFER INSTITUT FÜR INTEGRIERTE SCHALTUNGEN (IIS)

Institutsteil Entwicklung Adaptiver Systeme (EAS)

Dissertation

Effizienter Einsatz von Bildsensoren mit integrierter Signalverarbeitung

Vorgelegt von:	Dipl.-Ing. Peter Reichel		
Geboren am:	31.07.1983	in:	Freital

zum
Erlangen des akademischen Grades

Doktor-Ingenieur
(Dr.-Ing.)

Betreuer:	Prof. Dr.-Ing. habil. Rainer G. Spallek
Fachreferent:	Dr.-Ing. Jens Döge
Vorgelegt am:	18.11.2016

Die vorliegende Arbeit wurde am 18. November 2016 unter dem Titel „Effizienter Einsatz von Bildsensoren mit integrierter Signalverarbeitung“ als Dissertation an der Fakultät Informatik der Technischen Universität Dresden zur Erlangung des akademischen Grades „Doktoringenieur“ eingereicht und am 9. August 2017 verteidigt.

Promotionskommission:

Vorsitzender:	Prof. Dr. rer. nat. habil. Gerhard Weber
1. Gutachter:	Prof. Dr.-Ing. habil. Rainer G. Spallek
2. Gutachter:	Univ.-Prof. Dr. rer. nat. Gunther Notni
Fachreferent:	Dr.-Ing. Jens Döge
weiteres Mitglied:	Prof. Dr.-Ing. habil. Martin Wollschlaeger

„Wer hohe Türme bauen will, muss lange beim Fundament verweilen.“

Anton Bruckner (1824 - 1896), österreichischer Komponist der Romantik

Kurzfassung

Bildsensoren mit integrierter Signalverarbeitung – sog. *„Vision Chips“* – ermöglichen die Ausführung ansonsten rechenintensiver Verarbeitungsschritte während oder unmittelbar nach der Bildaufnahme. Gegenüber konventionellen CMOS-Bildsensoren, die sich vor allem durch eine gute Bildqualität auszeichnen, werden die auszugebenden Daten bereits auf dem Chip auf relevante Informationen beschränkt und lediglich extrahierte Merkmale ausgegeben. *Vision Chips* ermöglichen somit eine sehr hohe Bildwiederholrate bei gleichzeitig deutlich niedrigeren Anforderungen bzgl. der Übertragungsbandbreite und sind insbesondere für die Beobachtung sehr schneller Prozesse attraktiv.

Obwohl das Konzept der gemeinsamen Betrachtung von Bildaufnahme und -verarbeitung bereits in den Anfangsjahren der Halbleiter-Bildsensoren aufgegriffen wurde, können die meisten beschriebenen Sensoren als Machbarkeitsnachweise für bestimmte Pixelzellen- bzw. Bildverarbeitungstechnologien betrachtet werden. So finden sich, bis auf den in der optischen Maus eingesetzten Sensor zur Bestimmung der Verschiebung relativ zum Untergrund, nur für sehr wenige Sensoren Hinweise auf einen kommerziellen Einsatz. Neben einer geringen optischen Auflösung und einer eingeschränkten Empfindlichkeit können der Verzicht auf integrierte Steuerwerke und die erhebliche Komplexität bzgl. der Programmierung als wesentliche Hindernisse für einen breiten Einsatz genannt werden.

Im Rahmen dieser Arbeit werden wesentliche Beiträge zu der zum Einsatz von *Vision Chips* in realen Aufgabenstellungen erforderlichen Infrastruktur geliefert. So wird zur Ansteuerung der einzelnen Funktionseinheiten (*Functional Unit*, FU) zunächst das Konzept eines integrierten, Multi-ASIP (*Application Specific Instruction-set Processor*) basierten Steuerwerks erarbeitet, das durch die Bereitstellung mehrerer Kontrollflüsse die Ansteuerung paralleler FU ermöglicht. Die praktische Umsetzung des Konzepts in Hardware erfolgt als Bestandteil eines Vision-System-on-Chip (VSoC). Eine umfangreiche Simulationsumgebung ermöglicht Untersuchungen implementierter Algorithmen sowohl hinsichtlich zeitabhängiger Effekte als auch bzgl. der Auswirkung einzelner, in Bildaufnahme- und Verarbeitung gezielt eingebrachter Fehler und Nicht-Idealitäten. Die zum Betrieb des VSoC erforderliche Entwicklungs- und Kameraplattform ist sowohl für den Einsatz unter realen Bedingungen als auch zur Entwicklung von Bildverarbeitungsaufgaben geeignet und ermöglicht dabei die transparente Nutzung der Simulationsumgebung komplementär zur eigentlichen Hardware. Zur Erschließung der vom VSoC bereitgestellten Funktionalität für tatsächliche Aufgabenstellungen erfolgt die ganzheitliche Betrachtung einer Bildverarbeitungsaufgabe bestehend aus VSoC-basierter Vor- und konventioneller Nachverarbeitung in Form sog. *„Vision Tasks“*. Zur Vereinfachung der Implementierung werden parametrierbare Skeletons bereitgestellt, in denen generelle Abläufe zur Bildaufnahme und -verarbeitung hinterlegt werden. Basierend auf den entwickelten Konzepten werden schließlich mehrere Anwendungsbeispiele umgesetzt.

Danksagung

Zunächst möchte ich mich bei Dr. Jens Döge bedanken, der mich erst als Kollege und später als Gruppenleiter und Fachreferent, an das überaus spannende Gebiet der Bildsensoren mit integrierter Signalverarbeitung herangeführt hat. Jens gab mir die Möglichkeit, über mehrere Jahre hinweg an einer Aufgabe zu arbeiten und so sowohl die Vorteile und Stärken der Technologie, als auch deren Schwachstellen und Grenzen kennenzulernen. In unzähligen Diskussionen wurden offene Fragestellungen besprochen und immer neue Ideen entwickelt, um die eigentliche Vision salonfähig zu machen.

Mein besonderer Dank gilt außerdem Prof. Rainer G. Spallek, der mich sowohl während meines Studiums, als auch während meiner Promotionszeit unterstützt und mir stets alle wissenschaftliche Freiheit gelassen hat. Er hat nie den Glauben an mich verloren, hatte stets ein offenes Ohr und räumte mir so manchen Stein aus dem Weg. Zudem möchte ich mich bei seinem Mitarbeiter Dr. Thomas B. Preußer bedanken, dessen Ratschläge und Hinweise entscheidend zum Gelingen dieser Arbeit beigetragen haben.

Auch meinen Kollegen des Fraunhofer IIS/EAS und insbesondere Georg Düsterhöft, Martin Gaber, Björn Händler, Christoph Hoppe, Stephan Gillert, Ludger Irsig, Willi Neudeck, Christoph Niemtschke, Nico Peter, Holger Priwitzer, Andreas Reichel, Patrick Russell, Dr. Peter Schneider, Christian Skubich, Lucas Stach und Dr. Andreas Wilde sei gedankt, an deren Seite ich über mehrere Jahre hinweg jeden Tag neue Überraschungen erleben durfte. Gemeinsam haben wir vor dem Tapeout geschwitzt, uns über Erfolge gefreut und so manche Niederlage überwunden.

Weiterhin gilt mein Dank Hans-Peter Großmann, der mir als Mentor zu Beginn meiner Ingenieurskarriere mit Rat und Tat zur Seite Stand und mich mehr als einmal wieder auf den richtigen Weg gebracht hat sowie Jan Schirok und Dr. Johannes Pleikies, die sich gern Zeit für mich nahmen und auch die ein oder andere kritische Frage nicht für sich behielten.

Keinesfalls vergessen werden dürfen natürlich meine Eltern und Schwiegereltern, die mir des öfteren die Kinderbetreuung abnahmen und mir so wertvolle „Freizeit“ verschafften. Zu guter Letzt gilt mein ganz besonderer Dank meiner lieben Frau Annett sowie meinen Kindern Hendrik und Moritz. Viel zu oft musstet ihr mit tröstenden Worten vorlieb nehmen weil Papa „am dicken Buch“ arbeitete. Dankeschön!

Die dieser Arbeit zugrundeliegenden Vorhaben wurden im Rahmen der folgenden Forschungsprojekte erzielt:

- Projekt ENLIGHT (http://www.enlight-project-eu), gefördert durch ENIAC und das Bundesministerium für Bildung und Forschung unter dem Förderkennzeichen 16N11442.
- Projekt FemoBiDis im Forschungsprogramm „KMU Innovativ“ des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 13N11367.
- Projekt Leistungszentrum »Funktionsintegration für die Mikro-/Nanoelektronik«, gefördert durch die Sächsische Aufbaubank (SAB) unter dem Förderkennzeichen 100245395.

Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Autor.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Nomenklatur

Abkürzungen

ADC	*Analog-to-Digital Converter*
ALU	*Arighmetic Logic Unit* – Arithmetisch-logische Einheit
APS	*Active Pixel Sensor*
ASIP	*Application-Specific Instruction set Processor*
CCD	*Charge-Coupled Device*
CCPS	*CNN Chip Prototyping System*
CFG	*Control Flow Graph* – Kontrollflussgraph
CMOS	*Complementary Metal-Oxide-Semiconductor*
CNN	*Cellular Neural Network*
CoG	*Center of Gravity*
CPA	*Cellular Processor Array*
CPU	*Central Processing Unit*
DAC	*Digital-to-Analog Converter*
DS	*Development System*
DSP	*Digital Signal Processor*
FET	Feldeffekt-Transistor
FFT	*Fast Fourier Transform*
FPGA	*Field-Programmable Gate Array*
FPN	*Fixed-Pattern Noise*
FPS	*Frames Per Second* – Bildwiederholrate
FSM	*Finite State Machine* – endlicher Automat

FU	*Functional Unit* – Funktionseinheit
GPIO	*General-Purpose Input/Output*
HDR	*High Dynamic Range*
IDE	*Integrated Development Environment*
ISS	*Instruction Set Simulator*
LAN	*Local Area Network*
LBP	*Local Binary Pattern*
LSB	*Least Significant Bit*
MISO	*Master Input, Slave Output*
MOSI	*Master Output, Slave Input*
NoC	*Network on Chip*
PE	Prozessor-Element
PLD	*Programmable Logic Device* – Programmierbare logische Schaltung
ROI	*Region Of Interest*
SIMD	*Single Instruction, Multiple Data*
SNR	*Signal to Noise Ratio* – Signal-Rausch-Abstand
SoC	*System on Chip*
VM	*Virtual Machine* – Virtuelle Maschine
VSoC	*Vision System on Chip*

Mathematische Symbole

A_D	Fläche des Fotodetektors
e	Elementarladung, $e = 1{,}602{\cdot}10^{-19}$ C
I_{DS}	Drain-Strom
$i_{\mathrm{ph},l}(t)$	Fotostrom im Pixel l zum Zeitpunkt t
I_{SI}	Ein-/Ausgabestrom der SI-Zelle
λ	Wellenlänge

Q_l	Akkumulierte Ladung in Pixel l
$Q_{\mathrm{ph},l}$	in Pixel l während eines Zeitintervalls durch Integration von $i_{\mathrm{ph},l}(t)$ akkumulierte Ladung
$s_{l,x,y}(\lambda)$	ortsabhängige spektrale Empfindlichkeit von Pixel l
t_{int}	Belichtungszeit
V_{BPh}	Bulk-Spannung des Foto-FET
V_{DD}	Versorgungsspannung
V_{DPh}	Drain-Spannung des Foto-FET
V_{GPh}	Gate-Spannung des Foto-FET

Eigene Veröffentlichungen

[DHRP15] Döge, J. ; Hoppe, C. ; Reichel, P. ; Peter, N.: A 1 Megapixel HDR Image Sensor SoC with Highly Parallel Mixed-Signal Processing. In: *International Image Sensor Workshop (IISW)* IEEE, 2015

[RD14a] Reichel, Peter ; Döge, Jens: Hardware/Software Infrastructure for ASIC Commissioning and Rapid System Prototyping. In: *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on* IEEE, 2014

[RD14b] Reichel, Peter ; Döge, Jens: Prototyping-Plattform zur Inbetriebnahme integrierter Schaltkreise. In: Schneider, Peter (Hrsg.) ; Dittrich, Michael (Hrsg.): *Dresdner Arbeitstagung Schaltungs- und Systementwurf, DASS 2014. Tagungsband.*, 2014, S. 50–55

[RDH+16] Reichel, Peter ; Döge, Jens ; Hoppe, Christoph ; Peter, Nico ; Reichel, Andreas ; Schneider, Peter: Simulation platform for application development on a Vision-System-on-Chip with integrated signal processing. In: *Journal of Electronic Imaging* 25 (2016), Nr. 4, 041004. `http://dx.doi.org/10.1117/1.JEI.25.4.041004`. – DOI 10.1117/1.JEI.25.4.041004. ISBN 1017–9909

[RDP+16] Reichel, Peter ; Döge, Jens ; Peter, Nico ; Hoppe, Christoph ; Reichel, Andreas ; Schneider, Peter: Software Environment for Holistic Vision-System-on-Chip Programming. In: *Electronic Imaging* 2016 (2016), Nr. 12. – ISSN 2470–1173

[RDPH15] Reichel, Peter ; Döge, Jens ; Peter, Nico ; Hoppe, Christoph: An ASIP-based Control System for Vision Chips with Highly Parallel Signal Processing. In: *The 24th IEEE International Symposium on Industrial Electronics (ISIE)* IEEE, 2015

[RHDP15] Reichel, Peter ; Hoppe, Christoph ; Döge, Jens ; Peter, Nico: Simulation Environment for a Vision-System-on-Chip with Integrated Processing. In: *Proceedings of the International Conference on Distributed Smart Cameras (ICDSC)*, 2015

1 Einleitung

1.1 Motivation

CMOS-Bildsensoren [Dur14, Oht10] sind inzwischen sehr weit verbreitet und es gibt kaum ein Anwendungsfeld, in dem sie noch nicht eingesetzt werden [PJ15]. Die Kombination einer sehr guten Bildqualität mit gleichzeitig hohen Bildwiederholraten sowie der Einsatz moderner Bildverarbeitungsalgorithmen erschließen ständig neue Anwendungsfelder. In konventionellen Bildverarbeitungssystemen werden die Aufgabenstellungen der Bildaufnahme sowie der eigentlichen Bildverarbeitung weitgehend unabhängig voneinander betrachtet. Die von den Fotodetektoren im Bildsensor aufgenommene Repräsentation der beobachteten Szene wird ausgelesen und direkt zu einem digitalen Verarbeitungssystem übertragen. In der industriellen Bildverarbeitung [DSAS11] kommen meist Flächenkameras zum Einsatz, deren aufgezeichnete Bilder mit Hilfe PC-basierter Verarbeitungssysteme ausgewertet werden (siehe Abb. 1-1). Die Bilddaten dienen somit als gemeinsame Sprache zwischen beiden Aufgabenstellungen. Bei der Beobachtung und Steuerung sehr schneller Prozesse, bei denen neben einer sehr hohen Bildwiederholrate auch eine minimale Latenz erforderlich ist, geraten derartige Systeme jedoch an ihre Grenzen. Ein wesentlicher Flaschenhals stellt hierbei der Transfer der Bildinformationen vom Sensor zum Verarbeitungssystem dar.

Bildsensoren mit integrierter Signalverarbeitung [Moi97, Zar11] – sogenannte *„Vision Chips“* – hingegen ermöglichen die Ausführung besonders rechenintensiver Verarbeitungsschritte während oder unmittelbar nach der Bildaufnahme. Durch frühzeitige Merkmalsextraktion werden die auszugebenden Daten auf relevante Informationen beschränkt und der in konventionellen Systemen vorhandene Flaschenhals der Datenübertragung somit umgangen. Für spezielle Aufgabenstellungen wurden *Vision Chips* mit festem Funkti-

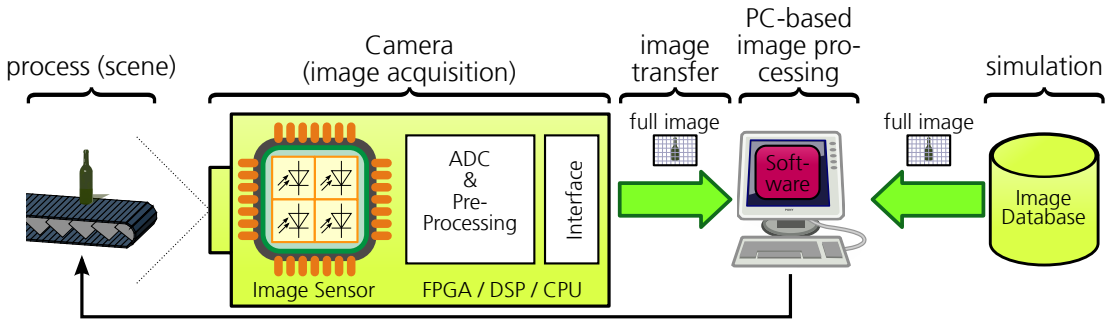


Abbildung 1-1: Klassisches Kamera-PC-basiertes Bildverarbeitungssystem [RDH+16].

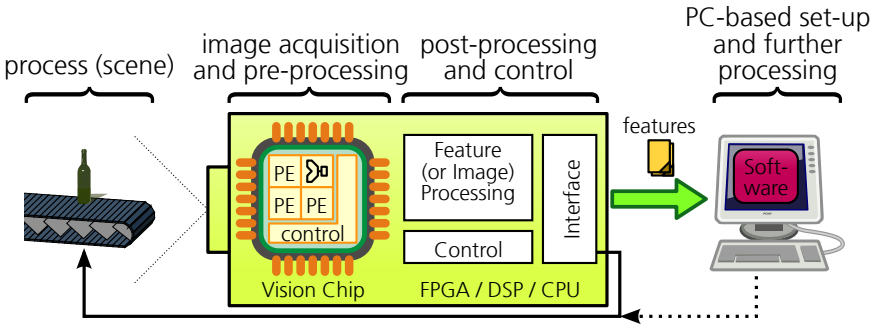


Abbildung 1-2: Bildverarbeitungssystem basierend auf einem *Vision Chip*.

onsumfang entwickelt, die genau für eine Anwendung zugeschnitten sind. Als Beispiel sei der in der optischen Maus [Lyo81] eingesetzte Sensor genannt, der die Verschiebung relativ zum Untergrund berechnet und anstelle realer Bilddaten lediglich Merkmale in Form von Verschiebungsvektoren ausgibt. Programmierbare *Vision Chips* können hingegen als „Bild-(Sensor-)Prozessoren“ aufgefasst werden, deren Prozessor-Elemente (PE) flexibel an die jeweilige Applikation angepasst werden können. Der Integrationsgrad reicht dabei von einzelnen, im Auslesepfad nachgeschalteten PE über jeweils ein PE pro Zeile bzw. Spalte bis hin zu einem eigenen PE in jeder Pixel-Zelle [Hon01, Zar11]. Durch Ausnutzung der inhärenten Parallelität von Bildaufnahme und -verarbeitung, können hohe Bildwiederholraten und geringe Reaktionszeiten erzielt werden. *Vision Chips*, die neben den Funktionseinheiten zur Bildverarbeitung auch über ein integriertes Steuerwerk für deren Ansteuerung verfügen, werden auch als *Vision-System-on-Chip* (VSoC) bezeichnet [RVLCC+04, RVDCJG+10].

In Abb. 1-2 ist ein Bildverarbeitungssystem basierend auf einem *Vision Chip* dargestellt, bei dem Teile der Verarbeitungssoftware vom PC in die Kamera verlagert wurden. Anders als bei gewöhnlichen Bildsensoren können vom Sensor bereits vorverarbeitete Merkmalsdaten anstelle vollständiger Bilder ausgegeben werden. Je nach Komplexität der Aufgabenstellung ist eine weitere Verarbeitung im Kamerasystem bzw. im PC erforderlich. Mitunter können nicht alle gewünschten Operationen auf dem *Vision Chip* abgebildet werden bzw. ist deren Umsetzung anderweitig effektiver möglich. In der Kamera erfolgt dies oft mit Hilfe von FPGAs („*Field Programmable Gate Array*“), digitalen Signalprozessoren (DSP) oder gewöhnlichen Prozessoren. Sind für eine Aufgabenstellung ausschließlich gut abbildbare Operationen notwendig, so kann auf einzelne Komponenten der Verarbeitungskette verzichtet werden. Insbesondere für mobile Anwendungen, bei denen die Leistungsaufnahme eine zentrale Rolle spielt, kann, neben dem Verzicht auf den Einsatz eines PCs, ein autonomer Betrieb des *Vision Chips* sinnvoll sein.

Im Vorfeld des Entwurfs eines neuartigen VSoC [DHRP15], basierend auf der von Jens Döge vorgestellten Pixelzellen-Technologie [Dög08], wurden auch Betrachtungen bzgl. der geringen Akzeptanz vorangegangener *Vision Chip*-Konzepte angestellt. In der angegebenen Literatur werden zahlreiche Arbeiten für *Vision Chip*-Konzepte mit z.T. sehr unterschiedlichen Funktionsprinzipien angeführt. Deren Autoren haben sich jedoch zumeist auf

die eigentliche Verarbeitung innerhalb der PE konzentriert. Neben dem CNN-basierten („*Cellular Neural Network*“) Bildverarbeitungssystem „*EyeRIS*“ [Ana08, RVDCJG+08], finden sich lediglich für den von Lindgren et al. [LMJM05] vorgestellten Sensor Hinweise für einen kommerziellen Einsatz. Die meisten Systeme stellen hingegen ausschließlich einen Machbarkeitsnachweis des jeweiligen Verarbeitungsprinzips dar und besitzen daher nur eine geringe optische Auflösung sowie eine geringe Empfindlichkeit [DTV+11]. Durch den Verzicht auf integrierte Steuerwerke erfolgt die Ansteuerung durch externe Komponenten, wodurch die Komplexität des Gesamtsystems deutlich gesteigert wird. Die Programmierung ist zudem sehr aufwändig und wird kaum durch Bibliotheken unterstützt. Nicht-Idealitäten, wie z.B. *Fixed-Pattern-Noise* (FPN) oder defekte Pixel, können in konventionellen Systemen noch vor der eigentlichen Bildverarbeitung korrigiert werden. In *Vision Chips* ist eine unabhängige Betrachtung von Bildaufnahme und Bildverarbeitung jedoch nicht möglich. Die integrierte Merkmalsextraktion erfordert daher, neben einer Berücksichtigung von Nicht-Idealitäten, auch die Betrachtung zeitabhängiger Effekte sowie den Umgang mit einer möglicherweise begrenzten Genauigkeit durch die Verarbeitung in der analogen Domäne.

Obwohl die Idee der sensornahen, integrierten Verarbeitung bereits auf das Jahr 1979 zurückgeht [NNTF79], konnten sich *Vision Chips* bis jetzt nur bedingt durchsetzen. Die zentrale Frage ist daher: Welche Schritte sind zum effizienten Einsatz von *Vision Chips* und deren Einbettung in konkrete Bildverarbeitungsanwendungen notwendig?

1.2 Zielstellung

Unter Verwendung der ladungsbasierten Schaltungstechnik [Dög08] wurde ein neuartiges VSoC [DHRP15] (siehe 3.1) entwickelt, das gegenüber dem Stand der Technik über eine höhere Auflösung sowie eine größere Empfindlichkeit verfügt. Die on-chip Verarbeitung umfasst dabei sowohl Operationen in analoger Domäne, wie z.B. spaltenweise Faltungsoperationen, als auch die spaltenparallele, digitale Verarbeitung durch ein *Single-Instruction-Multiple-Data*-Array (SIMD). Als Zielanwendungen wurden zunächst die Unterstützung der 3D-Rekonstruktion durch Triangulation mit dem Laser-Lichtschnitt-Verfahren [VM10] sowie die leistungseffiziente Detektion von Bewegungen in einem beobachteten Raum ausgewählt. Die Ansteuerung ist dabei frei programmierbar und nicht auf diese Anwendungsszenarien begrenzt.

Zielstellung dieser Arbeit ist die grundlegende Analyse der für den Einsatz von *Vision Chips* notwendigen Umgebung und, darauf aufbauend, Untersuchungen zur Entwicklung der zum Entwurf, zur Realisierung sowie zur Validierung von Bildverarbeitungsaufgaben erforderlichen Infrastruktur. Die Grundidee ist ein durchgängiges System, bestehend aus Hardware- und Software-Komponenten, das das fehlende Bindeglied zwischen der on-chip Verarbeitung im VSoC und dessen Integration in konkrete Anwendungen schaffen soll. Das Spektrum erstreckt sich, ausgehend vom Konzept eines integrierten, programmierbaren Steuerwerks, über eine Kamera- und Entwicklungsplattform zur Unterstützung der einfachen Integration in Bildverarbeitungsanwendungen, bis hin zu einem Simulationssystem zur Validierung implementierter Algorithmen unter Berücksichtigung von

Nicht-Idealitäten und zeitabhängigen Effekten. Schließlich soll das Zusammenspiel der einzelnen Komponenten anhand verschiedener Anwendungsbeispiele aufgezeigt werden.

Im Rahmen dieser Arbeit entstanden die folgenden wesentliche Beiträge als Bestandteil der zum Einsatz eines VSoC notwendigen Infrastruktur:

1. Zur Ansteuerung der Funktionseinheiten (*Functional Unit*, FU) von *Vision Chips* wurde das Konzept eines integrierten, Multi-ASIP (*Application Specific Instruction-set Processor*) basierten Steuerwerks entwickelt [RDPH15]. Die einzelnen ASIPs erweitern jeweils den gleichen Stack-basierten Prozessorkern um spezifische Instruktionen zur Ansteuerung der FU. Durch die Unterstützung mehrerer Kontrollflüsse können unabhängige FU parallel arbeiten. Die praktische Umsetzung des Konzepts in Hardware erfolgte als Bestandteil des von Döge et al. vorgestellten VSoC [DHRP15].

2. Insbesondere zur Simulation des dieser Arbeit zugrundeliegenden VSoC wurde eine umfangreiche Simulationsumgebung [RHDP15, RDH+16] realisiert. Gegenüber anderen Arbeiten erlaubt diese die Betrachtung zeitabhängiger Effekte und die gezielte Injektion von Defekten und Nicht-Idealitäten sowie Untersuchungen bzgl. deren Auswirkungen auf den Prozess der Bildaufnahme und -verarbeitung. Das Simulationssystem ist zum realen VSoC binär-kompatibel und modelliert außerdem wesentliche Bestandteile der Entwicklungs- und Kameraplattform.

3. Als Grundlage für die eigentliche Entwicklungs- und Kameraplattform wurde eine spezielle Kommunikations-Infrastruktur [RD14a] entwickelt. Diese ermöglicht die einfache Ansteuerung eigener im FPGA realisierter Hardware-Komponenten durch Linux-basierte Anwendungsprogramme. Die gleiche Ansteuersoftware kann dabei sowohl lokal bei Verwendung einer in die Entwicklungsplattform integrierten CPU als auch über eine Netzwerkverbindung ausgehend von einem entfernten PC bzw. basierend auf einem SystemC-Modell der Hardware eingesetzt werden. Die entwickelte Infrastruktur kommt bei der Kamera- und Entwicklungsplattform zum Einsatz und ermöglicht einen autonomen Betrieb als PC-unabhängiges Kamerasystem sowie die Verwendung des Simulationssystems komplementär zur eigentlichen Entwicklungsplattform.

4. Zur Abbildung von Bildverarbeitungsaufgaben wurde das Konzept des *Vision Tasks* [RDP+16] entwickelt, das eine ganzheitliche Betrachtung bestehend aus VSoC basierter Bildaufnahme und -vorverarbeitung sowie konventioneller Nachverarbeitung ermöglicht. Dabei wird durchgehend die Programmiersprache Python eingesetzt, wobei einzelne Programmabschnitte den ASIPs des Steuerwerks zugeordnet und automatisch in die jeweilige Assemblersprache übersetzt werden können. Durch die Bereitstellung von *Skeletons* zur Beschreibung der Abhängigkeiten der einzelnen Kontrollflüsse wird die Komplexität der Abbildung konkreter Bildverarbeitungsaufgaben deutlich reduziert.

1.3 Aufbau der Arbeit

Zunächst werden in Kapitel 2 einige, in der Literatur beschriebene *Vision Chips* vorgestellt und hinsichtlich verschiedener Merkmale klassifiziert. Dabei wird ein spezieller Fokus auf die Simulation von Bildsensoren gelegt. Nach der Darstellung vorhandener Entwicklungsplattformen werden Anforderungen an die zum Einsatz von *Vision Chips* notwendige Infrastruktur abgeleitet.

In Kapitel 3 wird zunächst die Architektur sowie der Funktionsumfang des entwickelten VSoC vorgestellt, bevor die verschiedenen Möglichkeiten zur Realisierung von Steuerwerken diskutiert werden. Daraufhin wird detaillierter auf das Konzept eines integrierten, Multi-ASIP basierten Steuerwerks eingegangen.

Kapitel 4 widmet sich der zur Entwicklung von Anwendungen und Validierung von Algorithmen entwickelten Simulationsumgebung. Nach der Vorstellung der Architektur wird die Modellierung von Bildaufnahme und -verarbeitung genauer betrachtet und schließlich der Einsatz bei der Entwicklung von Anwendungen dargestellt.

In Kapitel 5 wird auf die Entwicklungs- und Kameraplattform sowie auf Methoden und Werkzeuge zur Umsetzung von Bildverarbeitungsaufgaben unter Verwendung des VSoC eingegangen. Während die Plattform zur Entwicklung vom PC gesteuert wird, ist für den realen Einsatz als Messsystem auch ein autonomer Betrieb möglich. Zur Vereinfachung des Entwurfs komplexer Bildverarbeitungsanwendungen ermöglicht die zugehörige Software-Umgebung die Verwendung VSoC-integrierter Vor- sowie konventioneller Nachverarbeitung in einem gemeinsamen Programm.

In Kapitel 6 werden einige Anwendungsbeispiele unter Verwendung des VSoC sowie der entstandenen Bildverarbeitungsplattform beschrieben. Neben einem parametrierbaren Kamerasystem mit Integrationsmöglichkeiten für digitale Nachverarbeitungsfunktionen im VSoC wird die 3D-Rekonstruktion durch Triangulation mit dem Laser-Lichtschnitt-Verfahren sowie die optische Tonabnahme von einer E-Gitarre dargestellt.

Schließlich erfolgt in Kapitel 7 eine Zusammenfassung der Arbeit sowie ein Ausblick auf mögliche, zukünftige Entwicklungen.

2 Stand der Technik

2.1 Klassifikation von Vision-Chips

2.1.1 Bildaufnahme und Verarbeitung

Bei konventionellen Bildsensoren [Dur14], z.B. sog. CMOS *Active Pixel Sensors* (APS) [Oht10], besteht die Aufgabenstellung darin, eine möglichst exakte, digitale Repräsentation der beobachteten und dabei auf die Sensoroberfläche abgebildeten Szene auszugeben. Dabei findet sowohl eine räumliche als auch eine zeitliche Diskretisierung statt, wobei die Ausgabe i.d.R. in zuvor selektierbaren, rechteckigen Regionen (*Region-of-Interest*, ROI) geschieht. Je nach Anwendung werden an die Bildinformationen eine Reihe von Anforderungen bzgl. deren Qualität [RVDCJG$^+$10] gestellt. Neben der Auflösung, d.h. der Breite und Höhe des Bildes, und dem Datendurchsatz in Bildern pro Sekunde sind vor allem der Dynamikumfang, die Linearität, der Signal-Rausch-Abstand (SNR) oder die Treue der Farbwiedergabe zu nennen. Dies ermöglicht den Vergleich verschiedener Sensoren und somit eine Selektion anhand der Anforderungen einer konkreten Bildverarbeitungsaufgabe.

Die vom Sensor aufgenommenen Bilder dienen schließlich als Eingabe zur digitalen Bildverarbeitung. Wie in Abb. 2-1 dargestellt, können die einzelnen Verarbeitungsschritte grob in drei Ebenen [DC98, Bai11, PLB12] eingeteilt werden. Während die Komplexität der Operationen dabei nach oben hin zunimmt, sinkt gleichzeitig deren Parallelisierbarkeit. Die untere Ebene wird als *Low-Level-Vision* bezeichnet und beinhaltet einfache Operationen, die i.d.R. ein Eingabebild (Eingabematrix) direkt in ein Ausgabebild transformieren. Jedes einzelne Pixel trägt dabei lediglich eine geringe Menge an Information in Form seines Helligkeitswertes. Die Verarbeitung der Pixeldaten erfolgt weitgehend unabhängig voneinander, weshalb eine hohe Parallelität auf Pixelebene erreicht werden kann. Neben Methoden zur Bildverbesserung können hierbei Filterkerne, Morphologische Operationen oder Kanten- bzw. Ecken-Extraktion genannt werden. Die Operationen der mittleren Ebene (*Mid-Level-Vision*) dienen der Extraktion von Merkmalen, wobei, im Gegensatz zur darunterliegenden Ebene, eine Reduktion der Datenmenge sowie eine Steigerung des Informationsgehalts pro Datenelement stattfindet. Während die Eingabe durch Bilddaten (Matrix) erfolgt, kann die Ausgabe durch Vektoren charakterisiert werden. Eine parallele Verarbeitung kann lediglich auf Ebene der Merkmale erfolgen, deren Anzahl deutlich niedriger als die Zahl der Pixel im Eingabebild ist. Auf der oberen Ebene (*High-Level-Vision*) werden die extrahierten Merkmale schließlich bewertet, um einzelne Objekte zu erkennen und die Szene zu analysieren. Die Ausgabedaten sind i.d.R. Skalare, deren Informationsgehalt gegenüber den Merkmalen noch einmal zunimmt.

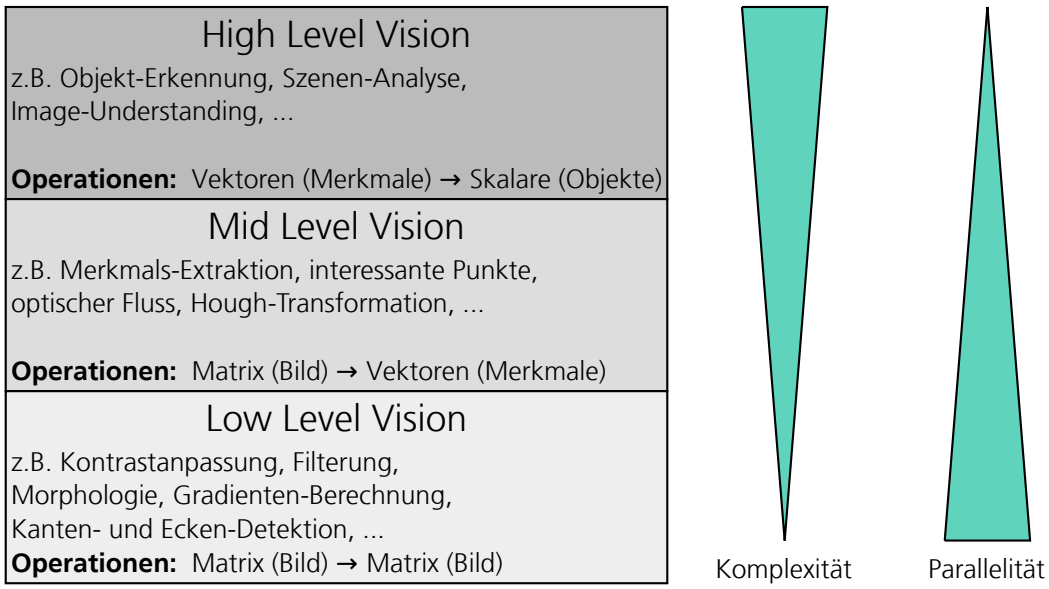


Abbildung 2-1: Aufteilung in Low-, Mid- und High-Level Vision.

Werden Vorverarbeitungsfunktionen aus dem Low- und Mid-Level-Vision Bereich mit dem Ziel einer frühzeitigen Datenreduktion in einen Bildsensor integriert, so wird von *Vision Chips* gesprochen. Bei derartigen Sensoren wird auf die Ausgabe vollständiger Bilder verzichtet, wodurch eine deutlich niedrigere Übertragungsbandbreite erforderlich ist sowie eine höhere Bildwiederholrate möglich wird. Zudem wird die elektrische Leistungsbilanz des Gesamtsystems deutlich verbessert, da weniger Energie in die Datenübertragung investiert werden muss und sich weitergehende, digitale Nachverarbeitungsoperationen vereinfachen lassen. Anforderungen bzgl. der Qualität beziehen sich nun jedoch auf die ausgegebenen Merkmale, was die Selektion eines *Vision Chips* für eine konkrete Anwendung bzw. den Vergleich verschiedener *Vision Chips* untereinander deutlich erschwert. Moderne APS enthalten oft ebenfalls umfangreiche digitale Nachverarbeitungsfunktionen [YPEC07], deren ausschließliches Ziel jedoch die Verbesserung der Bildqualität ist. Da keine Merkmalsextraktion stattfindet und stattdessen stets vollständige Bilder ausgegeben werden, werden derartige Systeme (z.B. ON Semiconductor AS0260 [Sem15]) von echten *Vision Chips* abgegrenzt.

In der Literatur lassen sich zahlreiche Beispiele für *Vision Chips* mit festem Funktionsumfang finden, die für spezielle Anwendungen entwickelt wurden. Anderson et al. [ABD+91] stellen einen Sensor vor, der zur Erkennung von Fingerabdrücken dient und neben der Steuerung der Bildaufnahme auch die Binarisierung sowie den Vergleich mit Referenz-Abdrücken durchführt. Der von Ni und Guan [NG00] vorgestellte Sensor nutzt analoge und digitale Verarbeitungseinheiten, um interessante Punkte zur 3D-Rekonstruktion mittels eines Stereo-Aufbaus zu finden. Ein weiteres Beispiel ist der bereits erwähnte, in der optischen Maus [Lyo81] eingesetzte Sensor, der mit Hilfe von Autokorrelationsfunktionen die Verschiebung gegenüber einer Oberfläche berechnet und ausgibt. Alle

Sensoren mit festem Funktionsumfang implementieren lediglich einen bestimmten Algorithmus und sind nur hinsichtlich weniger Optionen parametrierbar. Da die Entwicklung von Bildsensoren aufgrund des erforderlichen *Full-Custom-ASIC-Designs* sehr aufwändig ist, wurden programmierbare Systeme entwickelt, deren Funktionseinheiten flexibel eingesetzt werden können und die somit nicht auf eine bestimmte Applikation festgelegt sind. Diese Systeme werden im Folgenden genauer behandelt.

2.1.2 Klassifikation nach Berechnungsansatz

Nach Dupret et al. [DTV+11] können *Vision Chips* hinsichtlich des zugrundeliegenden Berechnungsansatzes („*computational approach*“) unterschieden werden. Insbesondere während der Frühphase der Entwicklung von *Vision Chips* wurde häufig versucht, biologische Vorgänge schaltungstechnisch nachzuempfinden („*biologically inspired*“). Globale Bildverarbeitungsoperationen, wie Diffusion oder das Weichzeichnen des Bildes, können in analoger Domäne schaltungstechnisch gut umgesetzt werden. Besonders hervorzuheben ist hierbei die von Carver Mead vorgestellte „*Silicon Retina*“ [Mea89], bei der das von den Fotodetektoren aufgezeichnete Bild mit Hilfe eines Widerstandsnetzwerks weichgezeichnet wird. Der Sensor besitzt einen festen Funktionsumfang, wobei für jedes Pixel die Differenz zum weichgezeichneten Bild an der jeweiligen Position ausgegeben wird.

Auch *Cellular Neural Network* (CNN) basierte Sensoren gehören zur Klasse der biologisch inspirierten *Vision Chips*. Bei ACE4k [LEDCRV02a] sowie dessen Nachfolger ACE16k [RVLCC+04, LEDCRV02b] verfügt jede Zelle über einen eigenen Fotodetektor. Die gewichtsabhängige Summation des eigenen Speicher- bzw. Eingabewertes mit den Werten der Nachbarzellen erfolgt in analoger Domäne. Die Gewichte selbst werden hingegen digital vorgegeben und durch Digital-Analog-Umsetzer (DAC) in eine korrespondierende Spannung umgewandelt. Durch die Anpassung der Gewichte zur Laufzeit können Bildverarbeitungsalgorithmen umgesetzt werden, bei denen stets alle Neuronen des Sensors beteiligt sind. Durch Weiterentwicklung der Sensoren entstand mit „Q-Eye“ [Ana12, RVDCJG+10] bzw. der Kameraplattform „Eye-RIS“ [RVDCJG+08] ein auf dem CNN-Berechnungsansatz beruhendes, kommerzielles Kamerasystem.

Beim zweiten Berechnungsansatz nach Dupret et al. („*algorithmically driven*“) werden einzelne, innerhalb eines bestimmten Algorithmus häufig benötigte bzw. besonders rechenintensive, mathematische Operationen für eine Implementierung auf dem Sensor ausgewählt und geeignet implementiert. Die erste Arbeit hierzu stammt von Nudd et al. [NNTF79] und repräsentiert gleichzeitig den ersten bekannten *Vision Chip* überhaupt. Der Sensor arbeitet nach dem *Charge Coupled Devices* (CCD) Prinzip und ermöglicht die Anwendung gaußscher Faltungskerne während des Auslesens.

Ein weiteres Beispiel ist der von Lindgren et al. [LMJM05] vorgestellte Bildsensor, der auf einer gewöhnlichen APS Pixelmatrix beruht. Jede Spalte besitzt eine bit-serielle ALU (*Arithmetic Logic Unit*) sowie einen bzgl. der Auflösung konfigurierbaren A/D-Wandler. Neben der spaltenparallelen Nachverarbeitung kann so auch die Extraktion von Merkmalen, wie z.B. Sub-Pixel genaue Minimum- oder Maximumsuche entlang der Spalte, implementiert werden.

In Tabelle 2-1 sind verschiedene *Vision Chips* sowohl hinsichtlich des Berechnungsansatzes als auch bzgl. des Grades der Parallelisierung sowie nach der Verarbeitungsdomäne der Prozessor-Elemente (PE) eingeordnet. Ein industrieller Einsatz lässt sich dabei durch die betrachtete Literatur nur für wenige Systeme sicher belegen.

2.1.3 Klassifikation nach dem Grad der Parallelisierung

Wie in Abb. 2-2 dargestellt, können *Vision Chips* hinsichtlich des Grades der Parallelisierung [Hon01, Zar11] auf Ebene der Pixel unterschieden werden (siehe auch Tab. 2-1). Die verschiedenen Topologien – pixelweise und spalten- bzw. zeilenweise Parallelisierung sowie rein sequentielle Verarbeitung – eignen sich für unterschiedliche Klassen von Algorithmen (vgl. Abb. 2-1) unterschiedlich gut.

Bei der pixelweisen Verarbeitung (siehe Abb. 2-2a) wird jedem Pixel ein eigenes PE zugeordnet, wodurch der maximale Grad der Parallelisierung erreicht wird. Die PE arbeiten zumeist nach dem *Single-Instruction-Multiple-Data* (SIMD) Prinzip und führen stets die gleiche Operation auf dem gesamten Bildfeld aus, wodurch auf hohe Taktfrequenzen verzichtet werden kann. Die mögliche Komplexität der PE, d.h. insbesondere die Anzahl der Transistoren, wird vornehmlich durch die zur Verfügung stehende Fläche sowie deren Konnektivität zur Umgebung begrenzt. Neben Einschränkungen bzgl. des Füllfaktors ist eine Vergrößerung der Pixel aufgrund abnehmender örtlicher Auflösung nur bedingt möglich. Als Beispiele sind insbesondere die Publikationen der Arbeitsgruppe um Dudek et al. zu nennen. Bei SCAMP3 (*„Current-Mode Analogue Processor Array“*) [Dud05, Dud11] verfügt jedes der 128×128 PE über mehrere sog. Strom-Speicherzellen („SI-Zellen“) zum Ablegen von Grauwerten in Form analoger Werte. Zur Speicherung wird dabei ein Kondensator auf eine zum Strom proportionale Spannung aufgeladen. Der so gespeicherte Wert kann zu einem späteren Zeitpunkt durch einen äquivalenten Strom wieder ausgegeben werden. Zusätzlich ist ein Datenaustausch mit den Nachbarn möglich. Berechnungen werden mit Hilfe geschalteter Ströme durch Verschiebung zwischen den SI-Zellen durchgeführt, wobei neben Addition, Subtraktion und Negation, auch die Multiplikation mit Konstanten möglich ist. Ein Komparator dient schließlich zum Übergang in die digitale Domäne. Bei SCAMP5 [CLB+13, CBW+12], der über eine höhere Auflösung von 256×256 Pixel verfügt, enthält jedes der PE zusätzliche digitale Register sowie einen analogen Quadrierer. Ein globales Netzwerk ermöglicht durch *„asynchronous wave propagation“* die schnelle Ausführung von Fülloperationen. Bei ASPA3 [LD13] sind die 160×80 PE hingegen rein digital realisiert und verfügen jeweils über 72 Bit lokalen Speicher sowie eine bit-serielle ALU. Komuro et al. beschreiben mit S³PE [KKI04b, IK01] einen weiteren *Vision Chip* mit digitalen, bit-seriellen PE in jedem Pixel.

Die zeilen- oder spaltenweise Verarbeitung (siehe Abb. 2-2b) ordnet jeweils einer Zeile oder Spalte ein PE zu. Die ebenfalls auf dem SIMD-Prinzip beruhenden Berechnungen erfolgen während dem eigentlichen Auslesevorgang. Während so die Extraktion von Merkmalen entlang einer Zeile bzw. Spalte einfach möglich ist, können globale Operationen nicht direkt abgebildet werden. Durch die Anordnung neben der eigentlichen Pixelmatrix wird der Füllfaktor im Pixel nicht beeinflusst. Zudem können die PE zumindest in einer Richtung skaliert werden, wodurch die schaltungstechnische Umsetzung

Arbeit	Jahr	Berechnungs-ansatz	Grad der Par-allelisierung	Verarbeitungsdomäne
ACE400 [DCERV+97]	1997	biolog.	pixelweise	analog / mixed-signal
ACE4k [LEDCRV02a]	2002	biolog.	pixelweise	analog / mixed-signal
ACE16k [RVLCC+04, LEDCRV02b]	2004	biolog.	pixelweise	analog / mixed-signal
Q-Eye[a] [Ana12, RVDCJG+10]	2010	biolog.	pixelweise	analog / mixed-signal
SCAMP-1 [DH01]	2001	algorith.	pixelweise	analog / mixed-signal
SCAMP-3 [Dud05, Dud11]	2005	algorith.	pixelweise	analog / mixed-signal
SCAMP-5 [CLB+13, CBW+12]	2012	algorith.	pixelweise	analog / mixed-signal
MIPA4k [PLP09, Zar11]	2009	algorith.	pixelweise	analog / mixed-signal
Flip-Q [FBCGCG11]	2011	algorith.	pixelweise	analog / mixed-signal
Ginhac et al. [GDHP10]	2010	algorith.	pixelweise	analog / mixed-signal
ASPA-3 [LD13]	2013	algorith.	pixelweise	digital
S^{3}PE [KKI04b, IK01]	2004	algorith.	pixelweise	digital
PARIS1 [EBD+06, DKN02]	2002	algorith.	spaltenpar.	analog / mixed-signal
Lindgren et al.[b] [LMJM05]	2005	algorith.	spaltenpar.	digital
Verdant et al. [VDV+13]	2013	algorith.	spaltenpar.	analog / mixed-signal
Zhang et al. [ZFW11]	2011	algorith.	kombiniert	digital
Döge et al.[c] [DHRP15]	2015	algorith.	spaltenpar.	analog / mixed-signal und digital

[a] Q-Eye wird im Kamerasystem „Eye-RIS“ eingesetzt – ein kommerzielles Produkt der Firma AnaFocus.
[b] Der von Lindgren et al. vorgestellte Sensor wird in einem Laser-Triangulationssensor des Herstellers Sick IVP verwendet.
[c] Das neuartige Vision-System-on-Chip, basierend auf ladungsbasierter Schaltungstechnik, wird in Abschnitt 3.1 genauer erläutert. Der industrielle Einsatz wird angestrebt.

Tabelle 2-1: Klassifikation verschiedener *Vision Chips* hinsichtlich des Berechnungsansatzes, des Grades der Parallelisierung sowie nach der Verarbeitungsdomäne der Prozessor-Elemente.

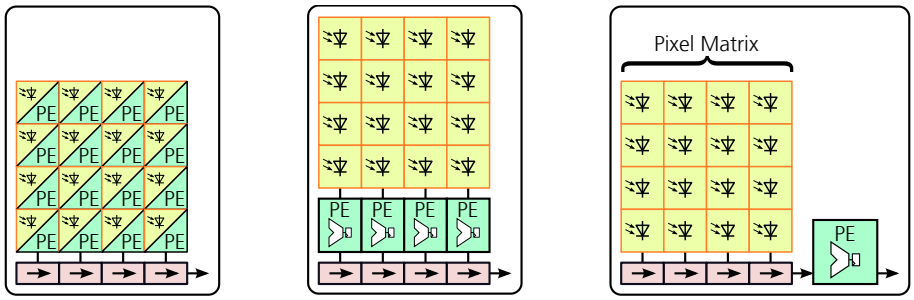
komplexerer Operationen möglich wird. Als Beispiel kann PARIS1 [DKN02, EBD+06] genannt werden, bei dem sich die Verarbeitung im Pixel auf die Speicherung von bis zu vier analogen Grauwerten beschränkt, die jeweils auf die Spaltenleitung ausgegeben werden können. Jede Spalte verfügt zudem über ein in analoger Domäne arbeitendes PE, welches Addition, Subtraktion und Multiplikation mit Konstanten ermöglicht. Verdant et al. [VDV+13] stellen einen *Vision Chip* mit spaltenweiser Verarbeitung basierend auf einer Standard-APS-Pixelmatrix vor. Jeweils 10×10 Pixel werden zu einem Makropixel zusammengefasst, wobei jeder Makropixel-Spalte jeweils ein analoges PE zugeordnet ist. Das System ist zur Detektion von Aktivität im Bild optimiert, wobei die Ausgabe mit hoher Auflösung lediglich für aktive Regionen erfolgt. Von Laforest et al. [LDV+12] wird ein ähnliches Sensorkonzept vorgestellt, bei dem die den Makropixel-Spalten zugeordneten PE auf digitalen, 8 Bit ALUs aufbauen.

Für komplexe Operationen aus dem Mid- und High-Level-Vision Bereich (vgl. Abb. 2-1), sind sequentielle Verarbeitungseinheiten erforderlich, die ebenfalls auf dem Sensor-Chip integriert werden können (siehe Abb. 2-2c). Die Berechnungen erfolgen somit während des eigentlichen Auslesevorgangs abhängig vom konkreten Ausleseschema. Bildaufnahme und -verarbeitung sind wie im klassischen Kamera-PC basierten Ansatz voneinander getrennt, weshalb der Ansatz auch „*Single Chip Smart Camera*“ [ACC+02] bezeichnet wird. Durch die freie Anordnung, unabhängig vom Bildfeld, wird weder der Füllfaktor beeinflusst noch wird die mögliche Komplexität durch Größenbeschränkungen der Pixelgeometrie limitiert. Die innerhalb der eigentlichen Verarbeitungseinheit durchzuführenden Berechnungen können bei geeigneter Implementierung grundsätzlich parallel ausgeführt werden, jedoch wird der Datendurchsatz maßgeblich durch die Ausleserate der Pixelmatrix sowie die mögliche Verarbeitungsfrequenz bestimmt. Albani et al. integrieren bei VISoC [ACC+02] einen Bildsensor mit ADC, einen 32 Bit Prozessor sowie einen speziellen, digitalen Co-Prozessor auf dem gleichen Chip. Die seriell ausgelesenen Bilder werden mit Hilfe des Co-Prozessors vorverarbeitet, wobei neben Filter- und Faltungsoperationen auch die Detektion von Bewegungen möglich ist.

Abhängig von der Aufgabenstellung, sind auch Kombinationen der o.g. Ansätze möglich. So beschreiben Zhang et al. [ZFW11] ein VSoC, bei dem ein APS zunächst zeilenparallel ausgelesen wird. Jeder Zeile ist ein eigenes PE zugeordnet, in welchem eine direkte Weiterverarbeitung der digitalisierten Daten während des Auslesens erfolgen kann. Ebenfalls neben dem Bildfeld befindet sich ein 2D-Feld aus PE mit bit-seriellen ALUs, das mit den Ergebnissen der zeilenparallelen Berechnung befüllt wird. Sowohl die zeilenparallele Verarbeitung als auch die Berechnungen im 2D-Feld erfolgen nach dem SIMD-Prinzip. Ein sequentieller Prozessorkern dient schließlich zur Steuerung des Gesamtsystems sowie zur Weiterverarbeitung der extrahierten Merkmale.

2.1.4 Klassifikation nach Verarbeitungsdomäne

Die von den PE durchgeführten Berechnungen können in digitaler, analoger bzw. mixed-signal Verarbeitungsdomäne (siehe Abb. 2-3) ausgeführt werden. Bei der digitalen Verarbeitung erfolgen die Berechnungen zeit- und wertdiskret, wobei die Verarbeitungseinheiten (z.B. Addierer, Multiplizierer, ALU) auf elementaren, logischen Operationen basieren.



(a) Pixelweise (2D) Parallelisierung.

(b) Spalten- bzw. zeilenweise (1D) Parallelisierung.

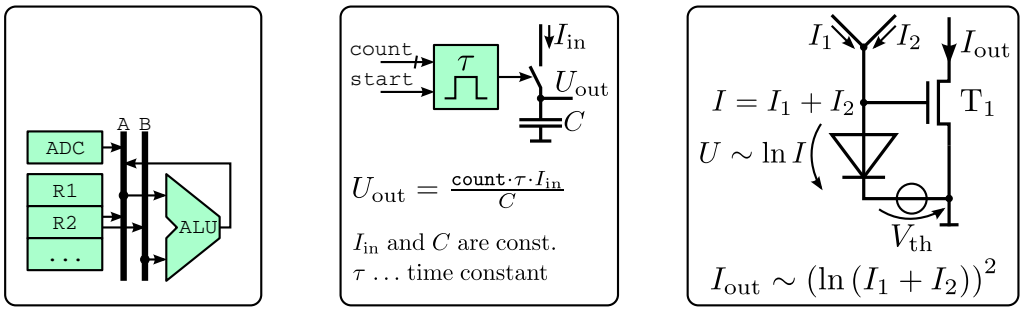
(c) Chip-Level Verarbeitung.

Abbildung 2-2: Gegenüberstellung der verschiedenen Grade der Parallelisierung.

In Abb. 2-3a wird dies exemplarisch durch einen Datenpfad mit mehreren Registern und einer ALU dargestellt. Beispiele für *Vision Chips* mit digitalen Verarbeitungseinheiten sind das von Lopich und Dudek [LD13] beschriebene System mit pixelweisen PE sowie der von Lindgren et al. [LMJM05] vorgestellte Sensor mit spaltenparallelen ALUs. Laforest et al. [LDV+12] ordnen in ihrem Sensorkonzept jeweils die Pixel von zehn aufeinanderfolgenden Spalten einem PE zu. Die PE arbeiten nach dem SIMD-Prinzip und besitzen eine 8 Bit ALU sowie eigenen, lokalen Speicher.

Bei der analogen Verarbeitung wird das Nutzsignal zeit- und wertkontinuierlich durch den Betrag einer elektrischen Größe repräsentiert. Berechnungen erfolgen durch die schaltungstechnische Ausnutzung von Bauelementeeigenschaften, wodurch komplexe mathematische Operationen oft mit wenigen Bauelementen realisiert werden können. Dies wird in Abb. 2-3c durch die Folge der Operationen Addition, Logarithmierung und abschließende Quadrierung veranschaulicht. Werden in einer Schaltung sowohl analoge als auch digitale Signale verarbeitet, so wird von „Mixed-Signal“ Verarbeitung gesprochen. Dabei können sowohl digitale Steuersignale zur Beeinflussung der analogen Schaltung dienen als auch digitale Signale mit Hilfe analoger Schaltungselemente erzeugt werden. In Abb. 2-3b wird die Anzahl ausgegebener Impulse einer definierten Länge digital vorgegeben, um die Integration eines Stromes auf einen Kondensator und somit die resultierende Spannung zu steuern.

Im *Vision Chip* Flip-Q [FBCGCG11] können nicht überlappende, rechteckige Bereiche definiert werden, innerhalb derer für die enthaltenen Pixel entweder der einfache Mittelwert oder der quadratische Mittelwert berechnet werden. Die Verbindungen werden durch Schieberegister-Ketten, die am Rand des Chips angeordnet sind, vorgegeben. Der von Ginhac et al. [GDHP10] beschriebene Sensor besitzt in jedem PE zwei Kondensatoren zur Speicherung von Grauwerten sowie einen Vier-Quadranten-Multiplizierer. In die Berechnung können auch die Werte der Nachbarpixel einfließen. Neben den SCAMP *Vision Chips* (z.B. SCAMP-3 [Dud05, Dud11], SCAMP-5 [CLB+13, CBW+12]) basieren auch die CNN-Sensoren (z.B. ACE16k [RVLCC+04, LEDCRV02b]) auf analogen PE. In Ta-



(a) Beispiel eines digitalen Datenpfades.

(b) Beispiel zur mixed-signal Verarbeitung.

(c) Beispiel eines analogen Datenpfades. Zur Quadrierung wird T_1 im Pinch-Off Bereich betrieben.

Abbildung 2-3: Gegenüberstellung der verschiedenen Verarbeitungsdomänen.

belle 2-1 sind weitere *Vision Chips* mit Zuordnung zur jeweiligen Verarbeitungsdomäne aufgelistet.

Ein Vergleich bzw. eine Bewertung von analoger oder digitaler Implementierung hängt sehr stark von den jeweiligen Rahmenbedingungen, d.h. insbesondere dem verfügbaren Platz sowie der eingesetzten Fertigungstechnologie ab. Während digitale Schaltungen bei einer Verkleinerung der Strukturgröße sehr gut skalieren, gilt dies für analoge Schaltungen im Allgemeinen nicht. Bildsensoren sind hingegen grundsätzlich analoge Schaltungen, deren Entwurf durch *Full-Custom-Design* geschieht. Dieser kann durch die Abhängigkeit von der Technologie sowie der Berücksichtigung von Offset, *Mismatch* oder Rauscheinflüssen sehr aufwändig sein. Die Festlegung auf eine bestimmte Technologie erschwert zudem eine mögliche Skalierung.

2.1.5 Ausführbare Operationen und Anwendungen

Während *Vision Chips* mit festem Funktionsumfang i.d.R. für einen konkreten Anwendungsbereich entwickelt wurden, ist dies für die in der angegebenen Literatur beschriebenen, programmierbaren Systeme i. Allg. nicht der Fall. Vielmehr wurden elementare Bildverarbeitungsoperationen ausgewählt und in den PE umgesetzt. Je nach Architektur und Funktionsumfang der PE können z.T. sehr unterschiedliche Bildverarbeitungsoperationen abgebildet werden. In Tabelle 2-2 sind verschiedene *Vision Chips* bzgl. der ausführbaren Operationen gegenübergestellt.

Die meisten in der Literatur dargestellten *Vision Chips* sind als Prototypen realisiert, weshalb nur wenige Anwendungsbeispiele existieren. So wird von Zarándy et al. [ZDCE02] der CNN-basierte Sensor Ace400 zur Erkennung bekannter Objekte auf einer schnell rotierenden Scheibe eingesetzt. Durch eine aufeinanderfolgende Sequenz von Entscheidungen (z.B. Rundheit und Dicke eines Objektes, sich kreuzende Linien) wird eine Klassifikation herbeigeführt. In einem zweiten Beispiel wird der Sensor zur Beobachtung von Zündfunken an Zündkerzen verwendet, wobei reguläre von sich verzweigenden Fun-

ken unterschieden werden. Die Erkennung von Zündfunken wird später von Zarándy und Rekeczky [ZR05] mit dem Sensor Ace16k erneut umgesetzt, zudem wird als weiteres Beispiel die Detektion von Fehlern in Textil-Etiketten angeführt. Die Testaufnahmen werden dabei mit Referenzbildern anhand mehrerer, durch den Sensor extrahierter, charakteristischer Punkte verglichen. Das kommerzielle Kamerasystem Eye-RIS wird von Bakkali et al. [BCGRV10] zur Erkennung von Bewegung verwendet, wobei lediglich aktive Bereiche ausgegeben werden und somit eine Datenreduktion erreicht wird. Fernandez-Berni et al. [FBCGCG$^+$10] bestimmen unter Verwendung von Eye-RIS Kandidaten-Regionen zur Erkennung von Rauch, wobei ein zuvor von der gleichen Gruppe vorgestellter Algorithmus [FBCGCG$^+$08] eingesetzt wird. Dieser dient auch in einer weiteren Arbeit [FBCGLC$^+$11] als Anwendungsbeispiel zur Demonstration des *Vision Chips* FlipQ.

Basierend auf SCAMP-3 suchen Carey et al. [CBD11] nach Veränderungen in der beobachteten Szene, wobei vom Sensor zunächst ein globales Maß für den Grad der Veränderungen berechnet und ausgegeben wird. Ein vollständiges Bild wird nur ausgelesen, wenn dieses Maß einen Schwellwert übersteigt. Barr et al. [BCD12] zeigen zudem die Zählung einfacher Objekte mit SCAMP-3 unter Verwendung von Binarisierung, Differenzbildern und globaler Fülloperationen („*Flood Fill*“). In einer weiteren Arbeit [CLB$^+$13] wird SCAMP-5 zur Beobachtung einer schnell rotierenden Scheibe eingesetzt, auf der sich mehrmals der Buchstabe „U“ sowie einmal der Buchstabe „O“ befinden. Unter Verwendung des asynchronen Propagations-Netzwerks werden durch globale Fülloperationen beide Zeichen unterschieden und die Trajektorie des Zeichens „O“ mit 100 kHz ausgegeben.

Von Lahdenoja et al. [LPL10] wird der oft zur Texturerkennung eingesetzte LBP-Operator (*Local Binary Pattern* [PHZA11]) direkt im *Vision Chip* MIPA4k berechnet, wodurch auf die Ausgabe von Bilddaten verzichtet werden kann.

2.1.6 Klassifikation nach verschiedenen Kennzahlen

Vision Chips können auch bzgl. verschiedener Kennzahlen klassifiziert werden, was in Tabelle 2-3 dargestellt ist. Die meisten der in der Literatur vorgestellten Sensoren sind als Prototyp konzipiert, weshalb diese lediglich über eine verhältnismäßig geringe Auflösung verfügen. Sensoren, die jedem Pixel ein eigenes PE zuordnen (vgl. Tabelle 2-1), besitzen deutlich größere Pixel als Sensoren mit spaltenparalleler Anordnung der PE. Gleichzeitig ist der Füllfaktor aufgrund der für die Verarbeitungsfunktionen im Pixel erforderlichen Transistoren deutlich geringer als bei spaltenparallelen Systemen.

2.2 Modellierung und Simulation von Bildsensoren

2.2.1 Verhaltensbasierte Modellierung

Klassische Bildsensoren transformieren das einfallende Licht zunächst in eine intensitätsabhängige, elektrische Repräsentation. In der Regel erfolgt daraufhin ein zeilenweises Auslesen mit exaktem, für jedes Pixel identischem Zeitverhalten, die Umwandlung in die digitale Domäne und schließlich die Ausgabe der eigentlichen Bildinformation.

Arbeit	HDR Bildaufn.	Binäroperationen	Grauwertop.	Differenzbilder	Mittelwert	Filter-/Faltungsop.	regionenbas. Auslesen	globale Füllop.
ACE16k [RVLCC$^+$04, LEDCRV02b]	+	(+)	+	+	+	+	(+)	+
Q-Eye [Ana12, RVDCJG$^+$10]	+	+	+	+	+	+	(+)	+
SCAMP-1 [DH01]	-	-	+	+	+	+	+	-
SCAMP-3 [Dud05, Dud11]	-	-	+	+	+	+	+	-
SCAMP-5 [CLB$^+$13, CBW$^+$12]	-	+	+	+	+	+	+	+
MIPA4k [PLP09, Zar11]	-	+	+	(+)	(+)	+	(+)	-
Flip-Q [FBCGCG11]	-	-	-	-	+	-	+	-
Ginhac et al. [GDHP10]	-	-	-	-	(+)	Nachbarn	-	-
ASPA-3 [LD13]	-	+	+	+	+	+	+	+
S^{3}PE [KKI04b, IK01]	-	+	+	+	+	+	-	-
PARIS1 [EBD$^+$06, DKN02]	-	+	+	+	Spalte	Spalte / Nachbarn	(+)	-
Lindgren et al.[LMJM05]	-	+	+	-	Spalte	Spalte / Nachbarn	+	-
Verdant et al. [VDV$^+$13]	-	+	+	-	Makropixel	-	+	-
Zhang et al. [ZFW11]	-	+	+	+	+	+	+	-
Döge et al. [DHRP15]	+	+	+	+	Spalte	Spalte / Nachbarn	+	-

Tabelle 2-2: Gegenüberstellung verschiedener *Vision Chips* hinsichtlich ausführbarer Operationen (- nicht ausführbar; + abbildbar / ausführbar; (+) keine genauen Angaben, aus Architekturbeschreibung extrahiert).

Arbeit	Prozess [µm]	Auflösung [Pixel]	Pixelgröße [µm²]	Füllfaktor [%]	Anzahl Trans.
ACE400 [DCERV+97]	0,8	20 × 22	190 × 190		
ACE4k [LEDCRV02a]	0,5	64 × 64	102,2 × 120		172
ACE16k [RVLCC+04, LEDCRV02b]	0,35	128 × 128	73,3 × 75,7	9,3	198
Q-Eye [Ana12, RVDCJG+10]	0,18	176 × 144	29,1 × 29,1		
SCAMP-1 [DH01]	0,6	21 × 21	98,6 × 98,6	8,4	128
SCAMP-3 [Dud05, Dud11]	0,35	128 × 128	49,4 × 49,4	5,6	111
SCAMP-5 [CLB+13, CBW+12]	0,18	256 × 256	32,3 × 32,3	6,2	176
MIPA4k [PLP09, Zar11]	0,13	64 × 64	72 × 61		1500
Flip-Q [FBCGCG11]	0,35	176 × 144	34,1 × 29,1	6,45	20
Ginhac et al. [GDHP10]	0,35	64 × 64	35 × 35	25	38
ASPA-3 [LD13]	0,18	160 × 80	54 × 51	5,5	588
S³PE [KKI04b, IK01]	0,35	64 × 64	67,4 × 67,4		400
PARIS1 [EBD+06, DKN02]	0,8	16 × 16	50 × 50	11	
Lindgren et al.[LMJM05]	0,35	1536 × 512	9,5 × 9,5	60	
Verdant et al. [VDV+13]	0,35	240 × 110	10 × 10	50	
Zhang et al. [ZFW11]	0,18	128 × 128	9 × 9	58	800
Döge et al. [DHRP15]	0,18	1024×1024	8,75 × 8,75	50	12

Tabelle 2-3: Gegenüberstellung verschiedener *Vision Chips* anhand von Kennzahlen. Da nicht für alle Sensoren vollständige Angaben verfügbar sind, wurden einige Tabelleneinträge leer gelassen.

Für den entsprechenden Signalfluss (siehe Abb. 2-4) kann eine rein verhaltensbasierte Modellierung erfolgen, wobei der Fokus auf einer möglichst exakten, mathematischen Beschreibung liegt. Dabei wird die radiometrische Repräsentation einer beobachteten Szene durch eine optische Übertragungsfunktion auf die Sensoroberfläche abgebildet, was der orts- und zeitabhängigen Bestrahlungsstärke $E_{e,x,y}(\lambda,t)$ entspricht. Zur Vereinfachung wird von senkrechtem Lichteinfall ausgegangen. Durch Lösung der Kameragleichung [Che03, CVB+09] (Gl. 2-1)

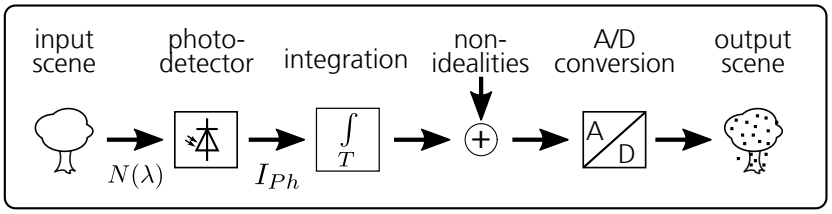


Abbildung 2-4: Verhaltensbasierte Modellierung konventioneller Bildsensoren.

$$Q_l = \int_0^{t_{\text{int}}} \underbrace{e \iint_{A_\text{D}} \int_{\lambda_{\min}}^{\lambda_{\max}} E_{\text{e},x,y}\,(\lambda,t)\, s_{l,x,y}\,(\lambda)\; d\lambda\, dA}_{=:i_{\text{ph},l}(t)}\; dt \tag{2-1}$$

für jedes Pixel l, erfolgt die durch die Fotodetektoren durchgeführte Transformation in eine korrespondierende elektrische Ladung. Dabei ist e die Elementarladung, $s_{l,x,y}\,(\lambda)$ die ortsabhängige spektrale Empfindlichkeit von Pixel l, A_D die Fläche des Fotodetektors und Q_l die im Pixel l während der Belichtungszeit t_{int} akkumulierte Ladungsmenge. Diese wird im realen System sowohl durch unterschiedliche, zeitabhängige Störgrößen als auch durch zeitunabhängige Nicht-Idealitäten überlagert. Im Modell werden dazu entsprechende Zufallsprozesse [GRF+07] realisiert, deren Charakteristik durch empirisch bestimmte oder angenommene Parameter festgelegt wird.

Von Chen [Che03] wird der Simulator vCam vorgestellt, der als *Toolbox* für MATLAB realisiert wurde und die Simulation eines vollständigen, digitalen Kamerasystems ermöglicht. Zur Modellierung der von der Kamera beobachteten Szene werden Eingabedaten mit einer hohen örtlichen Auflösung vorausgesetzt. Für jeden diskreten Punkt der Szene steht, neben der Anzahl der pro Zeiteinheit emittierten Photonen, auch deren spektrale Zusammensetzung zur Verfügung. Dazu können entweder RGB-Bilder mit lediglich 3 Stützpunkten bei 450 nm (blau), 550 nm (grün) bzw. 650 nm (rot) verwendet werden, oder Hyperspektralbilder, die als dreidimensionales Array (x,y,λ) aufgefasst werden können und die ortsabhängige Intensität somit auch bzgl. der Wellenlänge auflösen. Ein weiteres Szenenformat verwendet lediglich monochromatische Bilder, wobei der Helligkeitswert eines Bildpunktes als Reflektanz interpretiert wird. Verschiedene, in die Szene eingebrachte Lichtquellen unterschiedlicher Wellenlänge sorgen dabei für die notwendige Beleuchtung. Ein Modell der Optik transformiert die Helligkeitsinformation der Szene schließlich in eine Bestrahlungsstärke auf der Sensoroberfläche, wobei auch Parameter der Optik berücksichtigt werden. Das Modell des Sensors umfasst zunächst die Geometrie der Pixelzellen. Neben der reinen, fotoempfindlichen Fläche wird hierbei auch der Einfluss winkelabhängiger Abschattungen, bedingt durch den Aufbau der Metalllagen oberhalb der Pixelstruktur, berücksichtigt. Nach der Lösung der Kameragleichung, bei der auch die möglicherweise durch Farbfilter eingeschränkte spektrale Empfindlichkeit berücksichtigt wird, steht für jedes Pixel die Anzahl der im Integrationsintervall gesammelten Elektronen bereit. Nicht-Idealitäten und Rauscheinflüsse werden erst nach der

Integration berücksichtigt. Durch Chen et al. [CVB+09] wird das Modell der Optik deutlich erweitert und das Sensormodell um die Behandlung von Übersprechen (*crosstalk*) zwischen benachbarten Pixeln ergänzt. Außerdem wird eine Bibliothek mit Hyperspektralbildern vorgestellt, bei der für jeden Bildpunkt die wellenlängenabhängige Reflektanz hinterlegt wird, was die Beleuchtung mit unterschiedlichen Lichtquellen ermöglicht. Farrell et al. [FOP08, FCW12] stellen das kommerziell verfügbare *Image Systems Evaluation Toolkit* (ISET) vor, das ebenfalls als MATLAB *Toolbox* realisiert wurde und eine Abschätzung der Bildqualität bei Variation von Sensor-Parametern ermöglicht. Gow et al. [GRF+07] verwenden synthetische, rauschfreie Szenen und führen eine sehr ausführliche Beschreibung verschiedener Rauschquellen in CMOS Bildsensoren und deren Modellierung an. Costantinie und Süsstrunk [CS04] setzen hingegen auf die Positionierung von Lichtquellen verschiedener Wellenlängen, wobei den Objekten der Szene eine wellenlängenabhängige Reflektanz zugeordnet wird. Kolehmainen et al. [KAK+04] modellieren sowohl den Bildsensor als auch die abbildende Optik und setzen einfache RGB-Bilder als Szenenmodell ein.

Bei der verhaltensbasierten Modellierung werden weder die Struktur des Sensors noch dessen einzelne Komponenten oder deren Ansteuerung betrachtet. Stattdessen wird lediglich der Signalfluss für ein konkretes Ausleseschema berücksichtigt, um unter Verwendung der aktuellen Parametersätze eine Vorhersage der erzielbaren Bildqualität zu erreichen. Für Sensoren mit integrierter, programmierbarer Signalverarbeitung, eignet sich dieses Vorgehen daher nicht. Die beschriebenen Simulatoren unterstützen zudem ausschließlich statische Szenen, d.h. $E_{\mathrm{e},x,y}(\lambda)$ ist zeitunabhängig, und es werden ausschließlich Einzelbilder, ohne die Berücksichtigung zeitabhängiger Effekte des Ausleseprozesses, ausgewertet.

2.2.2 Strukturtreue Modelle

Bei strukturtreuen Modellen erfolgt der Aufbau des Gesamtsystems unter Verwendung von Modellen der einzelnen Komponenten sowie deren Schnittstellen untereinander. Neben der Ansteuerung wird auch das zeitliche Verhalten berücksichtigt und die Auswirkungen auf die Signalverarbeitung betrachtet. Die bedingt durch das Zusammenspiel der Sub-Komponenten auftretenden Wechselwirkungen können somit sichtbar gemacht werden [Bos04].

Zur Beschreibung komplexer Systeme wird im EDA-Bereich oft auf die Klassenbibliothek SystemC [IEE12] zurückgegriffen, die die Modellierung auf verschiedenen Entwurfsebenen in C++ erlaubt. Neben einem ereignisbasierten Simulationsmodell werden auch nebenläufige Prozesse sowie Methoden für deren Kommunikation und Interaktion bereitgestellt. Bjørnsen et. al. [BBY03] stellen ein Konzept zur Erweiterung von SystemC zur Modellierung von Mixed-Signal Systems-on-Chip (SoC) vor. Als Implementierungsbeispiel wird ein einfaches Bildsensor-SoC modelliert, welches die speziellen *Channels* zur Repräsentation von Strom, Spannung oder Lichtstärke verwendet.

Verdant et al. [VVDM07] beschreiben die strukturtreue Modellierung eines *Vision Chips*, mit integrierter, analoger Vorverarbeitung. Die Pixel-Matrix besteht aus Stan-

dard-3T Pixelzellen[1], die jeweils zu Makropixeln zusammengefasst sind. Die von diesen beobachtete Szene liegt als Bildfolge vor. Mit Hilfe des auf *Switched-Capacitor-Arrays* basierenden Analog-Prozessors, dessen Ansteuerung durch ein C++ Programm von außen erfolgt, wird eine Bewegungsdetektion anhand der Änderung von Bildinhalten implementiert. Zur Datenkompression werden lediglich die Regionen, in denen eine Bewegung erkannt wurde, mit voller Auflösung ausgegeben. Alle Bestandteile des Systems, bis hin zu einzelnen Kondensatoren, wurden als Module in SystemC realisiert und erlauben auch eine Abschätzung der elektrischen Leistungsaufnahme. Der implementierte Signalfluss berücksichtigt dabei sowohl zeitlich invariante als auch zeitabhängige Nicht-Idealitäten.

Blanchard et al. [BDP11] beschreiben ebenfalls eine SystemC basierte Methodik zur Simulation von Bildsensoren, die speziell von deren regulärer Struktur profitiert. Um die Simulationszeiten deutlich zu reduzieren, erfolgt zunächst eine Gruppierung mehrerer Pixel z.B. einer Zeile in Blöcke. Ein Block fasst dabei die Pixel zusammen, für die gleiche Berechnungen ausgeführt werden müssen. Anstelle periodischer Berechnungen mit vorher festgelegten Zeitschritten wird als weitere Optimierung eine Berechnung nur bei Bedarf durchgeführt, d.h. wenn das entsprechende Ergebnis tatsächlich benötigt wird. Zum Beispiel findet die Integration im Pixel im Zeitraum zwischen Rücksetzen und Auslesen erst bei der eigentliche Ausleseoperation statt. Als Beispiel wurde ein einfaches CMOS *Active Pixel Array* (APS) modelliert.

Basierend auf SystemC-AMS [OSC10] modellieren Cenni et al. [CSS11b, CSS11a, CSS11c] ein APS unter Verwendung des *Timed-Data-Flow* (TDF) Berechnungsmodells. Der Signalfluss, ausgehend von einer synthetischen Szene, wird unter Berücksichtigung der Optik auf ein Array von Pixeln abgebildet. Das Modell des Sensors wird in ein SoC integriert und von diesem angesteuert. Verschiedene Optimierungen hinsichtlich der Ausführungsgeschwindigkeit werden aufgezeigt, wobei jedoch die Strukturtreue des Modells immer mehr verloren geht.

Basierend auf der Topologie der SCAMP *Vision Chips* SCAMP-3 [Dud05] wurde mit APRON [BD08, BD09] ein System zur effektiven Simulation von 2D *Cellular-Processor-Arrays* (CPA) gezeigt. Der Simulator erlaubt die Ausführung von Bildverarbeitungsalgorithmen auf der jeweils modellierten Prozessor-Architektur. Der für alle Verarbeitungseinheiten gleiche Kontrollfluss wird durch einen zur Entwicklungsplattform kompatiblen, integrierten Mikrocontroller vorgegeben. Die modellhafte Integration der Simulation in ein vollständiges Bildverarbeitungssystem ist jedoch nicht vorgesehen.

2.3 Ansteuerung und Programmierung von Vision Chips

Der Einsatz eines programmierbaren *Vision Chips* erfordert eine geeignete Kamera- bzw. Entwicklungsplattform, die sowohl die für den Betrieb des *Vision Chips* erforderliche Infrastruktur bereitstellt als auch das Verhalten der Hardware zur Implementierung von

[1] 3T-Pixelzellen [Oht10] besitzen einen einfachen Aufbau mit lediglich drei Transistoren zum Rücksetzen des Pixels, zur Signalverstärkung sowie zur Selektion während der Ausgabe. Durch den fehlenden Speicher ist weder eine synchrone Belichtung der Pixelmatrix noch eine Korrektur des Schwarzwertes nach dem Rücksetzen möglich.

Algorithmen in Software abstrahiert. Für einen Großteil der in der Literatur beschriebenen, programmierbaren *Vision Chips* wird das erforderliche Steuerwerk extern realisiert (siehe auch Abschnitt 3.2.1). Aufgrund der großen Flexibilität geschieht dies meist mit Hilfe eines FPGA (*Field Programmable Gate Array*). Lediglich das System von Zhang et al. [ZFW11] besitzt ein integriertes Steuerwerk und verfügt, neben einem 8051-basierten Prozessor zur Steuerung sowie zur seriellen Nachverarbeitung, auch über einen separaten Controller zur Ansteuerung der parallelen SIMD-Recheneinheiten. Zur Programmierung wird ein spezieller, um parallele Konstrukte erweiterter C-Compiler eingesetzt. Es werden verschiedene elementare Bildverarbeitungsalgorithmen, wie z.B. Differenzbilder, Kanten-Detektion und Bewegungserkennung sowie diverse Filter-Operationen (Median, FFT), abgebildet und hinsichtlich ihrer Ausführungszeit untersucht.

Das CNN Chip Prototyping System (CCPS) [RZZ+99] wurde zur Ansteuerung verschiedener CNN (*Cellular Neural Network*) basierter Bildsensoren wie z.B. dem Ace400 [DCERV+97] entwickelt und dient sowohl zum Test der CNN-Chips als auch zur Ausführung von Bildverarbeitungsalgorithmen unter Verwendung der bereitgestellten PE. Um die Komplexität der zugrundeliegenden CNN-Chips zu verbergen, führt das System sowohl Hardware- als auch Software-basierte Abstraktionsebenen ein. Eine für jeden CNN-Chip individuell angefertigte Adapter-Platine dient zur Anbindung des eingesetzten CNN-Chips an die Schnittstellen der eigentlichen Entwicklungsplattform. Die Steuerung der Plattform und des CNN-Chips erfolgt mit einem DSP (*Digital Signal Processor*), der in der proprietären Assemblersprache „AMC“ vorliegende Programme mit Hilfe eines Interpreters ausführt. Durch AMC werden abstrakte Operationen auf Ebene der CNNs, wie z.B. das Laden verschiedener Gewichtsmatrizen oder deren aufeinanderfolgende Auswahl zur Abarbeitung eines Algorithmus, bereitgestellt. AMC-Operationen, die nicht auf die PE des CNN-Chips abgebildet werden können, werden stattdessen im DSP emuliert. Das Verhalten der CNNs wird durch die ebenfalls proprietäre, domänenspezifische Programmiersprache „ALPHA“ weiter abstrahiert, die mit Hilfe eines Compilers in AMC übersetzt wird.

„ALADDIN“ bzw. „ACE box“ [Zar01] repräsentiert die Weiterentwicklung von CCPS und ermöglicht die Integration in Industrie-PCs zum Aufbau eines Kamerasystems. Zunächst für ACE4k [LEDCRV02a] entwickelt, wurde ALADDIN später [ZRFS03] auch für den Einsatz von ACE16k [RVLCC+04, LEDCRV02b] erweitert. Programmierung und Datentransfer zum PC erfolgen über den PCI-Bus. Neben der Programmierung in ALPHA bzw. AMC ist auch eine Schnittstelle zur Programmiersprache C vorhanden. Eine Bibliothek mit elementaren Bildverarbeitungsfunktionen [SFRZ02] kann sowohl in C-, als auch in ALPHA-Programmen verwendet werden und versteckt die Details der CNN basierten Bildverarbeitung. Diese umfasst Funktionen zur pixelweisen Verarbeitung (Logik-Operationen, Addition/Subtraktion, Binarisierung), diverse Statistik-Funktionen (Mittelwert, Median, Varianz) sowie Funktionen zur Extraktion von Merkmalen und zur Filterung der aufgenommenen Bilddaten.

Von Zarándy et al. [ZRS03] wird ein auf ALADDIN basiertes System vorgestellt, welches neben dem *Vision Chip* ACE16k, auch über einen gewöhnlichen, monochromen

CMOS APS verfügt. Dieses wird später [ZR05] um einen wesentlich schnelleren DSP sowie die Unterstützung von Farbsensoren erweitert.

Von Carranza et al. [CJGLC+05] wird ein FPGA-basiertes Entwicklungssystem speziell für den CNN-Chip ACE16k vorgestellt, welches einen autonomen Betrieb ohne PC ermöglicht. Die Programmierung eines im FPGA realisierten Steuerwerks erfolgt ausschließlich in einer speziellen Assemblersprache. Die Datenausgabe erfolgt direkt auf einem Display, eine externe Weiterverarbeitung ist nicht vorgesehen.

Der kommerzielle und ebenfalls CNN-basierte Bildsensor „Q-Eye“ [Ana12, RVDCJG+08] stellt eine Weiterentwicklung des ACE16k dar und bildet die Grundlage des Bildverarbeitungssystems „Eye-RIS“ [Ana08]. Die Ansteuerung des Sensors erfolgt ähnlich zu ACE16k unter Verwendung eines im FPGA realisierten Nios-II Prozessors [Nio09]. Rodríguez-Vázquez et al. [RVDCJG+10] stellen zudem eine integrierte Variante vor, bei der Q-Eye gemeinsam mit dem Nios-II Prozessor als SoC auf dem gleichen Chip realisiert wird. Bzgl. der zugehörigen integrierten Entwicklungsumgebung (*Integrated Development Environment*, IDE) sind nur sehr wenige Informationen verfügbar. Diese enthält, neben den Werkzeugen zur Programmierung und zum Bildeinzug, auch eine Bibliothek mit CNN-basierten Bildverarbeitungsfunktionen, die u.a. Punkt-, Filter- und morphologische Operationen sowie die Extraktion von Merkmalen bereitstellt.

Zur Ansteuerung des *Vision Chips* SCAMP-3 [Dud11] wurde von Barr et al. [BCLD06] ein externes Steuerwerk vorgestellt. Ein im FPGA realisierter 8 Bit Mikrocontroller dient zur Implementierung von Algorithmen und erzeugt die für SCAMP-3 notwendigen Steuerworte. Durch die Verbreiterung des Programmspeichers werden diese gemeinsam mit den eigentlichen Instruktionen des Mikrocontrollers abgelegt und somit im Zuge des regulären Programmablaufs automatisch ausgegeben. Carey et al. [CBD13] beschreiben eine Plattform, die insbesondere für mobile Anwendungen entwickelt wurde. Neben einem ARM-Mikrocontroller dient ein FPGA als Decoder und somit zur eigentlichen Ansteuerung von SCAMP-3. Zur Programmierung wird die Entwicklungs- und Simulationsumgebung APRON [BD08, BD09] verwendet. Die IDE bindet, neben dem bereits erwähnten Simulator, auch die o.g. Test-Umgebungen ein. Zur Programmierung dient eine spezielle Assemblersprache, die in die zur Ansteuerung von SCAMP notwendigen Steuerworte sowie in den vom Mikrocontroller verarbeiteten Kontrollfluss übersetzt wird.

Komuro et al. realisieren für ihre *Vision Chips* [IK01] mit bitseriell PE die Plattform „VCS-IV“ [KKI04a], die durch einen eigenen, im FPGA realisierten Prozessor [KKII02] gesteuert wird. Dieser besitzt separate Pipelines für Integer und SIMD-Verarbeitung, wobei letzteres der Ansteuerung des *Vision Chips* entspricht. Zur Programmierung wurde ein Compiler [KKIK05] entwickelt, der die Programmiersprache C um parallele Datentypen zur Abbildung auf dem *Vision Chip* erweitert und die Transformation auf die Bit-serielle Verarbeitung der PE durchführt. Eine weitere Auswertung der extrahierten Features durch ein Nachverarbeitungssystem ist jedoch nicht vorgesehen.

Externe FPGAs zur Steuerung des *Vision Chips* werden auch von Ginhac et al. [GDHP10], Fernández-Berni et al. [FBGRRV14, FBCGR+14] sowie für MIPA4k [PLP09] eingesetzt. Hingegen setzt die Kamera-Plattform Wi-Flip [FBCGLC+11] basierend auf dem *Vision Chip* Flip-Q [FBCGCG11], ein für Funk-Anwendungen entwickeltes, kom-

merziell verfügbares Mikrocontroller-System ein. Als Schnittstelle zum Sensor wird der IO-Port des Prozessors genutzt, wobei die horizontalen und vertikalen Schieberegister durch die Ausgabe von Impulsen geladen werden. Die Impulserzeugung erfolgt direkt aus der Software, wodurch nur eine sehr geringe Auslesegeschwindigkeit erzielt und die Möglichkeiten des Sensors nicht voll ausgenutzt werden können.

Der *Vision Chip* PARIS1 [DKN02] wird als Peripherie-Komponente an einem Mikrocontroller betrieben und über dessen Systembus in den Adressraum der CPU eingebunden. Die Entwicklung von Software erfolgt in C/C++ mit Hilfe bereitgestellter Makros für die von den PE bereitgestellten Basisoperationen. Diese umfassen, unter anderem, die Selektion einer Zeile zur Ausgabe auf die Spaltenleitung sowie die Ansteuerung der analogen Verarbeitungseinheit. Alle Instruktionen werden vom Mikrocontroller über die gleiche Schnittstelle zu PARIS1 übertragen und sequentiell ausgeführt. Als Beispiel wird die Anwendung eines Faltungskerns zur Kanten-Extraktion dargestellt.

2.4 Anforderungen

Als Basis zur Entwicklung des auch im Rahmen dieser Arbeit betrachteten *Vision-System-on-Chip* [DHRP15] wurde die von Jens Döge vorgestellte, ladungsbasierte Schaltungstechnik [Dög08] ausgewählt. Im Gegensatz zu den meisten in der Literatur angegebenen *Vision Chip*-Konzepten soll das VSoC nicht als reine Machbarkeitsstudie der Technologie dienen, sondern den Einsatz in tatsächlichen Anwendungen ermöglichen. Neben einer hohen Auflösung und einem großen Füllfaktor (vgl. Tab. 2-3) soll eine frei programmierbare Bildaufnahme und -verarbeitung sowohl in der analogen Domäne (z.B. Faltungsoperationen oder Differenzbilder) als auch in der digitalen Domäne möglich sein. Außerdem ist ein spalten-paralleles SIMD-Prozessor-Array sowie eine, bzgl. der Auflösung, frei konfigurierbare A/D-Umsetzung vorgesehen.

Zum Betrieb eines *Vision Chips* sind, neben den Prozessor-Elementen, die zur eigentlichen Bildverarbeitung und Merkmalsextraktion dienen, eine Vielzahl weiterer Hilfskomponenten notwendig. Im Folgenden soll die Gesamtheit der Hilfskomponenten und der PE allgemein als Funktionseinheiten (*Functional Unit*, FU) bezeichnet werden. Einige FU besitzen untereinander Abhängigkeiten, die bei deren Ansteuerung berücksichtigt werden müssen, andere FU können hingegen parallel und unabhängig voneinander arbeiten. Beispielsweise muss die Ausgabe der Pixel vor der A/D-Umsetzung abgeschlossen sein, die Belichtung eines Folgebildes kann jedoch bereits während der Verarbeitung des aktuellen Bildes erfolgen. Zur Verringerung der Komplexität bzgl. der Ansteuerung soll, im Gegensatz zu den in der Literatur beschriebenen, programmierbaren *Vision Chips*, ein integriertes Steuerwerkskonzept erarbeitet und im VSoC umgesetzt werden. Dabei sind insbesondere parallel arbeitende FU zu berücksichtigen, die ggf. Abhängigkeiten untereinander besitzen. Je nach Anwendung soll das Steuerwerk sowohl den energie- und flächeneffizienten Betrieb in Kombination mit einem Mikrocontroller, als auch die schnelle Ausgabe von Bild- und Merkmalsdaten sowie eine einfache Konfiguration und Parametrierung ermöglichen.

Programmierbare *Vision Chips*, die als „Bild-(Sensor-)Prozessoren“ aufgefasst werden können, erfordern neben dem Steuerwerk auch eine Umgebung zur Programmierung, ein Simulationssystem zum Test von Algorithmen unter reproduzierbaren Bedingungen sowie eine Entwicklungs- und Kameraplattform, die einen Einsatz zur Bewältigung tatsächlicher Aufgabenstellungen erlaubt. Da zur Lösung komplexer Bildverarbeitungsaufgaben mitunter nur ein Teil der erforderlichen Operationen direkt auf dem *Vision Chip* abgebildet werden kann, ist dessen Integration in konventionelle Bildverarbeitungssysteme erforderlich.

Sofern in der Literatur ein zum jeweiligen *Vision Chip* gehörendes Steuerwerk dargestellt wird, geschieht die Umsetzung von Algorithmen zumeist mit Hilfe eigener Assemblersprachen bzw. durch Bereitstellung von Makros in der Programmiersprache C. Außerdem wird lediglich für ALADDIN [Zar01, ZRFS03] eine Bibliothek elementarer Bildverarbeitungsfunktionen [SFRZ02] zur Unterstützung der Entwicklung beschrieben.

Während zur Modellierung und Simulation konventioneller Bildsensoren verschiedene Simulationsumgebungen [Che03, GRF+07, FOP08, FCW12] verfügbar sind, werden für *Vision Chips* nur sehr wenige Implementierungen [VVDM07, BD08, BD09] angegeben. Durch die erforderliche, gemeinsame Betrachtung von Bildaufnahme und -verarbeitung müssen, neben zeitabhängigen Effekten, auch der Einfluss von Defekten und Nicht-Idealitäten auf die integrierte Bildverarbeitung berücksichtigt werden. Jedoch wird dies von keinem der in der Literatur angegeben Systeme unterstützt.

Für mehrere der beschriebenen *Vision Chips* werden Plattformen zur Charakterisierung und Inbetriebnahme angegeben [RZZ+99, Zar01, DKN02, ZRFS03, KKI04a, ZR05, BCLD06, FBCGLC+11, CBD13]. Dabei werden jedoch nur der kommerzielle Sensor Q-Eye [Ana12] in der Kameraplattform Eye-RIS [RVDCJG+08, RVDCJG+10] sowie der von Lindgren et al. [LMJM05] vorgestellte Bildsensor in praktischen Anwendungen eingesetzt. Keines der Systeme ermöglicht zudem die Kombination der *Vision Chip*-basierten Vor- mit konventioneller Weiter- bzw. Nachverarbeitung.

Obwohl für manche *Vision Chips* einige der erforderlichen Teilaufgaben – bestehend aus Steuerungssystem, Programmierumgebung, Modellierung und Simulation, Entwicklungs- und Kameraplattform sowie Einbindung in Bildverarbeitungssysteme – durchaus erfüllt werden, findet sich für keines der Systeme eine durchgängige Lösung. Im Rahmen dieser Arbeit soll daher ein ganzheitliches Konzept zur Bereitstellung der zum Einsatz von *Vision Chips* notwendige Infrastruktur geschaffen werden. Dies erstreckt sich ausgehend vom Konzept eines integrierten, programmierbaren Steuerwerks, über eine Kamera- und Entwicklungsplattform zur Unterstützung der einfachen Integration in Bildverarbeitungsanwendungen, bis hin zu einem Simulationssystem zur Validierung implementierter Algorithmen unter Berücksichtigung von Nicht-Idealitäten und zeitabhängigen Effekten. Zudem sollen reale Bildverarbeitungsaufgaben definiert und unter Verwendung des entwickelten Konzepts umgesetzt werden.

3 Vision-System-on-Chip

3.1 Architekturkonzept

3.1.1 Aufbau und Struktur

In dem von Döge et al. [DHRP15] vorgestellten VSoC wird die ladungsbasierte Schaltungstechnik [Dög08] in einem Bildsensor mit integrierter Signalverarbeitung umgesetzt. Die Architektur des VSoC ist in Abb. 3-1 als Übersicht dargestellt. Neben der eigentlichen Pixel-Matrix und deren Ansteuerung besitzt dieses einen spaltenparallelen, mixed-signal Datenpfad, integrierte Takt- und Impulsgeneratoren sowie verschiedene Schnittstellen zur Datenein- und -ausgabe. Die Steuerung der einzelnen Komponenten erfolgt durch das integrierte, im Rahmen dieser Arbeit entwickelte, Multi-ASIP (*Application-Specific Instruction-set Processor*) basierte Steuerwerk. Dieses umfasst die drei ASIPs „GLOBALCTRL“, „LINECTRL“ und „SIMDCTRL“ zur Ansteuerung der Funktionseinheiten (*Functional Units*, FU). Während LINECTRL (ASIP 1 in Abb. 3-1) zur Ansteuerung der Pixelmatrix und somit der Bildverarbeitung in der analogen Domäne dient, übernimmt SIMDCTRL (ASIP 2) die Ansteuerung der mixed-signal PE des spaltenparallelen SIMD-Datenpfades. GLOBALCTRL (ASIP 3) unterstützt schließlich das für die Bildaufnahme und -verarbeitung erforderliche, zeitlich exakte Zusammenspiel der einzelnen Komponenten sowie die Kommunikation mit der Außenwelt. Das Steuerwerk wird in Abschnitt 3.3 genauer betrachtet. Im Folgenden sind wesentliche Funktionseinheiten des VSoC aufgelistet und kurz erläutert.

Pixel-Matrix

Die Pixel-Matrix umfasst 1024×1024 ladungsbasierte Zellen, deren vereinfachte Ersatzschaltung in Abb. 3-2 dargestellt ist. Jede Pixelzelle besitzt einen Foto-FET [KDJS87] (Feldeffekt-Transistor) als lichtempfindliches Element ($\mathrm{T_{Ph}}$ in Abb. 3-2), wobei die Bulk-Substratdiode $\mathrm{D_{Ph}}$ des Transistors als Foto-Diode wirkt. Die Kapazität der Diode $\mathrm{D_{Ph}}$ ist in Abb. 3-2 als Kondensator $\mathrm{C_{Ph}}$ dargestellt und dient zur Integration des Fotostromes $i_{\mathrm{ph}}(t)$ während der Belichtung. Zum Rücksetzen, d.h. zu Beginn einer neuen Belichtung, wird $\mathrm{C_{Ph}}$ durch den Schalter $\mathrm{S_{SetC}}$ (Signal PIX_SELSETC) auf das Reset-Potential V_{rst} aufgeladen. Durch Integration des Fotostromes sinkt das Potential ab, wodurch sich abhängig von der integrierten Ladungsmenge bzw. der resultierenden Spannung V_{BPh} ein entsprechender Drain-Strom $i_{\mathrm{DS}}(t)$ als aktueller Grauwert des Pixels einstellt. Die Spannung V_{BPh} wird zudem durch die Source-Bulk-Diode $\mathrm{D_{SB}}$ auf $V_{\mathrm{clamp=}} V_{\mathrm{DD}} - V_{\mathrm{f}}$ geklemmt, wobei V_{f} der Vorwärtsspannung der Source-Bulk-Diode entspricht und logarithmisch vom Fotostrom $i_{\mathrm{ph}}(t)$ abhängt. Wird auf das Rücksetzen verzichtet, so ist die Ausgabe $i_{\mathrm{DS}}(t)$

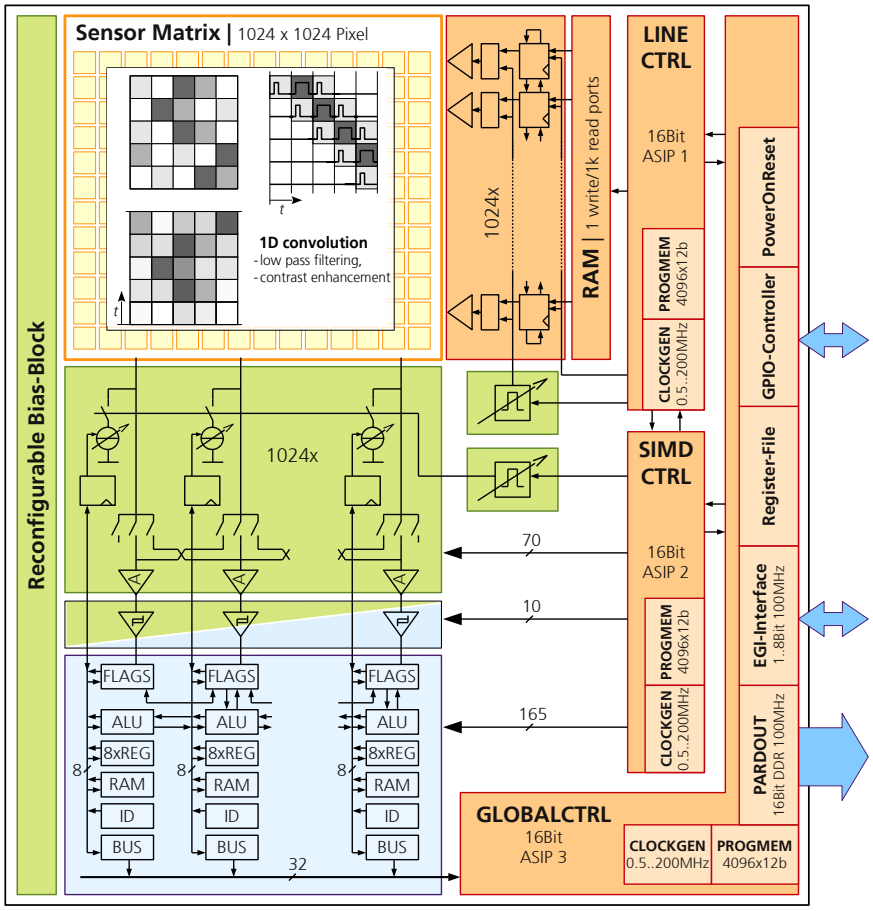


Abbildung 3-1: Übersicht über die Struktur des Bildsensor-SoC (Quelle: [DHRP15]).

somit logarithmisch vom aktuellen Fotostrom $i_{\mathrm{ph}}(t)$ abhängig, wodurch ein sehr hoher Dynamikumfang (*High Dynamic Range*, HDR) erreicht wird. Während durch V_{GPh} die Verstärkung des Foto-FET eingestellt wird, steuert das Signal PIX_PIXK die Ausgabe des aktuellen Pixelwertes auf den internen Knoten $\mathrm{N_I}$.

Jedes Pixel besitzt zudem eine Strom-Speicherzelle („SI-Zelle“), die das kurzzeitige Ablegen von Grauwerten durch Speicherung einer zum korrespondierenden Strom $i_{\mathrm{SI}} = i_{\mathrm{DS}}$ proportionalen Spannung mit Hilfe des Kondensators $\mathrm{C_{SI}}$ ermöglicht. Aufgrund von Leckströmen entlädt sich $\mathrm{C_{SI}}$ jedoch allmählich mit der Zeitkonstante $\tau_{\mathrm{SI,leak}}$. Sowohl der Foto-FET als auch die SI-Zelle sind durch den Ausgabeschalter $\mathrm{S_{IO}}$ mit der für alle Pixel einer Spalte gemeinsamen Spaltenleitung (IO in Abb. 3-2) verbunden. Wahlweise kann sowohl der aktuelle Grauwert $i_{\mathrm{IO}} = i_{\mathrm{DS}}$ als auch der Wert der SI-Zelle $i_{\mathrm{IO}} = -i_{\mathrm{SI}}$ ausgegeben werden. Bei gleichzeitiger Ausgabe ergibt sich der resultierende Strom zu $i_{\mathrm{IO}} = i_{\mathrm{DS}} - i_{\mathrm{SI}}$. Da die Richtung des in der SI-Zelle gespeicherten Stromes gleich bleibt, erfolgt während der Ausgabe eine Umkehrung des Vorzeichens.

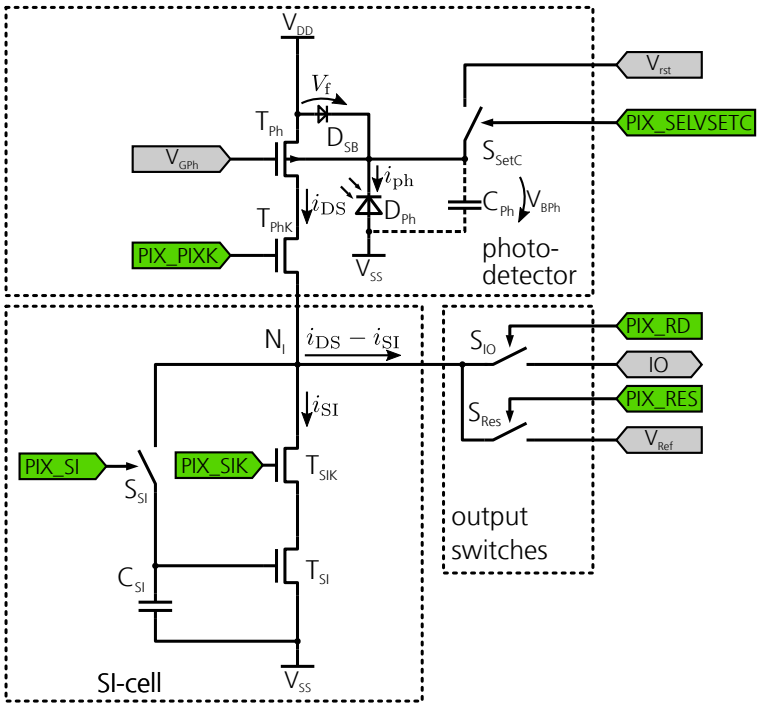


Abbildung 3-2: Vereinfachte Ersatzschaltung einer Pixelzelle bestehend aus Fotodetektor, SI-Zelle und Ausgabe.

Durch zeitliche Integration der von den Pixelzellen ausgegebenen Ströme ergibt sich auf den Spaltenleitungen eine sowohl vom Strom als auch von der Integrationszeit abhängige Ladung. Die Ansteuerung der Pixel-Matrix erfolgt durch Vorgabe der einzelnen Steuersignale (in Abb. 3-2 grün hervorgehoben) durch die Zeilensteuerung für alle Zellen einer Zeile gleich.

Mixed-Signal Prozessor

In jeder Spalte befinden sich 16 Kondensatoren, die mit der jeweiligen Spaltenleitung mit variabler Polarität verbunden bzw. auch abgekoppelt werden können. Durch Änderung ihrer Konfiguration während des Auslesevorgangs der Pixel-Matrix sind einfache Operationen in der analogen Domäne möglich (z.B. Differenz, Summe).

Die während der Ausgabe abgelegten Ladungspakete werden durch ein SIMD-Prozessor-Array mit mixed-signal PE weiterverarbeitet. Jedes PE besitzt einen 8 Bit Datenpfad mit eigener *Arithmetic-Logic-Unit* (ALU) sowie 24 Registern. Die Analog-Digital-Umsetzung (*Analog-to-Digital Conversion*, ADC) erfolgt durch die Ausgabe und zeitliche Integration definierter Ströme, wodurch die auf den Spalten vorhandene Ladung kompensiert wird. Ein Komparator, dessen Ergebnis im PE ausgewertet wird, dient zudem zum Vergleich des aktuellen Potentials der Spaltenleitung mit einer Referenz. Die Wertigkeit der ausgegebenen Kompensationsströme wird mit Hilfe der ALU aufsummiert, wodurch schließlich

eine digitale Repräsentation der kompensierten Ladungsmenge vorliegt. Der tatsächlich umgesetzte ADC-Algorithmus wird somit in Software realisiert.

Grundsätzlich führen alle PE jeweils die gleiche, durch SIMDCTRL vorgegebene Operation aus, jedoch erlaubt das bedingte Abschalten einzelner PE abhängig vom internen Zustand eine gewisse lokale Autonomie. Die in den PE berechneten Ergebnisse werden schließlich über einen gemeinsamen Bus ausgegeben.

Einstellbare Takt- und Impulsgeneratoren

Die Taktsignale, die zum Betrieb der als synchrone, digitale Schaltungen realisierten ASIPs erforderlich sind, werden durch konfigurierbare Oszillatoren direkt im VSoC bereitgestellt. Durch entsprechende Konfigurationsregister kann die Taktfrequenz jedes ASIPs im Betrieb individuell an die Erfordernisse eines konkreten Bildverarbeitungsalgorithmus angepasst werden.

Sowohl für die Ausgabe der aktuellen bzw. gespeicherten Pixelwerte als auch zur Ausgabe der Kompensationsströme während der A/D-Wandlung ist ein exakt definiertes und reproduzierbares Zeitverhalten erforderlich. Dazu steht jeweils ein Impulsgenerator bereit, der Impulse einer definierten Länge erzeugt und durch LINECTRL bzw. SIMDCTRL ausgelöst wird. Die Impulsgeneratoren arbeiten unabhängig vom jeweiligen ASIP und sind somit auch nicht abhängig von der eingestellten Taktfrequenz.

On-Chip Bias Erzeugung

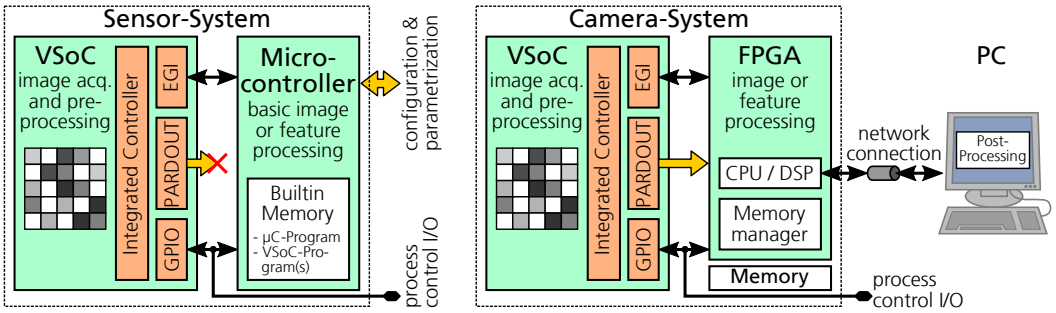
Zur Arbeitspunkteinstellung analoger Schaltungen sind einstellbare Strom- bzw. Spannungsquellen erforderlich. Entsprechende Konfigurationsregister ermöglichen deren Konfiguration auch im Betrieb.

Kommunikation mit der Außenwelt

Zur Kommunikation mit der Außenwelt stehen verschiedene externe Schnittstellen bereit, die durch GLOBALCTRL gesteuert werden. Mittels eines *General-Purpose Input / Output-Ports* (GPIO-Port), wird die Ansteuerung externer Sensoren und Aktoren sowie die Reaktion auf Ereignisse ermöglicht. Eine konfigurierbare, SPI-basierte Schnittstelle (EGI), erlaubt die Programmierung und Parametrierung des VSoC zur Laufzeit sowie die Übertragung von Bild- oder Merkmalsdaten [Pet13]. Verschiedene Erweiterungen, wie parallele MISO (*Master-In-Slave-Out*) oder MOSI (*Master-Out-Slave-In*) Kanäle, sowie ein vom Slave ausgegebener und vom Master empfangener Takt ermöglichen zudem eine gegenüber SPI deutlich gesteigerte Bandbreite von bis zu 100 $^{\mathrm{MiB}}/_{s}$. Für die besonders schnelle Ausgabe von Bild- oder Merkmalsdaten steht zudem ein paralleler Datenbus (PARDOUT) bereit, der einen Durchsatz von bis zu 400 $^{\mathrm{MiB}}/_{s}$ erlaubt.

3.1.2 Funktionsumfang

Im Gegensatz zu gewöhnlichen Bildsensoren, bei denen die Erzeugung qualitativ hochwertiger Bilder im Vordergrund steht, liegt der Fokus des entwickelten VSoC im Umfeld



(a) VSoC in Kombination mit einem Mikrocontroller für einfache Sensorsysteme.

(b) VSoC in Kombination mit einem FPGA sowie einem PC für komplexere Bildverarbeitungsaufgaben.

Abbildung 3-3: Darstellung verschiedener Anordnungen beim Einsatz des VSoC.

optischer Messaufgaben vor allem im industriellen Bereich. Zum Einsatz des VSoC in verschiedenen Anwendungsszenarien wurden zwei verschiedene Anordnungen definiert, die in Abb. 3-3 dargestellt sind. In beiden Fällen können externe Sensoren (z.B. eine Lichtschranke) und Aktoren (z.B. ein Beleuchtungssystem) ausgehend vom VSoC angesteuert werden, was eine sehr kurze Reaktionszeit und deren direkte Integration in die Bildaufnahme und -verarbeitungsalgorithmen ermöglicht.

Insbesondere für den energieeffizienten Einsatz in mobilen Anwendungen kann das VSoC in Kombination mit einem Mikrocontroller eingesetzt werden (siehe Abb. 3-3a). Das EGI (siehe 3.1.1) wird dabei als reguläre SPI-Schnittstelle betrieben und dient sowohl zur Parametrierung des VSoC als auch zum Transfer von Bild- und Merkmalsdaten. Die Bandbreite ist jedoch auf ca. 1 $^{\mathrm{MiB}}/_{\mathrm{s}}$ begrenzt. Sowohl die Konfiguration des Mikrocontrollers als auch verschiedene VSoC-Programme werden im internen Speicher des Mikrocontrollers abgelegt.

Insbesondere für industrielle Messaufgaben mit hohen Anforderungen bzgl. des Datendurchsatzes bzw. einer geringen Latenz sowie generell zur Entwicklung neuer Algorithmen, ist die wesentlich flexiblere Ansteuerung mit Hilfe eines FPGAs sinnvoll. Dabei wird das EGI im erweiterten Modus, d.h. mit mehreren MISO und MOSI-Leitungen betrieben, wodurch eine schnellere Konfiguration und Parametrierung des VSoC ermöglicht wird. Die extrahierten Bild- und Merkmalsdaten werden jedoch mit Hilfe der Schnittstelle PARDOUT zum FPGA übertragen, wo eine Weiterverarbeitung durch spezielle Hardware-Komponenten oder eine integrierte CPU stattfindet. Für komplexere Bildverarbeitungsaufgaben, die zusätzlich auch den Einsatz eines PC erfordern, werden die Bild- und Merkmalsdaten zunächst im Speicher abgelegt und schließlich über eine Netzwerkverbindung zum PC gesendet.

Je nach Ansteuerung der Pixelzellen (siehe 3.1.1) erfolgt die Bildaufnahme im linearen oder logarithmischen Betrieb. Letzterer ermöglicht einen sehr hohen Dynamikumfang und somit die Beobachtung kontrastreicher Szenen, wie sie z.B. bei der Laser-basierten Bearbeitung von Werkstücken auftreten. Das VSoC eignet sich somit nicht nur zur Merkmalsextraktion, sondern auch als regulärer HDR-Bildsensor mit hoher Bildwiederholrate.

Zur Bildverarbeitung in der analogen Domäne stehen Funktionen auf verschiedenen Ebenen bereit, die miteinander kombiniert werden können. So kann zunächst mit Hilfe der in jedem Pixel vorhandenen SI-Zelle sowohl ein Differenzbild als auch ein gleitender Mittelwert berechnet und ausgegeben werden. Durch gleichzeitige Ausgabe der Ströme mehrerer Pixelzellen ermöglicht die ladungsbasierte Signalverarbeitung Faltungsoperationen entlang einer Spalte. Durch Veränderung der Konfiguration der Kondensator-Bank ist zudem die Bildung von Summen bzw. Differenzen verschiedener aufeinander folgender Ausgabeoperationen möglich.

Nach der Analog-Digital-Umsetzung, die sowohl mit als auch ohne Vorzeichen sowie mit konfigurierbarer Auflösung erfolgt, kann eine digitale Weiterverarbeitung im SIMD-Prozessor-Array stattfinden. Arithmische und logische Operationen unter Einbeziehung der Nachbarspalten erlauben die Extraktion von Merkmalen, die Anwendung von Filter- und Korrektur-Operationen sowie die Vorbereitung der Daten zur Ausgabe.

3.1.3 Ladungsbasierte Bildaufnahme und -verarbeitung

Die eigentliche Pixel-Ausgabe erfolgt durch Integration des aktuellen Ausgabestromes $i_{\mathrm{DS}}(t)$ des Foto-FET sowie des in der SI-Zelle gespeicherten Stromes $i_{\mathrm{SI}}(t)$ mit jeweils definierten Zeiten auf die Kapazität der Spaltenleitung. Da die Richtung des Stromes i_{SI} erhalten bleibt, wird dessen Vorzeichen bei der Ausgabe jedoch umgekehrt. Durch Variation der Ausgabezeiten von aktuellem Grauwert (positives Vorzeichen) bzw. in der SI-Zelle gespeichertem Wert (negatives Vorzeichen) können Faltungsoperationen entlang einer Spalte durchgeführt werden. Die Integrationsdauer wird durch den zugehörigen Impulsgenerator vorgegeben. Der Beitrag der resultierenden Ladung eines Pixels l ergibt sich somit zu:

$$\Delta Q_l = \int_0^{t_{\mathrm{Pix,l}}} i_{\mathrm{DS},l}(t)\ dt - \int_0^{t_{\mathrm{SI,l}}} i_{\mathrm{SI},l}(t)\ dt \tag{3-1}$$

Die Ausgabezeit $t_{\mathrm{SI},l}$ der SI-Zelle ist gegenüber ihrer Haltezeit $\tau_{\mathrm{SI,leak}}$ sehr klein, weshalb $i_{\mathrm{SI}}(t)$ als konstant betrachtet werden kann. Je nach Anwendungsfall kann der Foto-Detektor im linearen oder logarithmischen Modus betrieben werden (siehe 3.1.1), wobei im logarithmischen Betriebsmodus i_{DS} direkt vom Fotostrom i_{ph} und somit vom einfallenden Licht abhängt. Im linearen Betriebsmodus ist hingegen die Auswahl einer zur Helligkeit der Szene passenden Belichtungs- bzw. Integrationszeit τ_{Expo} erforderlich. Für die korrekte Funktionsweise wird angenommen, dass die Ausgabezeit $t_{\mathrm{Pix},l}$ des Foto-Detektors sehr klein gegenüber dem gewählten τ_{Expo} ist. Befinden sich in der beobachteten Szene keine extrem hellen Regionen, so kann auch $i_{\mathrm{DS}}(t)$ für den kurzen Zeitraum $[0, t_{\mathrm{Pix},l}]$ als näherungsweise konstant betrachtet werden. Gleichung 3-1 vereinfacht sich somit zu:

$$\Delta Q_l = I_{\mathrm{DS},l} \cdot t_{\mathrm{Pix},l} - I_{\mathrm{SI},l} \cdot t_{\mathrm{SI},l} \tag{3-2}$$

Durch gleichzeitige oder aufeinanderfolgende Ausgabe mehrerer Pixel einer Spalte, lassen sich eindimensionale Faltungskerne der Größe K zu

$$Q = Q_0 + \sum_{l=0}^{K-1} \Delta Q_l \tag{3-3}$$

realisieren. Q_0 repräsentiert dabei die bereits vor der Faltungsoperation auf der Spaltenkapazität liegende Ladung. Mit für $I_{\mathrm{DS},l}$ und $I_{\mathrm{SI},l}$ jeweils unterschiedlichen Integrationszeiten $\underline{t_{\mathrm{Pix}}} = [t_{\mathrm{Pix,o}}, t_{\mathrm{Pix},1}, \ldots, t_{\mathrm{Pix},K-1}]$ bzw. $\underline{t_{\mathrm{SI}}} = [t_{\mathrm{SI,o}}, t_{\mathrm{SI},1}, \ldots, t_{\mathrm{SI},K-1}]$, kann die resultierende Ladung in Abhängigkeit der Ströme $\underline{I_{\mathrm{DS}}} = [I_{\mathrm{DS,o}}, I_{\mathrm{DS},1}, \ldots, I_{\mathrm{DS},K-1}]$ sowie $\underline{I_{\mathrm{SI}}} = [I_{\mathrm{SI,o}}, I_{\mathrm{SI},1}, \ldots, I_{\mathrm{SI},K-1}]$ nach Gl. 3-4 angegeben werden.

$$Q = Q_0 + \underline{I_{\mathrm{DS}}} \cdot \underline{t_{\mathrm{Pix}}}^{\top} - \underline{I_{\mathrm{SI}}} \cdot \underline{t_{\mathrm{Pix}}}^{\top} \tag{3-4}$$

Auch die Analog-Digital-Umsetzung beruht auf der zeitlichen Integration von Strömen auf die Kapazität der Spaltenleitung. Durch die dabei durchgeführte Kompensation ist die verbleibende Ladung nach der A/D-Wandlung nahe null.

3.1.4 Anvisierte Anwendungen

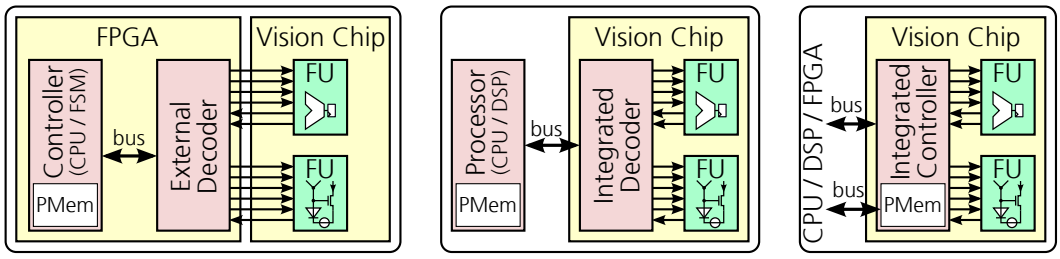
Die spaltenweise Anwendung von Faltungskernen während des Auslesens der Pixelmatrix ermöglicht die effiziente Implementierung von Algorithmen zur 3D-Rekonstruktion durch Laser-Triangulation. Bei diesen Verfahren muss die Position einer auf die Oberfläche des zu untersuchenden Objektes projizierten Laser-Linie in jeder Spalte möglichst exakt erkannt und ausgegeben werden. Durch die bekannte Anordnung von Laser und Kamera kann aus der Verschiebung der Linie auf die Geometrie der Oberfläche geschlossen werden. Die eigentlichen Bildinformationen sind dafür hingegen nicht notwendig. Durch die Berechnung der Profile im Bildsensor kann die Ausgabe auf relevante Informationen begrenzt und gleichzeitig eine sehr hohe Bildwiederholrate im Bereich mehrerer kHz realisiert werden.

Eine weitere Anwendung besteht in der Erkennung von Bewegungen in einer beobachteten Szene zur Präsenzdetektion, d.h. zur Lokalisation von Personen in Räumen. Die on-chip Extraktion von Merkmalen, wie LBP (*Local-Binary-Pattern* [PHZA11]) oder FAST (*Features from Accelerated Segment Test* [RD05, RD06]), ermöglicht, neben der deutlichen Vereinfachung der Weiterverarbeitung, auch die Sicherung der Privatsphäre durch den Verzicht auf vollständige Bilder. Die geringe erforderliche Außenbeschaltung erlaubt zudem den Betrieb ohne FPGA.

3.2 Steuerwerke für Vision Chips

3.2.1 Integrationsvarianten

In Abb. 3-4 sind die verschiedenen Integrationsvarianten eines Steuerwerks zur Ansteuerung der FU eines *Vision Chips* gegenübergestellt. Diese unterscheiden sich vor allem



(a) Vollständig externe Ansteuerung der Funktionseinheiten.

(b) Ansteuerung der FU durch einen integrierten Decoder sowie eine externe CPU.

(c) Vollständig integrierte Ansteuerung der FU.

Abbildung 3-4: Darstellung verschiedener Varianten zur Integration des Steuerwerks.

hinsichtlich der nach außen bereitgestellten Schnittstelle sowie der erforderlichen externen Beschaltung. In Tabelle 3-1 sind zudem sowohl die Integrationsvariante als auch die Realisierung des Steuerwerks für ausgewählte *Vision Chips* aufgelistet.

Externe Ansteuerung

Bei der externen Ansteuerung, die insbesondere für prototypische Realisierungen von Bedeutung ist, wird auf die Integration des Steuerwerks in den eigentlichen *Vision Chip* verzichtet. Die zur Ansteuerung der FU notwendigen Steuersignale werden gemeinsam mit den Ausgabedaten und zusätzlichen Statusinformationen nach außen geführt (siehe Abb. 3-4a), wobei als Gegenstelle i.d.R. ein FPGA eingesetzt wird. Da somit keine Festlegung vor der Fertigung des Chips erfolgen muss, wird die Flexibilität insbesondere während der Inbetriebnahme maximiert. Beliebigen Kombinationen bei der Ansteuerung stehen jedoch gleichzeitig sehr hohe Anforderungen bzgl. des Zeitverhaltens und der Leistungsaufnahme der Treiberzellen gegenüber. Je nach Komplexität und Anzahl der FU, variiert die Zahl der notwendigen Steuersignale erheblich (z.B. 170 bei MIPA4k [PLP09], 4 bei Flip-Q [FBCGCG11]).

Integrierter Decoder und externer Kontrollfluss

Besitzt eine FU n_{Ctrl} digitale Steuersignale zur Ansteuerung, so ergeben sich dafür $N_{\mathrm{Ctrl}} = 2^{n_{\mathrm{Ctrl}}}$ mögliche Belegungen. Für eine Abfolge der Länge n_{Seq} ergibt die Gesamtanzahl möglicher Belegungen somit zu $N_{\mathrm{Seq}} = 2^{n_{\mathrm{Seq}} n_{\mathrm{Ctrl}}}$. Abhängig von der konkreten FU wird lediglich eine Teilmenge davon tatsächlich benötigt. Zur Reduktion der Anzahl der notwendigen Steuersignale können die Abfolgen ausgewählter, elementarer Operationen mit Hilfe eines Decoders durch geeignete Instruktionen abstrahiert werden. Wie in Abb. 3-4b dargestellt, wird der Decoder dazu in den *Vision Chip* integriert. Die Auswahl der Operationen schränkt dabei die Flexibilität erheblich ein, erlaubt jedoch eine wesentliche Vereinfachung der externen Beschaltung bis hin zum Einsatz des *Vision Chips* als Peripherie-Komponente an einer CPU bzw. einem Mikrocontroller (z.B.

Arbeit	Integrationsvariante	Realisierung des Steuerwerks
Döge et al. [DHRP15]	intern	Multi-ASIP basiertes Steuerwerk
Zhang et al. [ZFW11]	intern	Mikrocontroller und spezieller SIMD-Prozessor
ACE4k [LEDCRV02a]	integr. Decoder	PLD und DSP („ALADDIN“ [ZRFS03])
ACE16k [RVLCC+04, LEDCRV02b]	integr. Decoder	PLD und DSP („Bi-i“ [ZR05])
Q-Eye / EyeRIS [Ana12, RVDCJG+10]	integr. Decoder	Nios-II Prozessor in FPGA oder on-chip Realisierung
PARIS1 [EBD+06, DKN02]	integr. Decoder	Mikrocontroller
MAPP [LMJM05]	integr. Decoder	Mikrocontroller
SCAMP-3 [Dud05, Dud11]	extern	FPGA-basiert (Mikrocontroller [BCLD06] bzw. CPU mit Decoder [CBD13])
MIPA4k [PLP09, Zar11]	extern	FPGA basiert
Flip-Q [FBCGCG11]	extern	Mikrocontroller, Ansteuerung über GPIO [FBCGLC+11]
S^{3}PE [KKI04b, IK01]	extern	FPGA basierter Mikrocontroller

Tabelle 3-1: Gegenüberstellung der Integrationsvarianten sowie der Realisierung der Steuerwerke für verschiedene *Vision Chips*.

[EBD+06, DKN02, LMJM05, RVLCC+04]). Die mögliche Instruktionsrate wird dabei maßgeblich durch den Datendurchsatz des Busses sowie dessen Anbindung an die CPU bestimmt. Ein bidirektionaler Bus, der auch zum Rücklesen der durch die FU ausgegebenen Status-Signale eingesetzt wird, schränkt die Instruktionsrate zudem weiter ein. Während unabhängige oder nicht gleichzeitig eingesetzte FU durch den gleichen Decoder angesteuert werden können, sind für parallel arbeitende FU separate Decoder und parallele Kontrollflüsse erforderlich.

Integrierte Ansteuerung

Wird neben dem Decoder auch der Kontrollfluss vollständig in den *Vision Chip* integriert (siehe Abb. 3-4c), so kann die externe Schnittstelle deutlich vereinfacht werden. Diese dient lediglich zur Programmierung und Parametrierung des Steuerwerks bzw. zum Abrufen der Ergebnisse. Auch wird die notwendige, externe Beschaltung weiter reduziert und ggf. ein autonomer Betrieb des *Vision Chips* ermöglicht. Statussignale der FU stehen zudem direkt zur Verarbeitung bereit, wodurch eine schnellere Reaktion möglich wird. Nachteilig kann sich hingegen die zur Realisierung des Steuerwerks zusätzlich be-

nötigte Fläche auswirken, die der eigentlichen Bildaufnahme und -verarbeitung nicht zur Verfügung steht. Zudem generiert das Steuerwerk örtlich konzentriert zusätzliche Verlustleistung, die die Funktion des Bildsensors beeinträchtigen kann.

3.2.2 Mögliche Realisierungen

FSM-Steuerwerk

Steuerwerke auf der Basis endlicher Automaten (*Finite State Machine*, FSM) [Vah10, Sch12] müssen genau für die anzusteuernden Komponenten entwickelt und abgestimmt werden. Durch den beliebigen Zugriff auf alle Kontroll- und Statussignale ist eine optimale Anpassung des Steuerwerks an die Komponente möglich. Da eine konkrete Implementierung insbesondere bei interner Realisierung jedoch unveränderlich ist, ist deren Flexibilität stark eingeschränkt. Zudem wird der Entwurf bei komplexen Algorithmen durch die Vielzahl der Zustände sehr aufwändig. In Bildsensoren kommen FSM basierte Steuerwerke insbesondere bei Standard-APS [Oht10] zum Einsatz sowie als Teil eines Decoders zur Ansteuerung einzelner FU.

Mikroprogrammsteuerwerk

Mikroprogrammsteuerwerke [Lyn93, Sch12] können als programmierbarer Sequenzer bzw. programmierbare FSM betrachtet werden. Die Belegung der Kontrollsignale wird durch den sog. Mikroprogrammspeicher vorgegeben, wobei der Zustandsvektor des Steuerwerks die jeweils aktuelle Speicheradresse repräsentiert. Zur Berechnung der Folgeadresse können Statussignalen der FU berücksichtigt werden, wodurch nicht-lineare Programmabläufe möglich sind. Das direkte Ablegen der Steuersignale im Mikroprogrammspeicher („horizontaler Mikrocode“) führt zu sehr breiten Speicherworten, weshalb oft kürzere, symbolische Instruktionen hinterlegt („vertikaler Mikrocode“) und vor der Ausgabe decodiert werden. Die Programmierung erfolgt durch Beschreiben des Mikroprogrammspeichers und erfordert erhebliches Detailwissen über die Architektur der zugrundeliegenden Hardware. In dem von Barr et al. [BCLD06] vorgestellten Steuerwerk für den *Vision Chip* SCAMP-3 (siehe 2.3) wird der Kontrollfluss zwar durch einen Mikrocontroller bereitgestellt, jedoch kommt die gewählte „*dual-instruction-stream*“ Architektur einem horizontalen Mikrocode sehr nahe. Der 47 Bit breite Steuervektor wird direkt zur Ansteuerung von SCAMP-3 verwendet und lediglich anhand des Programmzählers des Mikrocontrollers ausgegeben.

Prozessor-basiertes Steuerwerk

Bei Prozessor- bzw. Mikrocontroller-basierten Steuerwerken erfolgt die Einbindung der anzusteuernden FU mit Hilfe eines Decoders über den IO-Port der CPU bzw. durch Einbindung in deren Adressraum. Die für einen konkreten Algorithmus notwendige Abfolge einzelner, von den FU auszuführenden Operationen wird somit durch das von der CPU ausgeführte Programm vorgegeben. Da dieses neben den eigentlichen Instruktionen zur Ansteuerung der FU auch die für den Kontrollfluss erforderlichen Operationen enthält,

kann, bedingt durch die sequentielle Verarbeitung, nicht in jedem Schritt ein Befehl ausgegeben werden. Neben dem Lesen aus Registern bzw. aus Speicherbereichen müssen die Instruktionen zudem geeignet formatiert werden. So sind z.B. bei PARIS1 [DKN02] jeweils 6 Ausgaben über einen 8 Bit Datenbus notwendig, um eine 48 Bit breite Instruktion bereitzustellen. Zur Berücksichtigung von Statussignalen im Kontrollfluss sind diese zudem zunächst einzulesen.

ASIP-basiertes Steuerwerk

Applikationsspezifische Prozessoren [IL06, Liu08] besitzen spezielle Eigenschaften, durch die der Einsatz in bestimmten Anwendungsgebieten wesentlich erleichtert oder überhaupt erst ermöglicht wird [Sie99]. Insbesondere in den Bereichen Multimedia und Kryptographie werden zur Beschleunigung rechenintensiver Programmabschnitte besonders häufig benötigte Verarbeitungsschritte direkt in Hardware implementiert. Diese werden dem Programmierer i.d.R. als zusätzliche Instruktionen im Befehlssatz des Prozessors bereitgestellt. Zum Entwurf synchroner ASIPs zur digitalen Signalverarbeitung stehen Werkzeuge bereit, die sowohl die eigentliche Hardwarebeschreibung als auch die zugehörige Software-Umgebung aus einer gemeinsamen Eingabesprache erzeugen [HML07].

Im Gegensatz zum klassischen ASIP-Ansatz steht bei der Verwendung eines applikationsspezifischen Prozessors als Steuerwerk die Abstraktion des Verhaltens der anzusteuernden FU durch zusätzliche Instruktionen im Befehlssatz des ASIPs im Vordergrund. Der zur Ansteuerung der FU notwendige Decoder wird in den Prozessor integriert, wodurch eine enge Kopplung erreicht wird. So ist u.a. ein direkter Zugriff auf Datenfelder wie z.B. Register möglich. Zudem können Statussignale direkt ausgewertet und z.B. für bedingte Sprungoperationen verwendet werden.

3.3 Multi-ASIP basierte Steuerwerks-Architektur

3.3.1 Parallele Kontrollflüsse

Wie im vorhergehenden Abschnitt 3.2 beschrieben, werden die Steuersignale der FU in einem ASIP-basierten Steuerwerk durch einen in den Prozessor integrierten Decoder bereitgestellt. Um die Anforderungen hinsichtlich der Parallelität unabhängiger FU berücksichtigen zu können, sind jedoch mehrere, parallele Kontrollflüsse erforderlich.

Anstelle eines einzelnen, großen Decoders, der die Steuersignale aller Funktionseinheiten bereitstellt, werden die FU in Gruppen zusammengefasst. Alle FU einer Gruppe werden durch einen gemeinsamen Decoder angesteuert, der jeweils in einen separaten Prozessor integriert wird. In Abbildung 3-5 sind beispielhaft fünf Funktionseinheiten „FU1“ bis „FU5“ dargestellt, die zu Gruppen zusammengefasst sind und von drei ASIPs angesteuert werden. Zunächst werden die Funktionseinheiten, deren Ansteuerung parallel und unabhängig voneinander erfolgen muss, ausgewählt und jeweils einer neuen Gruppe zugeordnet. In Abbildung 3-5 sind dies die Funktionseinheiten „FU1“ und „FU2“. Die verbleibenden FU werden entweder zu einer weiteren, unabhängigen Gruppe („FU4“ und „FU5“ in Gruppe 3) zusammengefasst oder – insbesondere bei verwandten Operationen

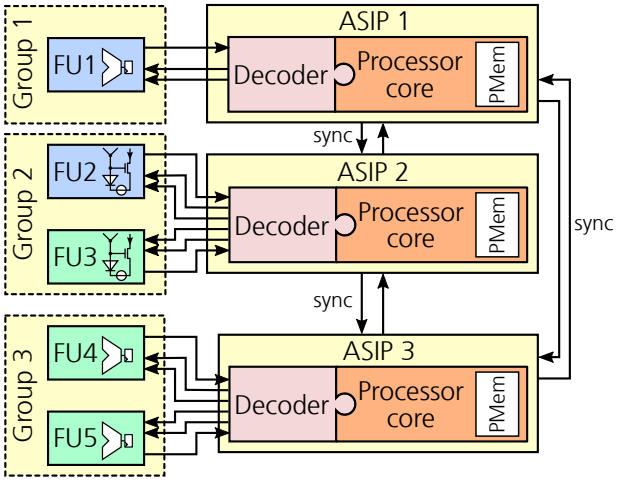


Abbildung 3-5: Gruppenzuordnung einzelner FU zu verschiedenen ASIPs.

– einer bestehenden Gruppe hinzugefügt („FU3“ zu Gruppe 2). Das resultierende, aus verschiedenen ASIPs bestehende Steuerwerk stellt entsprechend mehrere unabhängige Kontrollflüsse bereit. Da sich die einzelnen Prozessoren zudem hinsichtlich ihres Befehlssatzes unterscheiden, wird das System als heterogene Multi-ASIP Architektur bezeichnet.

Die im Rahmen dieser Arbeit entwickelte und in den folgenden Abschnitten genauer betrachtete Steuerwerksarchitektur [RDPH15] verwendet einen einfachen, Stack-basierten Prozessorkern, der durch individuelle Befehlssatzerweiterungen zur Ansteuerung der FU eingesetzt wird. Die FU agieren hierbei als zusätzliche, durch den jeweiligen Prozessor gesteuerte Datenpfadelemente. Zur Berücksichtigung von Abhängigkeiten der Kontrollflüsse bzw. der FU untereinander stehen Methoden zur Synchronisation sowie zum Datenaustausch bereit.

3.3.2 Stack-basierter Prozessorkern

Die Basis der einzelnen ASIPs bildet ein Stack-basierter [Koo89] Prozessorkern, der sich durch einen einfachen Aufbau und eine hohe Flexibilität auszeichnet. Im Vordergrund steht dabei die Integration der Ansteuerung der Funktionseinheiten durch die Erweiterung des Befehlssatzes. Da in Stack-basierten Prozessoren keine Adressierung der Operanden erfolgen muss, werden gegenüber Register-Maschinen weniger Bits zur Kodierung der Instruktionen benötigt. Gleichzeitig sind für Berechnungen jedoch mehr Instruktionen erforderlich, da die Operanden zunächst auf den Stack geladen werden müssen. Der Verzicht auf die explizite Adressierung der Operanden ermöglicht eine einfache Schnittstelle für Befehlssatzerweiterungen, da sich auch diese lediglich auf die oberen Stack-Elemente beziehen bzw. diese manipulieren können.

Die Architektur des Prozessorkerns, der über getrennte Stacks für Daten (`OpStack`) und Rücksprungadressen (`CallStack`) verfügt, ist in Abb. 3-6 dargestellt. Neben einem

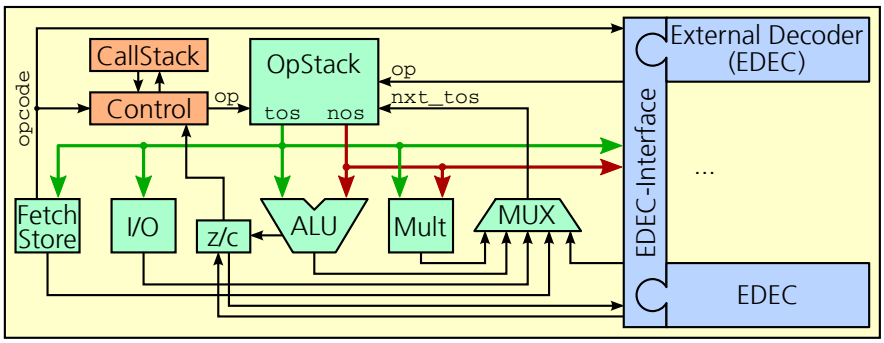


Abbildung 3-6: Architektur des Stack-basierten Prozessorkerns.

Stack	Operation	EDEC Zugriff	Beschreibung
`CallStack`	`nop`	nein	keine Veränderung des Stacks
	`push`	nein	Rücksprungadresse ablegen
	`pop`	nein	Rücksprungadresse löschen
`OpStack`	`nop`	ja	keine Veränderung des Stacks
	`push`	ja	Datum auf Stack legen
	`pop`	ja	`tos` von Stack löschen
	`replace`	ja	Kurzform für: `pop / push value`
	`replace_pop`	ja	Kurzform für: `pop / pop / push value`
	`swap`	nein	`tos` und `nos` vertauschen
	`load_sp`	nein	Datum von Adresse relativ zu Stack-Zeiger laden und auf Stack legen
	`store_sp`	nein	`tos` an Adresse relativ zu Stack-Zeiger speichern und von Stack löschen

Tabelle 3-2: Auflistung unterstützter Operationen zur Manipulation der Stacks.

digitalen Datenpfad ist das wesentliche Element eine spezielle Schnittstelle zur Erweiterung des Befehlssatzes durch externe Decoder (*EDEC-Interface*).

Sowohl der `OpStack` als auch der `CallStack` verfügen zur Entlastung der ALU jeweils über eigene Zähler zur Adressierung. Die jeweils unterstützten Operationen zur Manipulation sind in Tabelle 3-2 zusammengefasst. Während der `CallStack` lediglich Operationen zum Ablegen bzw. Entfernen einer Rücksprungadresse bereitstellt, sind im `OpStack` zusätzliche, häufig benötigte Operationen zum Datenaustausch sowie zum Laden bzw. Speichern relativ zum Stack-Zeiger realisiert. Da die oberen beiden Einträge des `OpStack` (Top-Of-Stack `tos` und Next-of-Stack `nos`) den meisten Instruktionen als Operanden dienen, werden diese in Registern abgelegt.

Der hinsichtlich seiner Breite frei parametrierbare Datenpfad umfasst eine ALU, die neben Addition und Subtraktion auch Verschiebe-, Rotations- und Logikoperationen unterstützt, sowie einen seriellen Multiplizierer. Für die Berechnungen der ALU werden

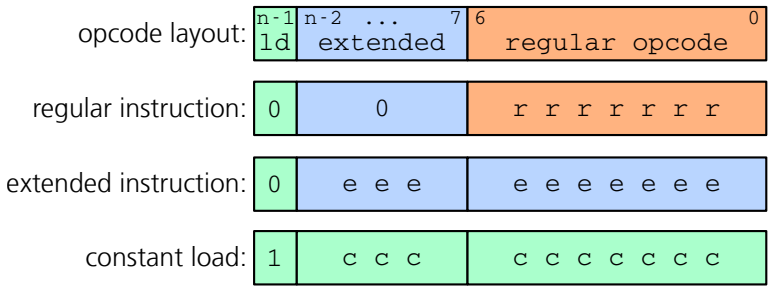


Abbildung 3-7: Aufbau der Befehlsworte.

die für bedingte Sprünge relevanten Flags „Zero“ (`z`) und „Carry“ (`c`) berechnet und in separaten Registern gespeichert. Ein einfacher Ein-/Ausgabe-Port stellt eine Schnittstelle zur Ansteuerung weiterer Komponenten bereit. Zudem können Werte aus dem Programmspeicher geladen bzw. in diesem abgelegt werden, wobei jedoch die möglicherweise unterschiedlichen Wortbreiten berücksichtigt werden müssen.

Der Befehlssatz des Stack-Prozessors ist sehr einfach gehalten und in Tabelle 3-3 dargestellt. Zur Kodierung der Befehle werden mindestens $n = 8$ Bit benötigt, wobei das Laden einer Konstanten („Load“, `ld`) gesondert behandelt wird. Angelehnt an die Architektur der „ZPU“ [Har08], entscheidet das höchstwertige Bit des Befehlswortes, ob ein LOAD-Befehl vorliegt und somit eine $n - 1$ Bit breite Konstante auf den Stack geladen werden soll. Um breitere Konstanten zu laden, werden mehrere, direkt aufeinanderfolgende LOAD-Operationen bis zur vollen Datenpfadbreite konkateniert. Verschiedene, aufeinanderfolgende Konstanten müssen durch einen beliebigen anderen Befehl unterbrochen werden, weshalb ggf. ein `nop` eingefügt werden muss. Abhängig von den individuellen Anforderungen der eingebundenen Befehlssatzerweiterungen kann die Breite der Befehlsworte angepasst werden, wobei sich entsprechend $N = 2^{n-1} - 2^7$ mögliche, zusätzliche Befehlsworte ergeben. Zur besseren Veranschaulichung wird das beschriebene Prinzip in Abb. 3-7 grafisch dargestellt.

Zur Erweiterung des Befehlssatzes, werden dem Prozessorkern zusätzliche externe Decoder (EDEC) zur Seite gestellt. Wird ein externer Befehl erkannt, so wird die Kontrolle an den entsprechenden EDEC übergeben, der Prozessorkern wird für die Dauer der Ausführung der jeweiligen Instruktion angehalten. Die Schnittstelle zwischen Prozessorkern und EDEC stellt neben dem aktuellen Befehlswort auch `tos` und `nos` sowie die Flags `c` und `z` als Parameter zur Verfügung. Mit Abschluss der Ausführung einer Instruktion kann der EDEC den `OpStack` manipulieren, wobei jedoch lediglich ein Teil der Manipulationsoperationen erlaubt ist (vgl. Tab. 3-2). Außerdem können die Flags `c` und `z` überschrieben werden, wodurch bedingte Sprungoperationen in Abhängigkeit erweiterter Befehle möglich sind.

3.3.3 Kommunikation zwischen den Prozessoren

Zum Austausch von Daten zwischen den einzelnen ASIPs stehen *Shared Register Files* zur Verfügung, die einen konkurrierenden Zugriff ausgehend von allen Prozessoren er-

Befehl	Beschreibung
`ld`	Laden einer Konstante aus Programmspeicher auf den Operandenstack
`add, adc, sub, sbc`	Addition bzw. Subtraktion von TOS und NOS ohne/mit einlaufendem Carry-Flag, Ergebnis auf Operandenstack
`xor, and, or`	binäre Logik-Operationen von TOS und NOS, Ergebnis auf Operandenstack
`not`	bitweise Negation von TOS
`shl0, shl1, shr0, shr1`	Verschiebung von TOS um 1 Bit nach links/rechts jeweils mit einlaufender 0 oder 1
`rol, rolc, ror, rorc`	Rotation von TOS um eine Bit nach links/rechts jeweils ohne/mit Einbezug des Carry-Flags
`swap`	Austausch TOS und NOS
`pop`	entfernen von TOS
`nop`	keine Operation
`jmp`	Sprungoperation, Adresse in TOS
`jmpz, jmpnz, jmpc, jmpnc`	bedingte Sprungoperation abhängig von Z/C-Flag, Adresse in TOS
`call`	Unterprogrammaufruf, Adresse in TOS, Rücksprungadresse wird auf Aufrufstack gelegt
`ret`	Rücksprung von Unterprogramm, Adresse von Aufrufstack
`mload`	Laden eines Wortes aus dem Programmspeicher, Adresse in TOS
`mstore`	Speichern von NOS in Programmspeicher, Adresse in TOS
`loadsp`	Laden eines Eintrags vom Stack relativ zu aktuellem Stack-Zeiger, Offset im Opcode
`storesp`	Speichern von TOS an Position relativ zu aktuellem Stack-Zeiger, Offset im Opcode
`in`	Lesen eines Wertes vom Eingangsport, Port im Opcode
`out`	Ausgabe von TOS auf Ausgangsport, Port im Opcode

Tabelle 3-3: Basis-Befehlssatz des Stack-Prozessors.

möglichen. Um einen deterministischen Ablauf zu gewährleisten, wurde auf die Unterstützung von Interrupts bewusst verzichtet. Zur Interaktion der einzelnen Kontrollflüsse, wird stattdessen ein auf Methoden der Inter-Prozess-Kommunikation [Tan09] basierender Synchronisationsmechanismus eingesetzt. Ein Prozessor kann auf Ereignisse warten

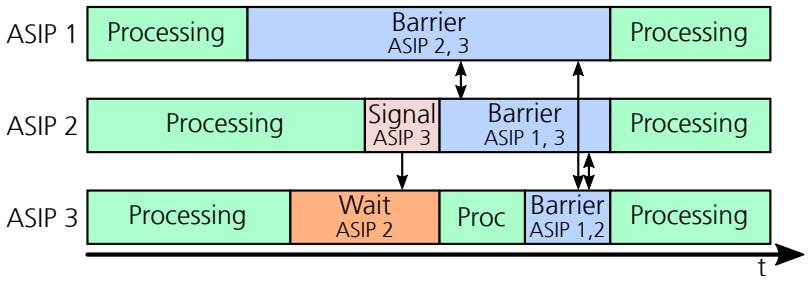


Abbildung 3-8: Beispiel des Ablaufs der Synchronisation dreier ASIPs durch gegenseitige Benachrichtigung mit Hilfe von WAIT, SIGNAL und BARRIER.

(WAIT) bzw. andere Prozessoren benachrichtigen (SIGNAL). Zudem kann auch auf externe Ereignisse gewartet werden, d.h. z.B. auf ein externes Signal am IO-Controller. Zur gleichzeitigen Synchronisation mehrerer Endpunkte stehen zudem Barrieren (BARRIER) bereit, die als Kombination von WAIT und SIGNAL aufgefasst werden können. Die an einer Barriere beteiligten Kontrollflüsse können diese erst überschreiten, wenn alle notwendigen Kontrollflüsse die Barriere erreicht haben. Die Reihenfolge des Eintretens der Kontrollflüsse in die Barriere spielt dabei keine Rolle.

In Abb. 3-8 ist der Ablauf der Synchronisation beispielhaft dargestellt: Zu Beginn arbeiten alle ASIPs parallel, bevor zunächst ASIP 1 eine Barriere zur Synchronisation mit den ASIPs 2 und 3 betritt. Daraufhin wartet ASIP 3 auf ASIP 2. ASIP 2 sendet das entsprechende Signal und betritt ebenfalls eine Barriere zur Synchronisation mit den ASIPs 1 und 3. Erst wenn auch ASIP 3 die Barriere betritt, können alle Beteiligten ihren Programmablauf fortsetzen.

Da die einzelnen Prozessoren in unterschiedlichen Taktdomänen arbeiten, wird zur Integration der Synchronisation eine spezielle, asynchrone Hardwareunterstützung eingesetzt, die die gegenseitige Benachrichtigung unabhängig vom Takt der jeweiligen Endpunkte ermöglicht.

3.3.4 Umsetzung in VSoC

Das Steuerwerk des in 3.1.1 vorgestellten VSoC umfasst die drei ASIPs GLOBALCTRL, LINECTRL und SIMDCTRL, wobei jeweils die gleiche Konfiguration der Stack-Prozessoren verwendet wird. Die Implementierung erfolgte in VHDL und wurde für eine 180 nm CMOS-Technologie synthetisiert. Die wesentlichen Parameter des Steuerwerks sind in Tab. 3-4 zusammengefasst.

GLOBALCTRL übernimmt die Rolle der globalen Steuerung sowie der Kommunikation mit der Umgebung. So sind, neben den externen Schnittstellen GPIO, EGI und PARDOUT, auch verschiedene Registerbereiche zum Datenaustausch zwischen den einzelnen ASIPs sowie für benutzerdefinierte, von außerhalb parametrierbare Einstellungen Bestandteil von GLOBALCTRL. Die spezialisierten Befehle beschränken sich daher auf die Ansteuerung der Kommunikationsschnittstellen sowie die Synchronisation mit den anderen ASIPs des Steuerwerks.

ASIP	Parameter	Wert
allgemein	Datenpfadbreite	16 Bit
	Befehlswortbreite	12 Bit
	Größe des Programmspeichers	4096 Befehle
	Tiefe des Operanden-Stacks	64 Einträge á 16 Bit
	Tiefe des Aufruf-Stacks	8 Einträge á 12 Bit
	Anzahl Basis-Befehle	33
GLOBALCTRL	Anzahl spezifischer Befehle	35
	benötigte Fläche	0,41 mm^2
SIMDCTRL	Anzahl spezifischer Befehle	54
	benötigte Fläche	0,18 mm^2
LINECTRL	Anzahl spezifischer Befehle	53
	benötigte Fläche	0,16 mm^2

Tabelle 3-4: Übersicht der wichtigsten Parameter der einzelnen ASIPs des im VSoC integrierten Steuerwerks.

Durch LINECTRL wird die zeilenweise Ansteuerung der Pixelmatrix ermöglicht, indem die einzelnen Steuersignale für alle Pixel einer Zeile jeweils gemeinsam vorgegeben werden. Dazu steht ein Schieberegister bereit, das zunächst, mit einem anzuwendenden Faltungskern beginnend, an einer beliebigen Position initialisiert wird. Durch Verschiebung des Faltungskerns während des Auslesens eines Bildes nach oben oder unten, kann dieser auf die gesamte Matrix angewendet werden. Die für Ausgabeoperationen auf die Spaltenleitungen (siehe 3.1.3) angewendete Integrationszeit wird durch den zu LINECTRL gehörenden Impulsgenerators realisiert.

SIMDCTRL obliegt schließlich die Ansteuerung der spaltenparallelen mixed-signal PE des SIMD-Arrays. So sind die einzelnen Funktionen der ALU (Addition, Subtraktion, Vergleichs-, Logik- und Verschiebe-Operationen), Operationen zum Register- und Speicherzugriff sowie die Ansteuerung der zur A/D-Wandlung erforderlichen Stromquellen als spezifische Instruktionen in den Befehlssatz des ASIP eingebunden. Durch die enge Kopplung erscheint das SIMD-Array als eigenständiger, zweiter Datenpfad des Prozessors.

Mit Ausnahme des von Zhang et al. [ZFW11] vorgestellten Systems beschränken sich alle in der betrachteten Literatur beschriebenen *Vision Chips* auf einen einzelnen Kontrollpfad. Der parallele Betrieb von analoger Bildaufnahme und -verarbeitung, digitaler Berechnungen im SIMD-Array und der Ausgabe der Ergebnisse erlaubt hingegen dem in dieser Arbeit behandelten VSoC eine Überlappung bzw. ein „*Pipelining*“ der einzelnen Verarbeitungsschritte. Gleichzeitig wird die Flexibilität durch den Verzicht auf fest vorgegebene Abläufe gesteigert.

4 Modellierung und Simulation

4.1 Aufbau und Struktur der Simulationsumgebung

4.1.1 Zielstellung

Bedingt durch die Heterogenität der Architektur des VSoC (siehe 3.1) sind Simulation und Test unter reproduzierbaren Bedingungen ein wesentlicher Bestandteil des Entwurfs sowie der Implementierung von Bildverarbeitungsalgorithmen. Beim klassischen, Kamera-PC-basierten Ansatz (siehe Abb. 1-1) kann dazu die Kamera durch eine Bilddatenbank ersetzt werden, die Referenzbilder des zu beobachtenden Prozesses bereitstellt. In einem *Vision Chip* ist hingegen die Betrachtung von Bildaufnahme und Bildverarbeitung nicht unabhängig voneinander möglich. So sind sowohl zeitabhängige Effekte als auch eine, bedingt durch die analoge Verarbeitung, begrenzte Genauigkeit zu berücksichtigen. Nicht-Idealitäten, wie FPN oder defekte Pixel, können in konventionellen Systemen bereits in der Kamera oder per Software auf dem PC noch vor der eigentlichen Bildverarbeitung korrigiert werden. Die integrierte Merkmals-Extraktion erfordert hingegen eine Berücksichtigung in den entsprechenden Algorithmen. Während zur Simulation des Verhaltens konventioneller Bildsensoren (siehe 2.2.1) mathematische Beschreibungen der Signaltransformation unter Einbeziehung von Störeinflüssen und Nicht-Idealitäten im Vordergrund stehen, sind jedoch für *Vision Chips* strukturtreue Modelle (siehe 2.2.2) mit Berücksichtigung des konkreten Algorithmus unabdingbar. Es muss untersucht werden, ob der Einfluss von Defekten und Nicht-Idealitäten zu ungewolltem Verhalten führt und dieses ggf. korrigiert werden muss.

Das wichtigste Ziel der Simulationsumgebung besteht in der deterministischen Simulation des Verhaltens implementierter Bildaufnahme- und -verarbeitungsalgorithmen. Das reale System erlaubt lediglich eine Auswertung basierend auf ausgegebenen Ergebnissen, wobei der Einfluss der im jeweils eingesetzten Chip vorhandenen Defekte und Nicht-Idealitäten nicht unabhängig betrachtet werden kann. In der Simulation ist hingegen die Betrachtung des internen Zustands in digitaler sowie analoger Domäne jederzeit möglich. Die Manipulation zahlreicher Konfigurationswerte für jedes Pixel separat, ermöglicht die Injektion von Nicht-Idealitäten, Parameterabweichungen oder Defekten unmittelbar am Ort ihrer Entstehung. Abhängig vom jeweiligen Programm kann daraufhin deren Auswirkung auf den Bildauslese- und Verarbeitungsprozess beobachtet werden. Die Simulation ermöglicht zudem einen Test, ob ein bestimmter Algorithmus auf der realen Hardware implementierbar ist.

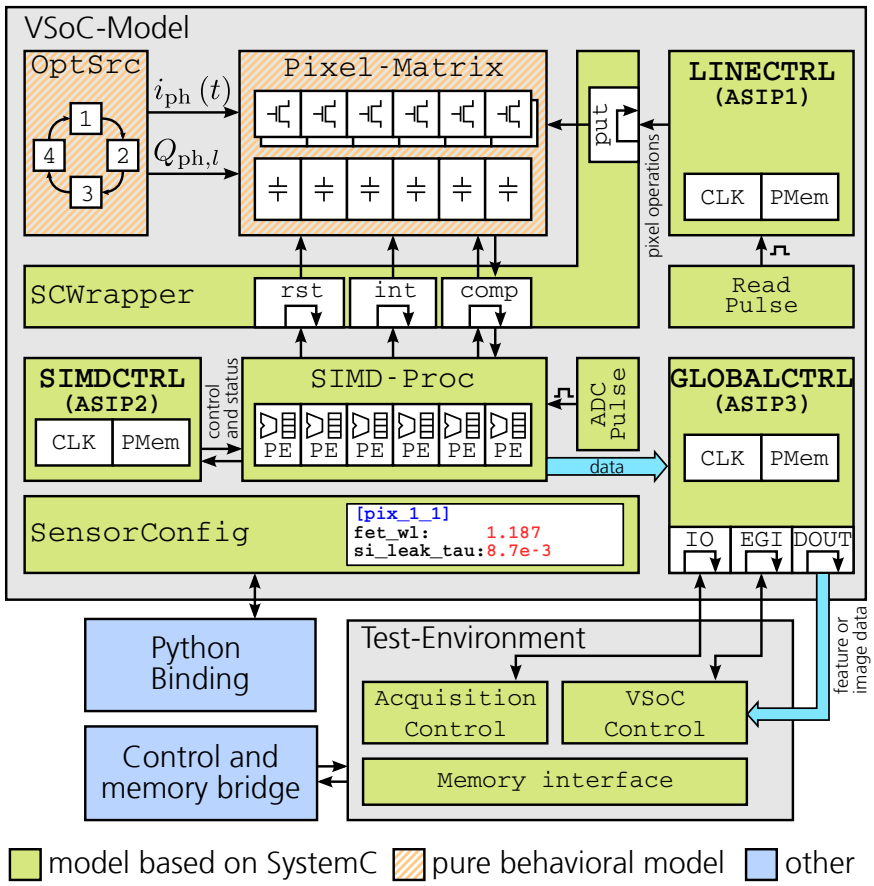


Abbildung 4-1: Darstellung der Struktur des Simulationssystems (Quelle: [RDH+16]).

4.1.2 Architekturübersicht

In Abb. 4-1 ist die Architektur des strukturtreuen Modells des VSoC als Übersicht dargestellt, das die Grundlage des im Rahmen dieser Arbeit entwickelten [RHDP15, RDH+16] Simulationssystems bildet. Analog zum VSoC selbst (vgl. Abb. 3-1) umfasst dieses ein Modell der Pixelmatrix sowie Modelle der einzelnen ASIPs des integrierten Steuerwerks. Zudem wird die beobachtete Szene durch ein Szenen-Modell beschrieben. Neben dem eigentlichen VSoC werden auch wesentliche Bestandteile der Testumgebung modelliert sowie Methoden zur Konfiguration und Parametrierung bereitgestellt. Sowohl das integrierte Steuerwerk (ASIP 1, 2 und 3 in Abb. 4-1) als auch die digitale Signalverarbeitung (`SIMD-Proc`) und das Modell der Testumgebung wurden mit Hilfe von SystemC realisiert. Auch zur Kommunikation der Module untereinander sowie zwischen dem Modell des VSoC und der Testumgebung, kommen Methoden von SystemC zum Einsatz.

Die analogen Komponenten der Pixelmatrix des VSoC wurden während ihres Entwurfs umfangreichen Simulationen unterzogen, wobei neben verschiedenen Umwelteinflüssen auch Parameterabweichungen berücksichtigt wurden. Diese Simulationen auf Schaltungs-

ebene sind sehr zeitaufwändig und rechenintensiv. Zur Steigerung der Simulationsgeschwindigkeit, wurde daher auf die strukturtreue Modellierung der Pixelmatrix auf Schaltungsebene in SystemC oder SystemC-AMS [OSC10] verzichtet. Stattdessen wird lediglich das Verhalten abhängig der durch LINECTRL angelegten Steuerworte und der durch SIMDCTRL bei der A/D-Wandlung initiierten Ausgabe von Kompensationsströmen betrachtet. Die spezielle Komponente `SCWrapper` dient schließlich zur Ankopplung des Modells der Pixelmatrix an den in SystemC implementierten Teil des VSoC-Modells.

Sowohl die Bildaufnahme als auch die Bildverarbeitung unter Verwendung analoger oder digitaler Funktionseinheiten kann mit Hilfe benutzerdefinierter Algorithmen beeinflusst werden. Ein Szenen-Modell (`OptSrc` in Abb. 4-1) stellt daher eine zeitabhängige Repräsentation der beobachteten Szene bereit und liefert sowohl den Fotostrom für einen konkreten Zeitpunkt als auch die in einem gegebenen Zeitintervall integrierte Ladung.

Die spezielle Komponente `SensorConfig` (siehe Abb. 4-1) ermöglicht die Registrierung von Schlüssel-Werte-Paaren und erlaubt somit die Parametrierung der verschiedenen Komponenten des Systems sowie das gezielte Einbringen von Defekten bzw. Parameterabweichungen. Mit Hilfe einer Anbindung an die Programmiersprache Python [Pyt16, PW09], können die Parameter ggf. auch zur Laufzeit verändert werden. Wird einem Parameter im Betrieb ein neuer Wert zugewiesen, so wird die entsprechende Komponente benachrichtigt.

Das eigentliche SystemC-Modell des VSoC wird in einer Test-Umgebung instantiiert, die als Modell einer einfachen Kamera betrachtet werden kann und die Umgebung zur Inbetriebnahme des eigentlichen Chips nachempfindet. Neben der eigentlichen Ansteuerung zur Konfiguration und Programmierung des VSoC umfasst diese auch die Aufnahmesteuerung sowie die Schnittstelle zur Speicherverwaltung (siehe dazu auch 5.1.4).

4.1.3 Steuerung und digitale Verarbeitung

Das digitale Steuerwerk umfasst zunächst die zur Steuerung der Funktionseinheiten verantwortlichen ASIPs (siehe 3.3.4). Für deren Modellierung ist ein *Instruction Set Simulator* (ISS) des stack-basierten Prozessorkerns (siehe 3.3.2) notwendig, der auch hinsichtlich der individuellen Befehlssatzerweiterungen der tatsächlichen Implementierung entspricht. Die spaltenparallele, digitale Verarbeitung in den PE des SIMD-Datenpfades ist als Befehlssatzerweiterung in SIMDCTRL (`SIMD-Proc` in Abb. 4-1) integriert. Diese modelliert die PE auf Verhaltensebene, wobei eine Operation auf allen PE im gleichen Simulations-Zeitschritt abhängig vom internen Zustand des PE ausgeführt wird. Zur Kommunikation mit der Testumgebung stehen zudem die GPIO-Ports, die zur Programmierung und Parametrierung verwendete EGI-Schnittstelle sowie die schnelle Datenausgabe über PARDOUT zur Verfügung. Die auszuführenden Programme sind binärkompatibel zum realen System und werden entsprechend durch die Testumgebung über EGI in die Programmspeicher (`PMem` in Abb. 4-1) geladen. Die für die Ausführung von Pixel-Operationen sowie für die Ausgabe der Kompensationsladungen während der A/D-Wandlung eingesetzten Impulsgeneratoren sind im Modell ebenfalls mit Hilfe von SystemC realisiert. Die Taktfrequenz bzw. die Impulslänge hängen jeweils von der aktuellen Konfiguration

ab. Zur Modellierung der Rauscheinflüsse („*Jitter*“) wird zudem eine Normalverteilung angenommen, deren Parameter mit Hilfe von `SensorConfig` vorgegeben werden.

4.2 Modellierung von Bildaufnahme und -verarbeitung

4.2.1 Szenen-Modell basierend auf Bildsequenzen

Bildaufnahme und Bildverarbeitung sind in *Vision Chips* sehr eng miteinander verflochten. Gleichzeitig ist das Zeitverhalten der integrierten Signalverarbeitung abhängig von der jeweiligen Programmierung. Zur Evaluierung konkreter Algorithmen ist somit eine zeitlich veränderliche Repräsentation der beobachteten Szene erforderlich.

Grundlage des Szenen-Modells bildet ein Ring-Puffer (siehe Abb. 4-2), in welchen eine Sequenz von k aufeinanderfolgenden Bildern mit Abstand τ_{dist} abgelegt wird. Die einzelnen Bilder wiederholen sich daraufhin periodisch mit $T = k \cdot \tau_{\mathrm{dist}}$. Dabei werden die RGB-Helligkeitswerte Pix_{ch} des Pixels l mit $ch \in \{\mathrm{r,g,b}\}$ als lokale Bestrahlungsstärke $\underline{E_{\mathrm{e},l}} = [E_{\mathrm{e,r},l}, E_{\mathrm{e,g},l}, E_{\mathrm{e,b},l}]^\top$ mit drei spektralen Komponenten interpretiert, wobei $E_{\mathrm{e},ch,l} = c_{ch} \cdot Pix_{ch}$ gilt und c_{ch} der jeweiligen Photonenrate pro LSB entspricht. Auf die Simulation der abbildenden Optik wird dabei jedoch verzichtet, zudem wird die Empfindlichkeit über die Fläche eines Pixels als homogen betrachtet. Zur Vereinfachung der weiteren Berechnung wird die Kameragleichung Gl. 2-1 separiert, so das für jedes Pixel l des Sensors der korrespondierende Fotostrom $i_{\mathrm{ph},l}(t)$ als Funktion der Zeit vorgegeben wird. Für das Bild m zum diskreten Zeitpunkt $t = m \cdot \tau_{\mathrm{dist}}$ ergibt sich dieser aus $i_{\mathrm{ph},l}(t = m \cdot \tau_{\mathrm{dist}}) = e \cdot \underline{E_{\mathrm{e},l}}^\top \cdot \underline{s_l}$. Die spektrale Empfindlichkeit $\underline{s_l} = [s_{\mathrm{r},l}, s_{\mathrm{g},l}, s_{\mathrm{b},l}]^\top$, d.h. der Anzahl erzeugter Elektronen pro Photon, kann dabei für jedes Pixel separat angegeben und z.B. zur Modellierung von Farbfiltern verwendet werden.

Mittels linearer Interpolation wird $i_{\mathrm{ph}}(t)$ für einen beliebigen Zeitpunkt berechnet, wobei gilt:

$$i_{\mathrm{ph}}(t) = \frac{\tau - t}{\tau} \cdot i_{\mathrm{ph}}\left(\left\lfloor \frac{t}{\tau} \right\rfloor \cdot \tau\right) + \frac{t}{\tau} \cdot i_{\mathrm{ph}}\left(\left\lfloor \frac{t}{\tau} + 1 \right\rfloor \cdot \tau\right)$$

Durch stückweise Integration von

$$Q_{\mathrm{ph},l} = \int_{t_1}^{t_2} i_{\mathrm{ph},l}(t)\, dt$$

kann die während eines Zeitintervalls $[t_1,t_2]$ akkumulierte Ladung $Q_{\mathrm{ph},l}$ berechnet werden.

Die notwendigen Bildfolgen können sowohl mit einer konventionellen Kamera aufgezeichnet als auch synthetisch erzeugt werden. Für die in Abb. 4-2 dargestellte Bildfolge wurde die Darstellung eines Windrads in 360 Schritten um jeweils 1° weiter rotiert und auf den Hintergrund abgebildet. Durch die Verwendung von Bildfolgen und der Integration über die Zeit ist das Modell in der Lage, zeitabhängige Effekte bei der Bildaufnahme und Bildverarbeitung aufzuzeigen. Der zeitliche Abstand τ_{dist} der Bildsequenz muss dabei auch unter Berücksichtigung der Zeitkonstanten des implementierten Algorithmus

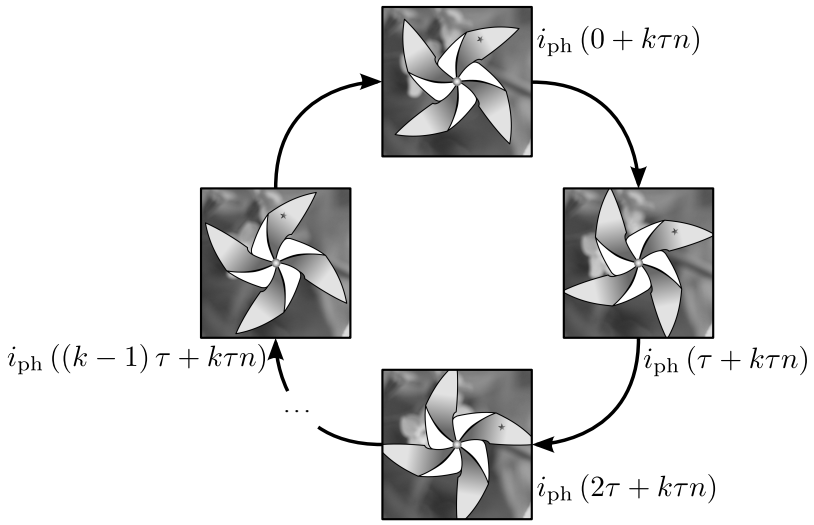


Abbildung 4-2: Darstellung des im Szenen-Modell eingesetzten Ring-Puffers. Die abgelegte Bildsequenz wiederholt sich periodisch mit $T = k \cdot \tau_{\mathrm{dist}}$ (Quelle: [RDH+16]).

gewählt werden. Sowohl der Parameter τ_{dist} als auch die für jedes Pixel separat einstellbaren Parameter c_{ch} und $\underline{s_l}$ werden durch `SensorConfig` vorgegeben und können somit individuell angepasst werden.

4.2.2 Modellierung der Pixel-Matrix

In Abbildung 4-3 ist die Struktur des Modells der Pixel-Matrix als Übersicht dargestellt. Dieses umfasst, neben den eigentlichen Pixel-Zellen („*Pixel cells*“), auch die in jeder Spalte vorhandenen Kondensatoren („*Capacitors*“) sowie den Komparator („*Comparators*“) als Übergang zwischen Analog- und Digitalteil. Eine Beeinflussung der Pixel kann sowohl durch globale und somit für alle Pixel gleiche Bias-Spannungen („*Bias Settings*“) als auch durch die zeilenweise vorgegebenen Steuerworte („*control words*“) erfolgen. Änderungen der Bias-Spannungen treten im Verhältnis zu Änderungen der vorgegebenen Steuerworte sehr selten auf. Da zudem alle Pixel einer Zeile stets die gleichen Kontrollworte erhalten, werden diese im Modell ähnlich dem Vorgehen von Blanchard et al. [BDP11] zeilenweise gruppiert.

Das Szenen-Modell (`OptSrc` in Abb. 4-1 und 4-3) stellt für jedes Pixel den aktuellen Fotostrom mittels `iph(t)` sowie die in einem Zeitintervall $[t_1,t_2]$ akkumulierte Ladung durch `q(t1, t2)` bereit. Die notwendige Schnittstelle zur Ansteuerung der Pixelmatrix beschränkt sich auf wenige Funktionen. Zur zeilenweisen Ansteuerung der Pixelzellen steht die Funktion `put(row_nr, op)` bereit, die die Ausführung (`row.proc(op)`) der jeweils übergebenen Operation auf der angegebenen Zeile veranlasst. Als Schnittstelle zum

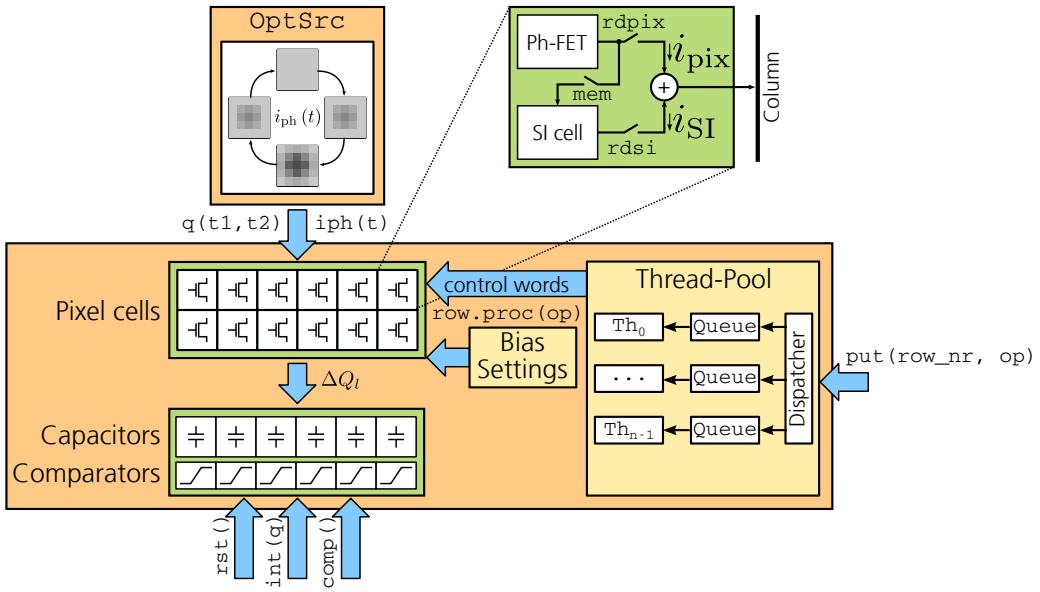


Abbildung 4-3: Architektur des Modells der Pixel-Matrix (Quelle: [RHDP15]).

SIMD-Array (siehe 4.2.5) dienen die Funktionen `rst()` zum Rücksetzen der Spaltenkapazität, `int(q)` zur Integration einer Kompensationsladung sowie `comp()` zum Vergleich der resultierenden Spannung gegen eine Referenz.

Das Modell jedes Pixels enthält neben dem fotoempfindlichen Element auch eine SI-Zelle zur Speicherung eines analogen Grauwertes.

4.2.3 Vereinfachte Ersatzschaltung eines Pixels

In Abbildung 4-4 ist die vereinfachte Ersatzschaltung des Fotodetektors eines Pixels mit Foto-FET dargestellt. Die Foto-Diode wurde gegenüber Abb. 3-2 durch eine gesteuerte Stromquelle mit dem lichtabhängigen Fotostrom i_{ph} ersetzt. Der vom Pixel ausgegebene Strom i_{pix} ist äquivalent zum Drain-Source Strom i_{DS} des FETs. Dieser kann mit Hilfe des SPICE Level-1 FET Modells [GMS07] abhängig vom Arbeitspunkt des Transistors — bestimmt durch die Spannungen V_{DD}, V_{GPh}, V_{BPh} und V_{DPh} — berechnet werden. Während die Spannungen V_{DD}, V_{GPh} und V_{DPh} als Bias von außen vorgegeben werden, ist V_{BPh} sowohl abhängig von der Ansteuerung als auch von der akkumulierten Ladung $Q_{\mathrm{ph},l}$ sowie vom einfallenden Licht. Letzteres wird repräsentiert durch den resultierenden Fotostrom i_{ph}. Sowohl i_{ph} als auch $Q_{\mathrm{ph},l}$ werden durch das Szenen-Modell (`OptSrc` in Abb. 4-1 bzw. 4-3) vorgegeben.

Für die Simulation wird ein vereinfachtes, zeitdiskretes Verhalten der Schaltungen angenommen und somit auf die Lösung der sonst notwendigen Differenzialgleichungen verzichtet. Stattdessen wird für eine Änderung von Bias-Spannungen bzw. der angelegten

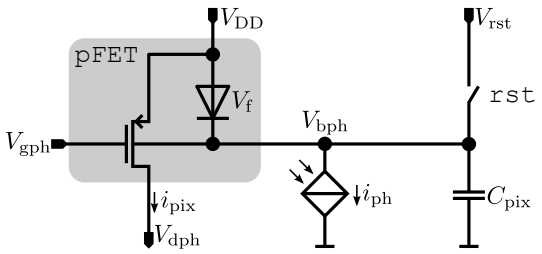


Abbildung 4-4: Vereinfachte Ersatzschaltung des Fotodetektors eines Pixels (Quelle: [RHDP15]).

Steuerworte zum Zeitpunkt t_i der Zustandsübergang abhängig von den im Zeitintervall $[t_{i-1},t_i)$ gültigen Steuerworten berechnet. Der Fotostrom $i_{\mathrm{ph},l}\,(t_i)$ sowie die im jeweiligen Zeitintervall integrierte Ladung $Q_{\mathrm{ph},l}$ für jedes Pixel l werden dabei durch das Szenen-Modell vorgegeben. Bedingt durch den Fotostrom ändert sich die Spannung $V_{\mathrm{bph},l}\,(t_i)$ somit um $\Delta V_{\mathrm{bph},l} = C_{\mathrm{pix},l} \cdot Q_{\mathrm{ph},l}$, wobei mit Unterschreiten von V_{clamp} eine entsprechende Klemmung erfolgt. Der vom Foto-FET ausgegebene Strom i_{pix} wird nur berechnet, wenn dieser für die nachfolgende Stufe erforderlich ist.

4.2.4 Implementierung

Die zur Ansteuerung der Pixelmatrix notwendigen Steuerworte werden zeilenweise durch LINECTRL (siehe Abb. 4-1) vorgegeben und bewirken jeweils einen Zustandsübergang in den zur entsprechenden Zeile gehörenden Pixeln. Die einzelnen Zeilen arbeiten zunächst vollständig unabhängig voneinander. Durch Faltungsoperationen, bei denen stets mehrere Zeilen involviert sind (siehe 3.1.3), oder globale Aktionen, wie das Rücksetzen der gesamten Matrix oder das Speichern der aktuellen Werte aller Pixel in den jeweiligen SI-Zellen, werden mehrere Zeilen auf einmal beeinflusst. Die entsprechenden Berechnungen zum Arbeitspunkt des Pixels sowie des ausgegebenen Stromes $i_{\mathrm{pix}} = i_{\mathrm{DS}}$ können parallel erfolgen.

Die vom Modell der Zeilensteuerung LINECTRL vorgegeben Steuerworte werden daher auf die einzelnen Arbeits-Threads $\mathrm{Th}_0 \ldots \mathrm{Th}_{n-1}$ eines Thread-Pools verteilt und dazu der jeweils zugehörigen Warteschlange angefügt. Die Zuordnung erfolgt abhängig von der Nummer der Zeile, wodurch die absolute Reihenfolge aller Signaländerungen pro Zeile stets erhalten bleibt. Die Berechnungen innerhalb der Pixelmatrix erfolgen somit parallel und unabhängig vom SystemC-Modell des restlichen VSoC. Eine Synchronisation ist erst notwendig, wenn die Ergebnisse tatsächlich benötigt werden, d.h. beim Rücksetzen der in den Spalten abgelegten Ladung (`rst()`), dem Auslesen des Komparator-Ergebnisses (`comp()`) oder der Ausgabe von Kompensationsladungen während der A/D-Umsetzung (`int(q)`). Genau wie bei der Änderung der globalen Bias-Spannungen wird dazu zunächst auf die Fertigstellung aller noch in den Warteschlangen der Arbeits-Threads verbleibenden Operationen gewartet.

Algorithmus	Anzahl Threads	FPS ohne SSE	FPS mit SSE
„*Global Reset*“ (zeilenweises Auslesen)	1	1,9	7,6
	2	2,1	7,6
	4	2	6,6
„*Global Shutter*“ mit Mittelung über 32 Zeilen	1	0,36	1,4
	2	0,59	1,7
	4	0,84	2,25
„*Global Shutter*“ mit Mittelung über 64 Zeilen	1	0,21	0,78
	2	0,4	1,2
	4	0,72	1,77

Tabelle 4-1: Erreichbare Geschwindigkeit des 1024×1024 Bildpunkte umfassenden Modells der Pixel-Matrix bei ausgewählten Algorithmen.

Sowohl die Pixelmatrix als auch das Szenen-Modell (`OptSrc`) besitzen eine sehr reguläre Struktur, welche die Nutzung von SIMD-Vektor-Operationen (SSE/SSE2) begünstigt. Durch eine entsprechende Umsetzung dieser besonders rechenintensiven Abschnitte konnte eine Beschleunigung von bis zu Faktor 4 erreicht werden. Die tatsächlich erreichbare Geschwindigkeit des Simulators hängt jedoch sehr stark vom implementierten Algorithmus ab. Bei zeilenweisem Auslesen der Matrix kann nicht von mehreren Threads profitiert werden, da eine Verteilung lediglich auf Zeilenebene stattfindet. Sind hingegen mehrere Zeilen an einer Operation beteiligt, so ist die Verteilung und parallele Ausführung möglich.

Werden vollständige Bilder einer bestimmten Auflösung ausgegeben, so kann der resultierende Datendurchsatz in Bildern pro Sekunde (FPS) angegeben werden. In Tabelle 4-1 ist ein Vergleich von drei verschiedenen Auslese-Modi angegeben, wobei das 1024×1024 Bildpunkte umfassende Modell der Pixel-Matrix jeweils vollständig ausgelesen wurde. Dabei wurde eine Workstation mit 2x Intel Xeon E5-2643 mit insgesamt 8 CPU Kernen eingesetzt. Bei der Betriebsart „*Global Reset*“ erfolgt nach einem globalen Rücksetzen aller Pixelzellen das zeilenweise Auslesen und Ausgeben. Da keine parallelen Berechnungen erfolgen können, wirkt sich der durch das Multithreading bedingte Mehraufwand sogar negativ aus. Bei „*Global Shutter*“ mit Mittelung über 32 bzw. 64 Zeilen werden nach dem Ende der Belichtungszeit zunächst alle Pixelwerte in den jeweiligen SI-Zellen gespeichert. Während der eigentlichen Ausgabe erfolgt schließlich die parallele Ausgabe von 32 bzw. 64 Zeilen und somit die entsprechende Mittelung. Der hohe parallele Anteil führt zu einer deutlichen Beschleunigung bei Verwendung mehrerer Threads.

4.2.5 SystemC Wrapper

Die rechenaufwändige Simulation des analogen Verhaltens innerhalb der Pixelmatrix erfolgt weitgehend unabhängig vom zeitlichen Verhalten der in SystemC implementierten Komponenten. An den Schnittstellen ist daher eine geeignete Synchronisation bzw. Adaption notwendig, um sowohl Aktionen auszulösen als auch um z.B. Ergebnisse des Komparators abzurufen. Durch die Zeilensteuerung LINECTRL wird für jede Zeile ein Steuerwort vorgegeben, dessen einzelne Bits direkt mit Schalt-Transistoren innerhalb der Pixelzellen verbunden sind. Wird durch den Wrapper eine Änderung an einem der Steuerworte festgestellt, so erfolgt ein korrespondierender Aufruf der Schnittstellenfunktion `put()` auf dem Modell der Pixel-Matrix. Ganz ähnlich erfolgt beim Rücksetzen der Spaltenkapazitäten oder der Integration während der A/D-Wandlung durch die PE in `SIMD-Proc` (siehe Abb. 4-1) ein Aufruf von `rst()` bzw. `int(q)` für die jeweils betroffenen Spalten der Pixelmatrix. Treten innerhalb eines Zeitschritts des Simulators mehrere Änderungen auf, so werden diese zunächst gesammelt und zum Ende des Zeitschritts gemeinsam an das Modell der Matrix übergeben. Wird das Komparator-Ergebnis durch mindestens ein PE aus `SIMD-Proc` vom Wrapper abgerufen, so erfolgt der Aufruf von `comp()` für alle Spalten parallel. Das Ergebnis wird im Wrapper zwischengespeichert und steht im aktuellen Zeitschritt auch allen weiteren PEs aus `SIMD-Proc` zur Verfügung.

4.3 Einsatz in Anwendungsentwicklung

4.3.1 Parametrierung und Nicht-Idealitäten

Sowohl zur Parametrierung als auch zur Injektion von Defekten und anderer, zeitunabhängiger Nicht-Idealitäten werden sämtliche Parameter der vereinfachten Ersatzschaltung (siehe 4.2.3) der Pixelzellen durch `SensorConfig` vorgegeben. Neben den Parametern des Foto-FET schließt dies auch die Diodenparameter sowie die Kapazität C_{pix} mit ein. Die entsprechenden Werte können für jede Pixelzelle individuell vorgegeben werden. Sowohl für die auf der Kapazität der Spaltenleitung abgelegten als auch für die in den SI-Zellen gespeicherten Werte wird ein exponentieller Lade-/Entladevorgang als Nicht-Idealität, verursacht durch Leckströme abhängig vom Zeitpunkt der letzten Änderung, angenommen. Die notwendigen Zeitkonstanten sind ebenfalls individuell über `SensorConfig` anpassbar. Das Rauschen der Spannung V_{BPh} nach einem Reset wird zudem näherungsweise durch eine Normalverteilung approximiert. In Tabelle 4-2 sind die variablen, durch `SensorConfig` anpassbaren Parameter aufgelistet.

4.3.2 Simulative Validierung von Algorithmen

Im Gegensatz zum Test eines Algorithmus auf der realen Hardware, bei dem lediglich die ausgegebenen Ergebnisse evaluiert werden können, erlaubt die simulative Validierung die Betrachtung des internen Zustands der digitalen und analogen Komponenten zu einem bestimmten Zeitpunkt. So kann auch der Einfluss von Defekten und Nicht-Idealitäten bestimmt und z.B. mit dem idealen Verhalten durch Abschalten von Rauscheinflüssen

Komponente	Parameter
Impuls- und Taktgenerator	Standardabweichung (*Jitter*)
Szenen-Modell	Bildabstand τ_{dist} Dateinamen der zu ladenden Bildsequenz Photonenrate pro LSB $\underline{c_l} = [c_{\text{r},l}, c_{\text{g},l}, c_{\text{b},l}]$ für jedes Pixel spektrale Empfindlichkeit $\underline{s_l} = [s_{\text{r},l}, s_{\text{g},l}, s_{\text{b},l}]^\top$ für jedes Pixel Ein-/Ausschalten der Berücksichtigung von *„Photon-Shot-Noise“*
Modell der Pixelzellen	FET Breite-zu-Längenverhältnis $\frac{w}{l}$ FET Schwellspannung ohne Bias v_{th0} FET Oberflächen-Inversions-Potential ϕ FET Body-Effekt Faktor γ FET Transkonduktanz kp FET Kanallängenmodulation λ Pixelkapazität C_{Pix} Temperaturspannung Diode U_{T} Sättigungssperrstrom Diode I_{S} Zeitkonstante $\tau_{\text{SI,leak}}$ zum Verlust der SI-Zelle durch Leckströme Zeitkonstante $\tau_{\text{SI,store}}$ zum Speichern eines neuen Wertes in der SI-Zelle
Modell der Spaltenparallelen Analogverarbeitung	Kapazität $C_{1,n}, \ldots, C_{16,n}$ der einzelnen Kondensatoren in Spalten n Zeitkonstanten $\tau_{\text{leak},Cx,n}$ für Verlust der einzelnen Kondensatoren in Spalte n durch Leckströme intrinsische Kapazität $C_{\text{col},n}$der Spaltenleitung von Spalte n selbst Zeitkonstanten $\tau_{\text{leak,col},n}$ für Verlust der intrinsischen Kapazität von Spalte n durch Leckströme

Tabelle 4-2: Auflistung der variablen, jeweils durch `SensorConfig` vorgegebenen Parameter für verschiedene Komponenten.

und Nicht-Idealitäten verglichen werden. Die Simulation kann darüber hinaus zu einem beliebigen Zeitpunkt angehalten bzw. wieder fortgesetzt werden. Zudem ist die Veränderung von Parametern (siehe Abschnitt 4.3.1) zur Laufzeit sowie die Untersuchung des Synchronisationsverhaltens der einzelnen ASIPs möglich.

Zur Untersuchung von Algorithmen wurde das Simulationssystem um ein Verfahren zur Speicherung des Zustands („*Snapshot*“) der einzelnen Komponenten zu einem bestimmten Zeitpunkt erweitert. Der in das System integrierte ISS (siehe 4.1.3) unterstützt dazu die Annotation von auszuführenden Aktionen an die einzelnen Adressen des Programmspeichers. Wird eine entsprechend markierte Adresse im Kontrollfluss erreicht, so wird die hinterlegte Aktion ausgeführt. Zur Erstellung eines *Snapshots* ist somit kein separater Befehl erforderlich, wodurch die Binär-Kompatibilität zum realen VSoC erhalten

Komponente	Im *Snapshot* gespeicherte Information
Bias	- aktuelle Bias-Spannungen
GPIO	- aktuelle Werte aller Ein- und Ausgänge
Impulsgeneratoren	- Zustand sowie aktuell eingestellte und verbleibende Zeit
Pixelmatrix	- aktueller Fotostrom i_{ph}, aktuelle Spannung am Knoten V_{bph} und resultierender Strom i_{pix} - in SI-Zelle gespeicherter Strom I_{SI} - aktuelle Ladung der einzelnen mit der Spaltenleitung verbindbaren Kondensatoren sowie die resultierende Spannung
Registerfile	- Werte aller Steuer- und Austauschregister
SIMD-Proc	- Werte aller Register und Flags
Zeilensteuerung	- aktuelle und im Speicher abgelegte Steuerworte aller Zeilen - Konfiguration der mit der Spaltenleitung verbindbaren Kondensatoren - Konfiguration der Spaltenankopplung
Zustand der ASIPs	- Flags C und Z, Programmzähler - Operanden- und Aufruf-Stack - Zustand der Synchronisation

Tabelle 4-3: Auflistung der wichtigsten, im *Snapshot* abgelegten Informationen für verschiedene Komponenten.

bleibt. In Tabelle 4-3 sind die jeweils im *Snapshot* abgelegten Informationen der einzelnen Komponenten dargestellt.

5 Bildaufnahme- und Verarbeitungsplattform

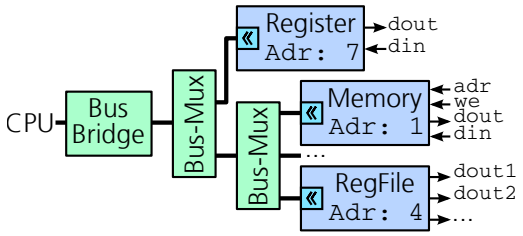
5.1 Entwicklungs- und Kamerasystem

5.1.1 Hardware / Software Schnittstelle

In modernen, FPGA-basierten SoC, wie z.B. Xilinx Zynq® [Xil16], stehen neben der eigentlichen, rekonfigurierbaren Logik, auch Mikroprozessoren zur Verfügung. Diese verfügen über genügend Rechenleistung zum Einsatz eines Betriebssystems wie Embedded Linux [Hal07]. Sollen eigene, im FPGA realisierte Hardware-Komponenten durch die auf einem integrierten Prozessor laufende Software angesteuert werden, so müssen diese zunächst mit dem Systembus des Prozessors verbunden werden. Bei der Verwendung eines Betriebssystems ist zur Ansteuerung außerdem ein entsprechender Treiber [CRKH05, QK11] erforderlich, dessen Entwurf sehr aufwändig ist.

Zur Vereinfachung der Ansteuerung eigener Hardware-Komponenten durch Linux-basierte Anwendungsprogramme wurde im Rahmen dieser Arbeit eine spezielle Infrastruktur [RD14a] entwickelt. Elementare Komponenten, wie z.B. einzelne Register, größere Register- oder Speicherblöcke oder externe SPI- bzw. I²C-Schnittstellen, werden zur Verwendung in eigenen Komponenten bereitgestellt. Jede dieser Komponenten besitzt einen eigenen Adressraum. Zur Verbindung wird eine baumartige Struktur eingesetzt, bei der die elementaren Komponenten die Endpunkte, d.h. die Blätter des Baumes, repräsentieren. Zur Verzweigung stehen spezielle Bus-Multiplexer bereit. Die Adressauflösung erfolgt dezentral durch die Endpunkte selbst, wodurch die Baumstruktur an beliebiger Stelle durch das Hinzufügen weiterer Bus-Multiplexer erweitert werden kann. Dies ermöglicht eine einfache Anpassung der FPGA-basierten Hardware, was insbesondere bei der Inbetriebnahme integrierter Schaltungen aufgrund von unerwartetem Verhalten oder geänderten Testbedingungen häufig erforderlich sein kann. Jede Basis-Komponente verfügt zudem über einen Interrupt-Eingang zur asynchronen Benachrichtigung der Steuersoftware durch die umgebende Logik. In Abb. 5-1a ist eine Baumstruktur mit drei Endpunkten und zwei Bus-Multiplexern dargestellt. Die Verbindung zum Systembus des Prozessors erfolgt durch die Komponente „*Bus Bridge*“.

Zur Ansteuerung der Komponenten wurde ein spezieller Linux Kernel-Treiber entwickelt [RD14a], der den Zugriff direkt aus dem *Userspace* erlaubt. Dieser sucht beim Start nach allen vorhandenen Komponenten und stellt Informationen bzgl. deren Art sowie der Tiefe des jeweiligen Adressraumes im SysFS [Moc05] zur Verfügung. Mittels einer Gerätedatei wird die eigentliche Schnittstelle zum *Userspace* realisiert. Der Datenzugriff geschieht schließlich durch *Memory-Mapping*. In Abb. 5-2 ist das Zusammenspiel der



(a) Beispielhafte Darstellung der baumartigen Kommunikationsinfrastruktur zur Ansteuerung von Hardware-Komponenten.

```
1  import eascbus                                          }(1)
2
3  drv=eascbus.kernel_driver()                             }(2)
4  #drv=eascus.remote_driver("host")
5
6  def irq():                                              }(3)
7    print("IRQ from Comp 1!")
8
9  c_id = 7                                                }(4)
10 c1 = eascbus.component(c_id, irq)
11
12 #write into register                                    }(5)
13 c1.write(0x7, 0xcafebabe)
14 #read 2 values from regfile
15 print(c1.read(0x4, 2))
16 ...
```

(b) Beispielprogramm zum Zugriff auf Hardware-Komponenten unter Verwendung der Python-Anbindung.

Abbildung 5-1: Gegenüberstellung der Kommunikationsinfrastruktur sowie der zu deren Ansteuerung eingesetzten Software.

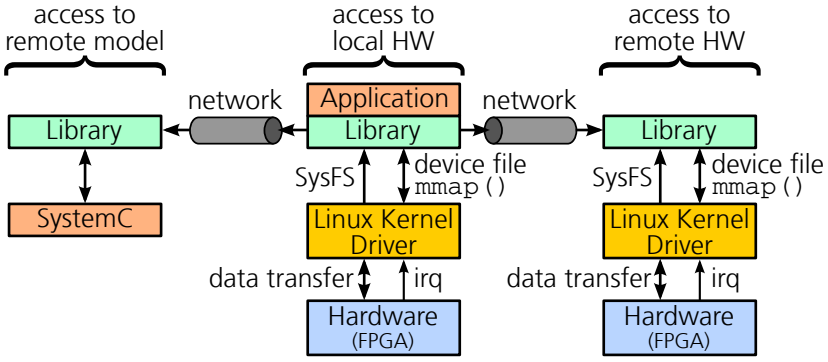


Abbildung 5-2: Darstellung der Hardware / Software Schnittstelle der Kommunikationsinfrastruktur.

Hardware / Software Schnittstelle der im Rahmen dieser Arbeit entwickelten Kommunikationsinfrastruktur dargestellt. Neben dem lokalen Zugriff („*access to local HW*“) auf die im FPGA realisierten Hardware-Komponenten wird auch der Netzwerk-basierte Zugriff auf eine entfernte Entwicklungsplattform („*access to remote HW*“) sowie auf ein SystemC Modell („*access to remote model*“) unterstützt.

Als Basis für die Anwendungsentwicklung steht außerdem eine objektorientierte Bibliothek bereit, die sowohl in C/C++ als auch ausgehend von Python verwendet werden kann. Neben dem Lese- bzw. Schreibzugriff auf die Komponenten ermöglicht diese auch die Registrierung von Funktionen, deren Aufruf beim Auftreten von Interrupts automatisch erfolgt. Anstelle des direkten Zugriffs auf die Hardware mit Hilfe des Kernel-Treibers können Datenzugriffe und Interrupts auch über eine Netzwerkverbindung (LAN) zu einem entfernten PC weitergeleitet werden (vgl. Abb. 5-2). Auch die Ansteuerung von SystemC-Modellen der Hardware ist auf dem gleichen Weg möglich. Die vorgestellte

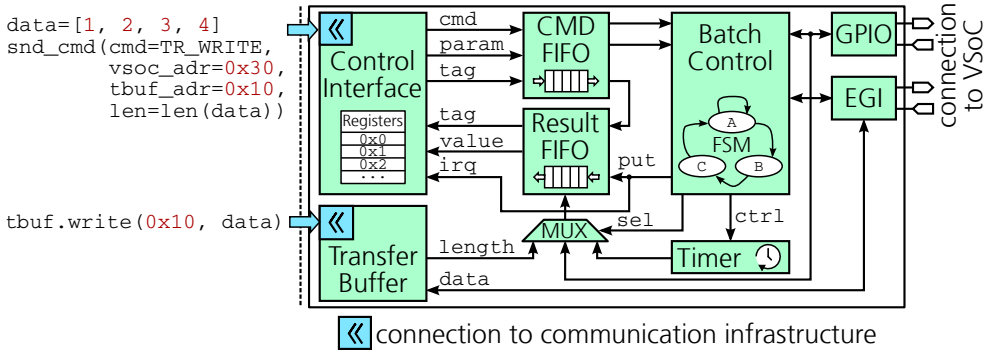


Abbildung 5-3: Darstellung der Struktur des Moduls zur Ansteuerung des VSoC.

Infrastruktur ermöglicht es somit, die gleiche Anwendungssoftware sowohl zur Ansteuerung eines Simulationsmodells als auch zum Betrieb der realen Hardware einzusetzen. In Abb. 5-1b ist ein Beispielprogramm zur Ansteuerung der Komponenten aus Abb. 5-1a angegeben. Nach dem Laden der entsprechenden Bibliothek (1) wird ein Objekt des Treibers (2) sowie eine Funktion (3) zur Reaktion auf Interrupts erstellt. Die anzusteuernde Komponenten wird schließlich durch ein Objekt (4) repräsentiert, auf welchem Schreib- und Lese-Operationen (5) ausgeführt werden können.

5.1.2 Ansteuerung des VSoC

Die eigentliche Programmierung und Parametrierung des VSoC durch die ansteuernde Software erfolgt über die SPI-basierte EGI-Schnittstelle (siehe 3.1.1) durch Zugriffe auf integrierte Register. Kernstück des im FPGA realisierten Kommunikations-Moduls bildet dabei eine Stapelverarbeitung, mit deren Hilfe ein deterministisches Zeitverhalten unabhängig von Einflüssen der Steuersoftware bzw. des Betriebssystems realisiert wird. Durch die Verwendung der in 5.1.1 dargestellten Bus-Infrastruktur wird ein Netzwerk-Proxy bereitgestellt und die Nutzung ausgehend von einem entfernten PC aus möglich.

In Abb. 5-3 ist die Architektur der Komponente zur Ansteuerung des entwickelten VSoC dargestellt. Die auszuführenden Befehle (`cmd`) werden mit ihren Parametern (`param`) und mit einer jeweils zugeordneten Kennzeichnung (`tag`) über die Bus-Infrastruktur in einem FIFO-Puffer (`CMD-FIFO`) abgelegt und nacheinander durch die Steuerung verarbeitet. Wird die Abarbeitung eines Befehls abgeschlossen, so wird dessen Kennzeichnung gemeinsam mit einem vom Befehl abhängigen Ergebnis (`value`) in den Ergebnis-FIFO-Puffer (`Result-FIFO`) geschrieben, was eine Rückmeldung zur Steuersoftware ermöglicht. Der Transfer von Daten in bzw. aus VSoC-internen Registern erfolgt durch einen separat als Speicherblock an die Bus-Infrastruktur angebundenen Transfer-Puffer. Die Quell- bzw. Zieladresse wird den Schreib- und Lese-Operationen jeweils als Parameter übergeben. Neben den Datentransferoperationen stehen auch Befehle zur genauen Zeitmessung sowie zur Ansteuerung und Überwachung der GPIO-Schnittstelle des VSoC zur Verfügung (siehe Tab. 5-1).

Gruppe	Befehl (`cmd`)	Beschreibung
Zeitmessung	TMR_START	Start der Zeitmessung
	TMR_STOP	Beenden der Zeitmessung und Ablegen der gemessen Zeit im Ergebnis-FIFO
	TMR_RST	Rücksetzen der Zeitmessung
GPIO	SET_GPI	Vorgabe der Belegung der Eingangs-Pins (GPI) des VSoC
	GET_GPO	Abrufen der aktuellen Ausgangs-Pins (GPO) des VSoC
	WAIT_GPO	Warten auf ein Ereignis am Ausgangs-Pin (GPO) des VSoC
Registerzugriff	TR_WRITE	Übertragen eines Speicherbereichs aus dem Transfer-Puffer in VSoC-interne Register
	TR_READ	Übertragen eines Speicherbereichs von VSoC-internen Registern in den Transfer-Puffer.

Tabelle 5-1: Auflistung der zur Ansteuerung des VSoC unterstützten Befehle.

5.1.3 Bildspeicherverwaltung

Die in Abschnitt 5.1.1 dargestellte Bus-Infrastruktur dient zur Steuerung und Parametrierung von Komponenten, ist jedoch nicht zum schnellen Transfer großer Datenblöcke geeignet [RD14a]. Vom VSoC ausgegebene Bild- oder Merkmalsdaten werden jedoch ggf. zunächst im FPGA weiterverarbeitet, wobei auch Referenzbilder oder Korrekturdaten in die Berechnungen einfließen können. Zur Abstraktion der erforderlichen Speicherzugriffe sowohl durch die Hardware- als auch durch die Software, wird eine Bildspeicherverwaltung bereitgestellt, deren Architektur in Abb. 5-4 dargestellt ist.

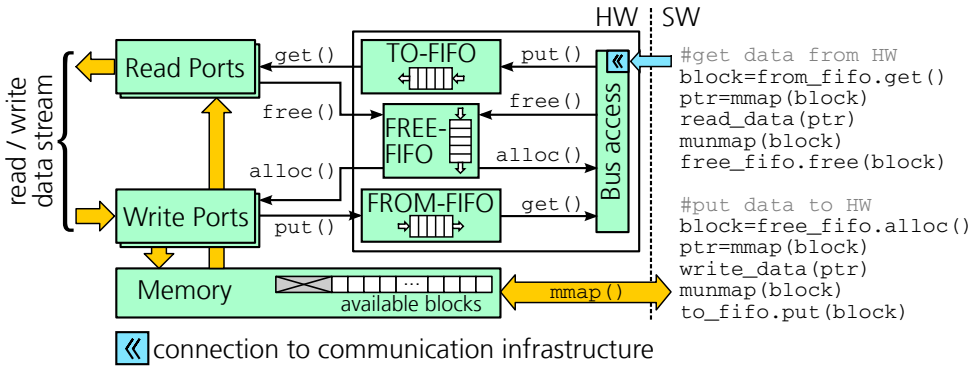


Abbildung 5-4: Darstellung der Architektur der Bildspeicherverwaltung sowie der zugehörigen Ansteuerung aus der Software (Pseudocode).

Der zur Verfügung stehende Speicher, auf den sowohl ausgehend von der Hardware- als auch von der Softwareebene zugegriffen werden kann, wird zunächst in Blöcke gleicher Größe unterteilt, wobei ein Teil des Speichers für das Betriebssystem exklusiv zur Verfügung steht. Der Transfer von Daten zwischen Hard- und Software-Ebene erfolgt jeweils auf der Basis eines Blocks. Der Austausch der zugehörigen Start-Adressen der Blöcke sowie diverser Zusatzinformationen, wie der Anzahl gültiger Daten, geschieht in beide Richtungen jeweils über an die Bus-Infrastruktur angekoppelte FIFO-Puffer (`TO-FIFO` bzw. `FROM-FIFO` in Abb. 5-4). Die Adressen nicht verwendeter Blöcke werden in einem separaten FIFO-Puffer (`Free-FIFO`) abgelegt. Sowohl der Hardware- als auch der Softwareebene ist somit die Allokation `alloc()` eines leeren bzw. die Freigabe `free()` eines nicht mehr benötigten Blocks durch Schreib- bzw. Leseoperationen auf dem `Free-FIFO` möglich. Während der eigentliche Datenzugriff auf Softwareebene durch *Memory-Mapping* `mmap()` des Speichers eines Blocks geschieht, stehen auf Hardwareebene Komponenten zum Schreiben (Write-Ports) bzw. Lesen (Read-Ports) zusammenhängender Datenströme bereit. Neben der Weiterleitung aufgenommener Bild- und Merkmalsdaten von der Hardware- in die Softwareebene können auch in die umgekehrte Richtung Datenblöcke (z.B. Korrekturparameter) übertragen werden.

5.1.4 Remote-Zugriff und Ankopplung des Simulators

In Abb. 5-5 ist die Interaktion zwischen der Anwendungssoftware und dem Entwicklungssystem (*Development System*, DS) bzw. der Simulationsumgebung dargestellt. Eine FPGA-basierte Plattform bildet die Grundlage des DS und enthält die Schnittstelle zur Ansteuerung des VSoC (siehe 5.1.2), die Bildspeicherverwaltung (siehe 5.1.3) sowie ggf. weitere Verarbeitungsfunktionen. Zur Verwendung der in Kapitel 4 vorgestellten Simulationsumgebung, komplementär zum DS, wurden dessen wesentliche Komponenten ebenfalls mit Hilfe von SystemC modelliert (vgl. 4.1.2).

Die eigentliche Anwendungssoftware zur Ansteuerung des VSoC sowie aller Komponenten des DS kann sowohl auf der in das DS integrierten CPU ((1) in Abb. 5-5) als auch auf einem entfernten PC ausgeführt werden. Sowohl dessen Parametrierung und Konfiguration als auch der Transfer von Bild- und Merkmalsdaten erfolgen dabei transparent über Netzwerkverbindungen ((2) und (3)). Das Simulationssystem stellt die gleichen Schnittstellen wie das DS bereit und kann daher unter Verwendung der gleichen Software zur Ansteuerung und Weiterverarbeitung der Daten alternativ zum DS eingesetzt werden.

5.2 Programmierung des ASIP-basierten Steuerwerks

5.2.1 Übersicht des Entwurfsflusses

Zur Umsetzung konkreter Bildverarbeitungsaufgaben unter Verwendung des VSoC (siehe 3.1.1) ist die Partitionierung der Aufgabenstellung in VSoC-basierte Verarbeitungsschritte und möglicherweise konventionelle Nachverarbeitung erforderlich. Die durch die Funktionseinheiten des VSoC in analoger oder digitaler Domäne umzusetzenden Operationen werden durch Programmierung des Multi-ASIP-basierten Steuerwerks (siehe 3.3)

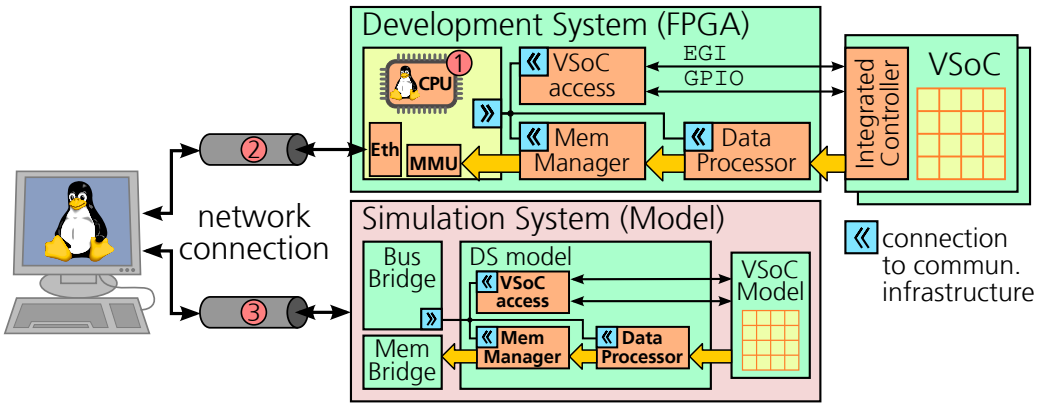


Abbildung 5-5: Darstellung der Interaktion zwischen Anwendungssoftware und Entwicklungs- bzw. Simulationsumgebung (aus [RDH+16]).

implementiert. Die heterogene Architektur der Plattform setzt dabei eine geeignete Abstraktion der Software-Komponenten zum Entwurf und zur Implementierung von Bildverarbeitungsalgorithmen voraus. Das im Rahmen dieser Arbeit entwickelte Konzept [RDP+16] zur ganzheitlichen Betrachtung von Bildverarbeitungsaufgaben, bestehend aus VSoC basierter Bildaufnahme und -vorverarbeitung sowie konventioneller Nachverarbeitung, wird in diesem sowie in den folgenden Abschnitten genauer betrachtet.

Der Entwurfsfluss zur Umsetzung von VSoC-basierten Bildverarbeitungsaufgaben ist in Abb. 5-6 als Übersicht dargestellt und lässt sich grob in drei Domänen unterteilen: Die untere Domäne (Abb. 5-6 (c)) repräsentiert das VSoC mit dem integrierten, Multi-ASIP-basierten Steuerwerk und der Unterstützung mehrerer, z.T. voneinander abhängiger Kontrollflüsse. Die zweite, mittlere Domäne ist ein Assemblersystem, das hinsichtlich der unterstützten Befehlssätze leicht an die speziellen Erfordernisse eines auf mehreren ASIPs basierten Steuerwerks angepasst werden kann. Die einzelnen Kontrollflüsse können zusammenhängend, d.h. in einer gemeinsamen Eingabedatei, beschrieben werden, um Abhängigkeiten zwischen den ASIPs auszudrücken, aber auch um Optimierungen über mehrere ASIPs zu ermöglichen. Die obere Domäne (a) enthält schließlich eine auf der Hochsprache Python [Pyt16, PW09] basierende Programmierumgebung. Diese erlaubt die Zuordnung einzelner Programmabschnitte („*VSoC sections*“) zu spezifischen ASIPs und die direkte Transformation einer Teilmenge der Programmiersprache Python in die jeweilige Assemblersprache. Sowohl in der Assembler- als auch in der Python-Ebene stehen, zur weiteren Abstraktion des Funktionsumfangs des zugrundeliegenden VSoC, Bibliotheken mit Funktionen zur Bildverarbeitung sowie zur Steuerung der Bildaufnahme bereit. Diese dienen als Grundlage zur Umsetzung konkreter Bildverarbeitungsaufgaben. Zur weiteren Verarbeitung der vom VSoC ausgegebenen Daten können Programmabschnitte auf dem Host („*host sections*“) ausgeführt werden, in denen existierende Bibliotheken wie z.B. OpenCV [BK08] zum Einsatz kommen.

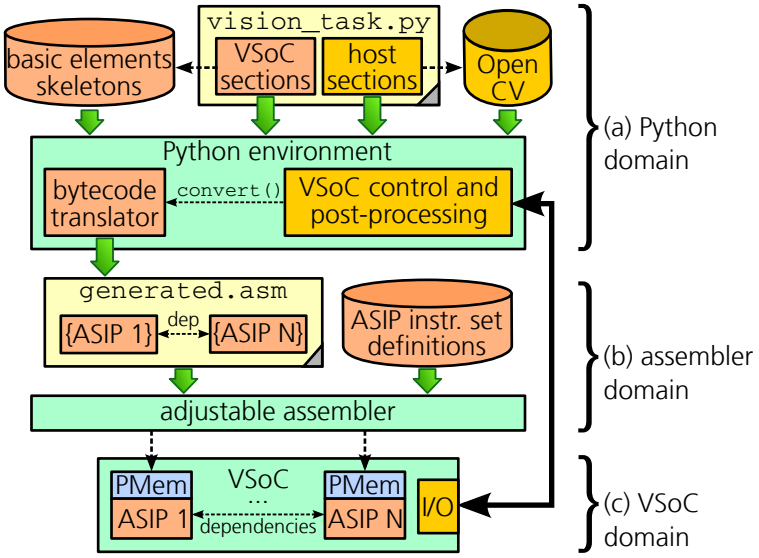


Abbildung 5-6: Zusammenhängende Darstellung des Entwurfsflusses zur High-Level Programmierung des Multi-ASIP-basierten Steuerwerks (aus [RDP+16]).

5.2.2 Multi-ASIP Assemblersystem

Architektur des Assemblers

Die Programmierung applikationsspezifischer Prozessoren erfordert parametrierbare Entwicklungswerkzeuge, die leicht an individuelle Befehlssatzerweiterungen angepasst werden können [IL06, HML07]. Im Falle eines heterogenen Systems mit mehreren, miteinander kommunizierenden ASIPs mit unterschiedlichen Befehlssätzen müssen zudem Abhängigkeiten zwischen den einzelnen Kontrollflüssen abgebildet werden. Speziell für heterogene Multi-ASIP-Architekturen wurde daher ein Assemblersystem entwickelt, das mehrere parallele Kontrollflüsse verschiedener Prozessoren mit jeweils unterschiedlichen Befehlssätzen unterstützt. Die Befehlssätze selbst sind dabei nicht fest vorgegeben, wodurch individuelle Anpassungen auch an zukünftigen Architekturen jederzeit möglich sind.

Die Architektur des Assemblersystems ist in Abb. 5-7 dargestellt. Dieses umfasst, neben dem eigentlichen Parser zur Verarbeitung der Eingabedaten („`input.asm`“), auch ein Modul zum Laden und Speichern von Bibliotheken zur Bereitstellung prozessorspezifischer Befehlssätze („*library import/export*“) sowie die eigentliche Erzeugung der Ausgabedaten („*output generator*“). Alle dem Assembler bekannten Symbole sind einem Namensraum (*Namespace*) zugeordnet, wobei jeder Namensraum einem bestimmten Prozessor in der Zielplattform entspricht. Im sog. „T-Pool“ (*type-pool*, siehe Abb. 5-7) werden zunächst Integer- oder String-Konstanten registriert, jeweils einem Bezeichner zugeordnet und ggf. zu Gruppen zusammengefasst. So können z.B. die Register „`A`“ und „`B`“ eines Prozessors die Gruppe „`reg`“ bilden. Neben elementaren Typen wie Zahlen oder Zeichenketten dienen diese Gruppen zur Beschreibung der Parameter von im „F-Pool“ (*function-pool*)

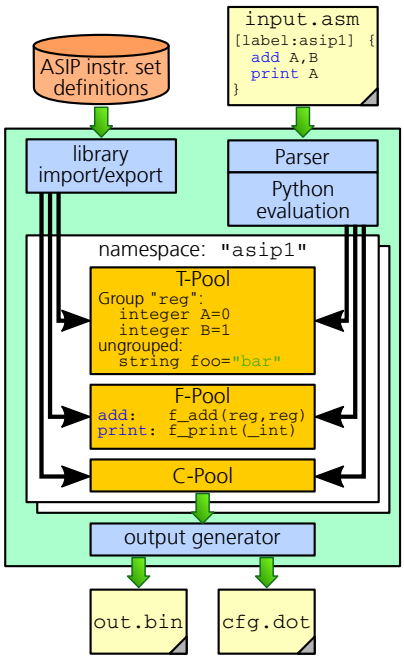


Abbildung 5-7: Darstellung der Architektur des Assemblersystems (aus [RDP+16]).

zu registrierenden Funktionen (z.B. „`f_add(reg, reg)`“). Diese Funktionen realisieren die eigentliche Übersetzung der Eingabedaten und besitzen jeweils eine Kurzbezeichnung, die später als Mnemonic bei der Verwendung in Assembler-Programmen dient. Die gleiche Kurzbezeichnung kann dabei für verschiedene Funktionen verwendet werden, wenn sich diese hinsichtlich der Typen ihrer Parameter unterscheiden. So kann z.B. das Mnemonic „`add`“ sowohl zur Kennzeichnung einer Funktion „`f_add(a : registers, b : registers)`“ als auch einer Funktion „`add_immediate(a : registers, b : int)`“ dienen.

Anstelle der direkten Konvertierung der Eingabedatei in den jeweiligen Maschinencode der Prozessoren, werden bereits verarbeitete Programmabschnitte intern in Form von Kontrollflussgraphen (CFG) [Tei13] im C-Pool (*CFG-pool*) abgelegt. Die Einträge von T-, F- und C-Pool können zudem in Bibliotheken exportiert bzw. aus diesen importiert und somit in die aktuelle Umgebung eingefügt werden. Dies ist insbesondere bei der Erweiterung und Anpassung des Assemblers von Bedeutung. Zur Ausgabe (*output generator* in Abb. 5-7) des eigentlichen Maschinencodes werden Abhängigkeiten zwischen den im C-Pool hinterlegten CFGs aufgelöst und diese schließlich in das gewünschte Ausgabeformat (z.B. Binärdaten, HEX-Format, DOT-Format [Gra16] zur Visualisierung der CFGs, ...) konvertiert.

```
1  .import "asip1"
2  .import "asip2"
3
4  [main1:asip1] {
5    ...
6    [lbl1]
7    add A,B
8    print A
9    jmpnc [lbl1]
10   sync_notify "asip2"
11   call [func]
12   ...
13 }
14 [func:asip1] {
15   ...
16   ret
17 }
18 [main2:asip2] {
19   ...
20   sync_wait "asip1"
21   ...
22 }
23
24 [tab1:asip1]
25 table <? sqrt(25) ?>
```

library imports

code segment for asip1

another code segment for asip1

code segment for asip2

table for asip1

(a) Beispiel für ein Programm mit Code für zwei ASIPs.

```
1  #import library into namespace "asip1"
2  [:asip1] .import "asip_base"
3
4  [cmd_def:asip1] {
5    <?python
6      def f_add(r1, r2):
7        n=cdfg.insert_node("add")
8        n.sem.opc("1001 10%d%d" % (r1, r2))
9        ...
10
11     #register new function with 2 param
12     regmn("add", "reg, reg", f_add)
13   ?>
14 }
```

(b) Beispiel zur Anpassung des Assemblers durch Registrierung einer Funktion für den Prozessor „`asip1`“.

Abbildung 5-8: Format der Eingabedateien des Assemblersystems.

Parser und generelle Sprachdefinition

Zur Strukturierung von Assembler-Programmen werden Blöcke und Sub-Blöcke beliebiger Verschachtelungstiefe gebildet, wobei, wie in der Programmiersprache C, geschweifte Klammern zur Kennzeichnung verwendet werden. Die Blöcke der oberen Ebene teilen ein Programm in Abschnitte und werden einem Namensraum und somit einem Prozessor zugeordnet. Für jeden dieser Abschnitte werden im C-Pool eigene CFGs angelegt. Deren Identifikation erfolgt mittels Bezeichnern, die nach dem Schema `[label:namespace]` aufgebaut sind. Zur Bezeichnung von Sub-Blöcken entfällt die Angabe des Namensraumes, diese sind nur innerhalb des aktuellen Programmabschnittes sichtbar.

Zur Verarbeitung einer Eingabedatei (siehe z.B. Abb. 5-8a) wird diese zunächst vom Parser in einzelne Blöcke zerlegt, wobei jeder Block eine Sequenz von Ausdrücken repräsentiert (z.B. „*code segment for asip1*“). Ein Ausdruck besitzt den Aufbau „`mnemonic par1, par2, ...`“ (z.B. Abb. 5-8a, Zeile 7: „`add A, B`“) und kann ebenfalls durch einen Bezeichner identifiziert werden (z.B. Zeile 6: „`[lbl1]`“). Anhand des jeweiligen Mnemonics sowie der Typen der angegebenen Argumente erfolgt die Zuordnung zu entsprechenden Funktionen des F-Pools. Ein Eingabeprogramm wird daher als Folge prozessorspezifischer Funktionsaufrufe interpretiert. Neben Einträgen des T-Pools und expliziten Konstanten können auch Bezeichner (z.B. Zeile 11: „`call [func]`“) sowie eingebettete Python-Ausdrücke (z.B. Zeile 25: „`table <? sqrt(25) ?>`“) angegeben werden. Spezielle Mnemonics (z.B. Zeile 1: „`.import 'asip1'`“) dienen dem Aufruf von Funktionen zur Deklaration neuer Typen und Konstanten, zum Import von Bibliotheken sowie zur Steuerung des Assemblers.

Erweiterung und Anpassung des Assemblers

Zur Erweiterung und Anpassung des Assemblers für eine bestimmte Prozessorarchitektur, wird zunächst ein entsprechender Namensraum angelegt. Zudem sind die Typen und Funktionen zur Abbildung der einzelnen Instruktionen in den T- und F-Pools zu registrieren. Dies wird durch die direkte Einbindung von Programmabschnitten der Programmiersprache Python in eine Eingabedatei erreicht (siehe Abb. 5-8b), wodurch sowohl zur Nutzung des Assemblers als auch zu dessen Anpassung, lediglich eine einzige Eingabesprache notwendig ist. Neben der Definition neuer Instruktionen kann dieses Vorgehen auch zur Erstellung komplexer Makros eingesetzt werden. Grundsätzlich kann jede Eingabedatei eingebettete Programmabschnitte enthalten.

Zum Anlegen einer neuen Assembler-Instruktion wird zunächst eine Funktion mit der entsprechenden Anzahl Parametern deklariert (Zeile 6 in Abb. 5-8b) und schließlich unter Angabe der Kurzbezeichnung sowie der Typen der einzelnen Parameter am F-Pool registriert (Zeile 12 in Abb. 5-8b). Die Funktion selbst erstellt bei ihrem Aufruf entsprechende Knoten im aktuellen CFG, jeweils abhängig von den übergebenen Argumenten.

Die Einträge der T-, F- und C-Pools werden in Bibliotheken exportiert und können somit direkt in weiteren Programmen eingebunden werden. Durch Importieren einer Bibliothek in einen bestimmten Namensraum (Zeile 2 in Abb. 5-8b) wird die Definition und Wiederverwendung eines gemeinsamen Befehlssatzes als Basis der einzelnen ASIPs des VSoC ermöglicht.

Kontrollflussgraphen

Die Verwendung von CFGs als Zwischenformat ermöglicht spezielle Operationen wie

- die Wiederholung von Blöcken,
- das Einbetten anderer CFGs sowie
- die Annotation von Attributen an Teil-Graphen.

Zudem können nicht erreichbare Abschnitte identifiziert werden. Wird die zu einer Assembler-Instruktion gehörende Funktion aufgerufen, so wird der aktuelle CFG durch neue Knoten und Verbindungen erweitert. Während für einfache Instruktionen lediglich ein einzelner Knoten eingefügt und direkt mit dem Vorgänger verbunden wird, wird z.B. bei bedingten Sprungoperationen, auch ein Verweis auf das Sprungziel hinterlegt.

Knotenverweise können sich sowohl auf den gleichen CFG als auch auf andere CFGs beziehen. Da die Assemblerbeschreibung Code für mehrere Prozessoren enthält, steht somit beispielsweise auch die Adresse eines Blocks im Programmspeicher eines anderen Prozessors zur Verfügung. Dies kann im Rahmen der Inter-Prozessor-Kommunikation verwendet werden, um einen anderen ASIP anzuweisen, ein bestimmtes Unterprogramm aufzurufen.

5.2.3 High-Level Programmierung in Python

Konzeptbeschreibung

Die Verwendung einer Hochsprache stellt, insbesondere zur Implementierung des Kontrollflusses, eine deutliche Vereinfachung ggü. einem reinen Assemblerprogramm dar. Aufgrund der Plattformunabhängigkeit und der Vielzahl der verfügbaren Bibliotheken wurde dafür Python ausgewählt. Während bei *Micropython* [Geo16] eine einfache virtuelle Maschine (VM) auf einem Mikrocontroller zum Einsatz kommt, ist dies bei den im VSoC (siehe 3.1.1) eingesetzten ASIPs aufgrund deren begrenzten Ressourcen nicht möglich. Jedoch arbeiten sowohl der den ASIPs gemeinsame Prozessorkern (siehe 3.3.2) als auch die VM der Python Referenzimplementierung *CPython* [Pyt16] nach einem Stack-basierten Prinzip. Dies ermöglicht die Transformation einer Teilmenge von Python, basierend auf der Übersetzung der von *CPython* generierten Bytecodes, wobei sowohl Funktionsaufrufe als auch Schleifen (`while`) und Bedingungen (`if/elif/else`) eingeschlossen sind. Alle Berechnungen und Vergleiche sowie Parameter und Rückgabewerte von Funktionsaufrufen beschränken sich allerdings auf Ganzzahl-Verarbeitung. Die Einbettung von Inline-Assembler-Abschnitten ist an beliebigen Stellen im Programm möglich.

Architektur

In Abb. 5-9 ist die Vorgehensweise der ebenfalls in Rahmen dieser Arbeit entstandenen Python-Übersetzung [RDP+16] als Übersicht dargestellt. Das Eingabeprogramm („`input.py`“) enthält sowohl zu übersetzenden Programmcode als auch Abschnitte zur Steuerung des eigentlichen Übersetzungsprozesses. Die Ausgabe („`generated.asm`“) erfolgt schließlich in Form von prozessorspezifischem Assemblercode.

Ähnlich dem in Abschnitt 5.2.2 beschriebenen Konzept wird für jeden Prozessor zunächst ein eigener Namensraum bereitgestellt. Einzelne Abschnitte des Python-Programms werden einem Prozessor durch Zuordnung zu dessen Namensraum zugewiesen. Dabei wird zwischen sog. *Funktionen* und *Makros* unterschieden: Während Funktionen transformierbaren und somit auf dem VSoC ausführbaren Code enthalten, der lediglich eine Teilmenge der Programmiersprache Python darstellt, gelten für Makros derartige Einschränkungen nicht. Letztere werden während des Übersetzungsprozesses i.d.R. zur Code-Generierung mittels Inline-Assembler ausgeführt und können dabei beliebige Python-Bibliotheken verwenden.

Durch Aufruf der Funktion `convert(entry_list)` (siehe Abb. 5-9), wird die Übersetzung ausgehend von der Liste der angegebenen Eintrittspunkte (z.B. „`convert([main])`“) gestartet. Die jeweils registrierten und ausgehend von den Eintrittspunkten erreichbaren Funktionen werden dazu unabhängig voneinander betrachtet. Ein Konverter (*global convertor* in Abb. 5-9) führt den Übersetzungsprozess für jede registrierte Funktion separat durch, wobei eine Analyse aller möglichen Kontrollpfade einer Funktion und deren Wirkung auf den Operanden-Stack erfolgt.

Auch Inline-Assembler-Abschnitte, die im Gegensatz zu regulären Python-Ausdrücken auf unterschiedlichen Pfaden verschiedene Stack-Layouts produzieren können, werden dabei hinsichtlich aller möglichen Pfade analysiert. Ungültige Stack-Manipulationen führen

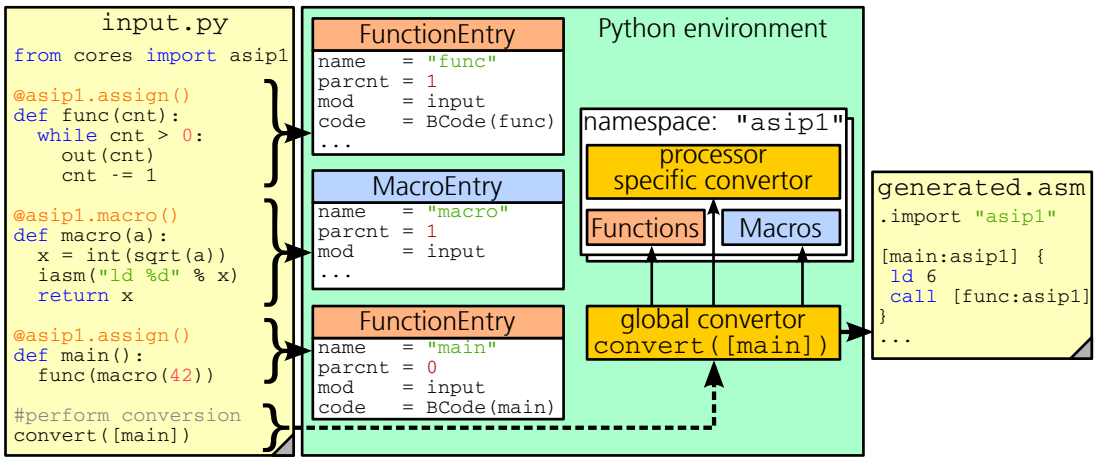


Abbildung 5-9: Darstellung der Vorgehensweise zur Transformation von Python-Programmabschnitten in ASIP-spezifischen Assemblercode (aus [RDP+16]).

entsprechend zum Abbruch der Konvertierung. Zur tatsächlichen Übersetzung einzelner Bytecodes wird schließlich ein dem jeweiligen Namensraum zugeordneter prozessorspezifischer Konverter verwendet. Die Ausgabe erfolgt in einer gemeinsamen Assembler-Beschreibung, welche für jede übersetzte Funktion einen dem jeweiligen Namensraum zugeordneten Block enthält.

Während des Transformationsprozesses kann zudem eine Optimierung hinsichtlich der Minimierung der notwendigen Befehle bzw. hinsichtlich einer Maximierung der Verarbeitungsgeschwindigkeit erfolgen. Zunächst wurden lediglich die Verfahren *„Constant-Propagation“* sowie *„Constant-Folding“* realisiert.

Zuordnung von Programmabschnitten

Die Zuordnung von Funktionen und Makros zu einem konkreten Namensraum geschieht durch Angabe eines *Decorators* [Smi03] unter Verwendung der üblichen @–Schreibweise (siehe Abb. 5-9). Wird das umgebende Python-Modul ausgeführt, so wird nach dem Anlegen eines Funktionsobjektes der zugehörige *Decorator* mit Angabe der entsprechenden Referenz auf das Funktionsobjekt aufgerufen. Der *Decorator* `@assign()` legt ein korrespondierendes `FunctionEntry` im jeweiligen Namensraum an, welches neben dem Namen und der Anzahl der Parameter auch das definierende Modul sowie den Bytecode der Funktion beinhaltet. Für Makros wird durch `@macro()` entsprechend ein `MacroEntry` angelegt, der Bytecode ist hier jedoch nicht notwendig. Auch Methoden einer Klasse können als Funktionen bzw. Makros den einzelnen ASIPs zugeordnet werden. Die tatsächliche Registrierung erfolgt dabei jedoch erst nach der Erstellung eines Objektes der jeweiligen Klasse, wobei Funktionen und Makros für jedes erstellte Objekt separat registriert werden. Innerhalb jedes Moduls bzw. innerhalb einer Klasse können Funktionen bzw. Makros beliebigen Prozessoren zugewiesen werden. Programmcode, der weder mit

`@assign` noch mit `@macro` zugewiesen wurde, wird hingegen regulär ausgeführt und als „*Host Section*“ bezeichnet. Letztere werden zur Steuerung des Übersetzungsprozesses sowie zur Parametrierung und Konfiguration des VSoC sowie zur Weiterverarbeitung von Daten verwendet.

5.3 Abbildung von Bildverarbeitungsaufgaben

5.3.1 Gemeinsame Bildvor- und -nachverarbeitung

Neben der bereits in 5.2.1 erwähnten Partitionierung in VSoC-basierte Vor- sowie konventionelle Nachverarbeitung ist zur Abbildung von Bildverarbeitungsaufgaben auch die Parametrierung der Bildaufnahme- und Verarbeitungsplattform sowie die eigentliche Programmierung des VSoC erforderlich. Insbesondere zur Synchronisation mit dem zu beobachtenden Prozess ist ein weiterer wesentlicher Bestandteil die Ansteuerung externer Sensoren und Aktoren. Die gemeinsame Betrachtung dieser Aufgabenstellungen soll im Folgenden als „*Vision Task*“ bezeichnet werden.

Für die konventionelle Nachverarbeitung können existierende Bibliotheken wie z.B. OpenCV eingesetzt werden, die Implementierungen für zahlreiche Bildverarbeitungsalgorithmen bereitstellen. Insbesondere im Bereich *Low-* und *Mid-Level-Vision* (siehe 2.1.1) kann aufgrund der geringen Komplexität und der gleichzeitig hohen Parallelisierbarkeit der Algorithmen von der Abbildung auf das VSoC profitiert werden. Dies kann ggü. konventioneller Berechnung zu höherer Verarbeitungsgeschwindigkeit bzw. geringerer Latenz beitragen. Die Auslagerung häufig benötigter Operationen bzw. Basisfunktionen zur Bildverarbeitung in Bibliotheken ist auch auf Ebene des VSoC notwendig. Der in 5.2.1 gezeigte Entwurfsfluss erlaubt die Einbindung von Bibliotheken sowohl auf Ebene des Assemblersystems als auch bei Verwendung der Hochsprache Python. Neben Funktionen zur Programmierung der Zeilensteuerung werden auch verschiedene Verfahren zur Analog-Digital-Umsetzung, ausgewählte Algorithmen zur digitalen Verarbeitung im SIMD-Array, wie das Auffinden interessanter Punkte, sowie verschiedene Datenausgabeformate bereitgestellt. Zur Gewährleistung einer möglichst hohen Verarbeitungsgeschwindigkeit wurden die meisten Basisfunktionen direkt in Assembler implementiert.

5.3.2 Abstraktion der Aufnahme- und Verarbeitungs-Schemata

Die für Bildaufnahme und -verarbeitung auf dem VSoC notwendigen Abläufe erfordern ein komplexes Synchronisationsschema zwischen den einzelnen, parallelen Kontrollflüssen des Multi-ASIP-basierten Steuerwerks (siehe 3.3) sowie der Entwicklungsplattform und ggf. weiteren externen Sensoren und Aktoren. Zur Vereinfachung der Implementierung werden parametrierbare *Skeletons* bereitgestellt, in denen generelle Abläufe zur Bildaufnahme und -verarbeitung hinterlegt werden. Die zwischen den ASIPs und der Umgebung erforderliche Synchronisation und Kommunikation wird durch die *Skeleton*s abstrahiert und bleibt dem Benutzer somit weitgehend verborgen. Ein *Skeleton* wird unter Verwen-

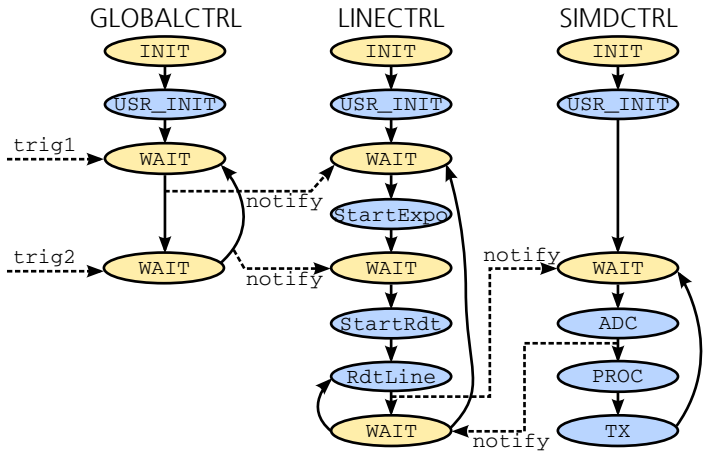


Abbildung 5-10: Vereinfachte Darstellung des Ablaufs der Synchronisation eines *Skeletons* zur Bildaufnahme durch miteinander kommunizierende Zustandsautomaten (Quelle [RDP+16]).

dung der in 5.2.3 dargestellten Methodik durch eine Python-Klasse repräsentiert, die den einzelnen ASIPs Funktionen bzw. Makros zuordnet. Zudem wird eine Schnittstelle zur Laufzeitparametrierung vorgegeben, mit der z.B. die Festlegung eines auszulesenden Bildausschnitts (*Region-Of-Interest*, ROI) erfolgen kann. Frei belegbare Adressbereiche im Register-Block des VSoC dienen dabei der Übergabe der Parameter an die Programme der einzelnen ASIPs.

Die auf den ASIPs realisierten Abläufe können als miteinander kommunizierende Zustandsautomaten betrachtet werden. In Abb. 5-10 ist hierzu ein Beispiel mit drei ASIPs GLOBALCTRL, LINECTRL und SIMDCTRL angegeben. Die einzelnen Zustände repräsentieren die verschiedenen Verarbeitungsschritte während einer Bildaufnahme. Diese hängen jeweils voneinander ab, weshalb Synchronisationspunkte, d.h. Wartezustände (`WAIT`) sowie Benachrichtigungen (`notify`), erforderlich sind. Zum Beispiel kann die A/D-Umsetzung (`ADC` in Abb. 5-10) durch SIMDCTRL erst starten, nachdem die Ausgabe der Zeile (`RdtLine`) durch LINECTRL abgeschlossen wurde. Zur Synchronisation können auch externe Steuersignale verwendet werden, z.B. zum Starten oder Beenden der Belichtungszeit (`trig1` und `trig2`).

Zur Anpassung des durch ein *Skeleton* vorgegebenen Schemas an die individuellen Anforderungen der Bildaufnahme und -verarbeitung werden *Slots* zur Registrierung von Funktionen bzw. Makros (siehe 5.2.3) an bestimmten Positionen der Verarbeitung bereitgestellt. Diese sind in Abb. 5-10 hellblau hervorgehoben. Einige *Slots* wie z.B. die A/D-Umsetzung (`ADC`) haben vordefinierte Werte und können durch den Anwender des *Skeletons* z.B. zur Auswahl eines anderen ADC-Verfahrens überschrieben werden. Neben der Auswahl von Bibliotheksfunktionen ist auch die Definition und Zuweisung eigener Funktionen möglich.

Zur Vereinfachung der Programmierung kann, anstelle miteinander interagierender Zustandsautomaten, der prinzipielle Ablauf an zentraler Stelle, z.B. durch GLOBALCTRL,

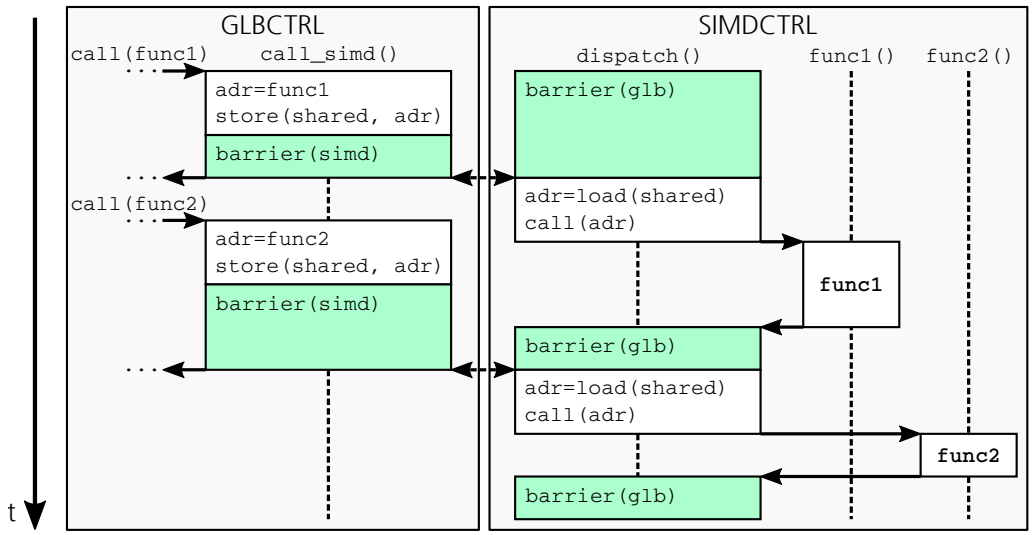


Abbildung 5-11: Darstellung der von GLOBALCTRL ausgehenden, zentralen Steuerung durch verteilte Funktionsaufrufe zu anderen ASIPs.

vorgegeben werden. Dies ist in Abb. 5-11 dargestellt. Die einzelnen von LINECTRL und SIMDCTRL auszuführenden Operationen werden jeweils in Form von Unterprogrammen, d.h. aufrufbaren Funktionen (z.B. `func1()` aus SIMDCTRL), realisiert. Deren eigentlicher Aufruf erfolgt jedoch erst auf Veranlassung durch GLOBALCTRL. Dieser wählt die aufzurufende Funktion aus und teilt deren Adresse dem entsprechenden ASIP mit Hilfe eines Austauschregisters mit. Dazu warten LINECTRL und SIMDCTRL zunächst an einem Synchronisationspunkt vom Typ „Barriere“ (siehe 3.3.3) auf die Benachrichtigung von GLOBALCTRL und lesen schließlich die übergebene Zieladresse aus. Der eigentliche Aufruf der Funktion erfolgt somit stets parallel zu anderen ASIPs. Mit Rückkehr der Funktion wird die Barriere wieder erreicht, wodurch erneut mit GLOBALCTRL synchronisiert wird.

5.3.3 Aufbau von Vision Tasks

In Abb. 5-12 ist die Python-Beschreibung eines *Vision Tasks* mit VSoC-basierter Vor- sowie konventioneller Nachverarbeitung als Beispiel angegeben. Das Programm wird auf dem PC bzw. auf dem in die Entwicklungsplattform integrierten Prozessor mit Hilfe des Python-Interpreters *CPython* ausgeführt.

Die Schnittstelle zur Bus-Infrastruktur (siehe 5.1.1) und die damit verbundene Ansteuerung des VSoC (siehe 5.1.2) sowie der Zugriff auf die Speicherverwaltung (siehe 5.1.3), werden durch ein Objekt einer Plattform-Klasse repräsentiert (siehe Abb. 5-12 (1)). Neben lokalen Zugriffen (`Platform_LocalDS`) bei autonomem Betrieb des DS ist auch eine Netzwerk-Verbindung zum DS (`Platform_RemoteDS`) bzw. zum Simulationssystem (`Platform_Sim`) möglich. Die einzelnen Plattform-Klassen stellen jeweils spezialisierte

```
1  #instance of the plattform                                    }(1)
2  plat = Platform_RemoteDS("host")
3  #plat = Platform_LocalDS()
4  #plat = Platform_Sim("localhost")
5
6  #instance of the vsoc connection                              }(2)
7  vsoc = VSoC(plat)
8
9  #own processing function                                      }(3)
10 @simd.assign()
11 def my_proc():
12   ...
13
14 #skeleton instance                                           }(4)
15 skel = skeletons.SimpleImgRead(vsoc)
16 skel.reg_slot("ADC", adc.single_slope)
17 skel.reg_slot("PROC", my_proc)
18 ...
19
20 #conversion and programing                                    }(5)
21 vsoc.program(convert_and_asm(param))
22
23 #set region-of-interest                                       }(6)
24 skel.set("roi", (0,0),(1024,1024))
25
26 #image acquisition and processing                             }(7)
27 import cv2 as cv
28 import numpy as np
29 krnl = np.ones((5,5),np.uint8)
30
31 while True:
32   img=skel.acquire_img()
33   erosion = cv.erode(img,krnl)
34   cv.imshow("Image", erosion)
35   ...
```

Abbildung 5-12: Beispiel der Python-Beschreibung eines *Vision Tasks* mit Nachverarbeitung basierend auf OpenCV.

Schnittstellen zur Steuerung der Simulation bereit. Das Objekt der Plattform-Klasse wird daraufhin (2) bei der Erstellung des Objekts zur Kapselung des VSoC übergeben.

Durch die in 5.2.3 gezeigte Methodik werden einzelne Programmabschnitte in Form von Funktionen oder Makros den ASIPs zugewiesen. Im gezeigten Beispiel wird in (3) eine Verarbeitungsfunktion angelegt. Nach der Auswahl eines *Skeletons* als Basis zur Bildaufnahme und -verarbeitung (4) wird diese im *Slot* „`PROC`“ (siehe Abb. 5-10) des *Skeletons* registriert und somit in den Ausleseprozess eingebaut. In (5) wird, ausgehend von den im *Skeleton* definierten Einsprungpunkten, eine Konvertierung der Python-Funktionen in den eigentlichen Maschinencode durchgeführt und schließlich das VSoC programmiert.

Nach der Programmierung ist ein Zugriff auf das VSoC sowohl zur Parametrierung des durch das *Skeleton* definierten Auslese- und Verarbeitungsprozesses als auch zum Abruf von Daten möglich. In (6) wird z.B. eine Funktion des *Skeletons* zum Setzen des Bildausschnitts aufgerufen. Diese übergibt die zu setzenden Optionen an das auf dem VSoC ausgeführte Programm. Im letzten Abschnitt des *Vision Tasks* (7) erfolgt der Abruf der aufgenommenen Daten sowie eine Weiterverarbeitung und Anzeige der Bilder mit Hilfe von OpenCV. Der gezeigte Ansatz erlaubt somit die ganzheitliche Betrachtung eines *Vision Tasks* bestehend aus VSoC-basierter Bildaufnahme und -vorverarbeitung sowie konventioneller Nachverarbeitung.

6 Anwendungsbeispiele

6.1 Kamerasystem

6.1.1 Aufbau und Struktur

Zur Inbetriebnahme und Charakterisierung sowie zum Einsatz des VSoC (siehe 3.1) in konkreten Anwendungen ist ein Testsystem erforderlich, das die Verwendung unter realen Bedingungen ermöglicht. Die beim Entwurf und während der Nutzung einer vorhergehenden Plattform [RD14b] zur Inbetriebnahme von Prototypen gewonnenen Erfahrungen flossen in die Entwicklung eines flexiblen und erweiterbaren Kamerasystems ein. So wurde die Elektronik des Kamerasystems anhand ihrer Funktion in einzelne Baugruppen partitioniert und auf vier Leiterplatten aufgeteilt (vgl. Abb. 6-1a):

- Das Sensor-Board zur elektrischen Ansteuerung des Bildsensors und Verbindung der Datenschnittstellen mit dem Prozessor-Board. Zudem werden die für den Bildsensor notwendigen Spannungen bereitgestellt. Durch Austausch des Sensor-Boards kann die Kamera-Plattform für verschiedene Bildsensoren angepasst werden und ist daher nicht auf den Einsatz des VSoC aus 3.1 begrenzt.
- Das Prozessor-Board, basierend auf einem FPGA mit integrierten ARM-Prozessoren vom Typ Xilinx Zynq® [Xil16] XC7Z030, bildet die zentrale Komponente sowohl zur Ansteuerung des Bildsensors als auch zur Verarbeitung der Daten und zur Kommunikation mit der Umgebung.
- Das Interface-Board stellt die externen Schnittstellen zur Kommunikation mit einem PC sowie mit zusätzlichen Sensoren und Aktoren bereit. Außerdem werden die intern erforderlichen Spannungen bereitgestellt.
- Das Interface-Raiser-Board ist eine Erweiterung des Interface-Boards und dient zur galvanischen Trennung der GPIO-Schnittstellen zwischen dem FPGA und der Außenwelt. Außerdem beinhaltet es die externen Steckverbinder sowie LEDs zur Benachrichtigung.

In Abb. 6-1b ist das Kamerasystem zudem im geschlossenen Zustand mit Gehäuse abgebildet. Dieses verfügt über die notwendigen mechanischen und optischen Schnittstellen zur Montage in industriellen Umgebungen bzw. zur Aufnahme handelsüblicher C-Mount Objektive. Die wesentlichen Daten sind in Tab. 6-1 zusammengefasst.

Im FPGA wurde das in 5.1.2 dargestellte Modul zur Ansteuerung des VSoC implementiert und mit Hilfe der Bus-Infrastruktur (siehe 5.1.1) mit dem Systembus der integrierten ARM-Prozessoren verbunden. Für die Bildspeicherverwaltung (siehe 5.1.3) stehen 768 MiB zur Verfügung, während die verbleibenden 256 MiB dem Betriebssystem

(a) Platinenstapel bestehend aus Sensor-, Prozessor- und Interface-Board sowie dem Interface-Raiser-Board.

(b) Die Kamera mit geschlossenem Gehäuse.

Abbildung 6-1: Darstellung des zur Inbetriebnahme und Charakterisierung des VSoC sowie zu dessen Einsatz unter realen Bedingungen entwickelten Kamerasystems (beide Bilder © Fraunhofer IIS).

vorbehalten sind. Die vom VSoC an dessen PARDOUT-Schnittstelle (siehe 3.1.1) ausgegebenen Daten werden vom FPGA entgegengenommen. Neben der Übergabe an die Bildspeicherverwaltung stehen Schnittstellen zur FPGA-basierten Weiterverarbeitung bereit. Ein spezielles Modul zur Belichtungssteuerung („*Acquisition Control*“) dient außerdem zur Vorgabe des exakten Zeitverhaltens der Kamera, wobei entsprechende Signalleitungen mit der GPIO-Schnittstelle des VSoC verbunden sind. Dies umfasst die Festlegung von Beginn und Ende der Belichtungszeit und der Bildwiederholrate, die Bereitstellung eines Zeilentaktes sowie unterschiedliche Möglichkeiten zur Triggerung.

6.1.2 Parametrierbare Auslese-Modi

Obwohl einige Bildsensoren zur Steigerung der Bandbreite mehrere, parallele Ausgabekanäle besitzen [Oht10], erfolgt das Auslesen der Daten aus der Matrix i.d.R. seriell, d.h. zeilen- sowie pixelweise. Auch im betrachteten VSoC aus 3.1 erfolgt nach der zeilenweisen AD-Umsetzung eine serielle Ausgabe der einzelnen Pixel über die PARDOUT-Schnittstelle. Das zeitliche Zusammenspiel zwischen Rücksetzen und Belichtung der einzelnen Pixel sowie schließlich deren Ausgabe bildet den Auslese-Modus (engl. „*shutter mode*“).

In [Rei16] wurden, basierend auf den in 5.3.2 eingeführten *Skeletons*, die Auslese-Modi „*Global Shutter*“ und „*Rolling Shutter*“ implementiert. Bei „*Global Shutter*“ werden zu Beginn der Belichtungszeit zunächst alle Pixel gleichzeitig zurückgesetzt. Nach Abschluss

FPGA	Xilinx Zynq® XC7Z030
Arbeitsspeicher	1 GiB DDR3, davon 768 MiB Bildspeicher
nichtflüchtiger Speicher	256 MBit NAND Flash sowie SD-Karten-Slot
Leistungsaufnahme	ca. 2,5 W, max. 10 W
Eingangsspannungsbereich	4,5 bis 16 V
Signalleitungen zum Sensor-Board	76
Signalleitungen zum Interface-Board	83, davon 10 für Ethernet
Abmessungen der Leiterplatten	48×48 mm^2
Abmessungen des Gehäuses (inkl. Steckverb.)	$55 \times 55 \times 68{,}7$ mm^3

Tabelle 6-1: Wesentliche Daten der Kamera- und Entwicklungsplattform.

der Belichtung erfolgt das gleichzeitige Kopieren der aktuellen Werte in die in jedem Pixel vorhandene SI-Zelle. Diese steht somit für Berechnungen in analoger Ebene nicht zur Verfügung. Während der Ausgabe wird schließlich der Wert aus der SI-Zelle ausgegeben, wodurch sowohl der Zeitpunkt zum Anfang als auch zum Ende der Belichtungszeit stets für alle Pixel gleich ist. Bei „*Rolling Shutter*“ verlaufen der Beginn der Belichtungszeit – gekennzeichnet durch das Rücksetzen der Pixel – und deren Auslesen, d.h. das Ende der Belichtungszeit, nacheinander als „Vorhänge“ über das Bild. Die je nach Gesamtdauer der Belichtung überlappenden Intervalle variieren somit von Zeile zu Zeile. Die SI-Zelle wird für das reguläre Auslesen nicht benötigt und kann somit für Berechnungen in analoger Domäne genutzt werden. Der weitere Auslese-Modus „*Global Reset*“ ist gegenüber „*Global Shutter*“ und „*Rolling Shutter*“ deutlich einfacher. Der geringe Synchronisationsaufwand erlaubt eine höhere Verarbeitungsgeschwindigkeit. Das Rücksetzen aller Pixel erfolgt gleichzeitig, jedoch steigt die Belichtungszeit mit jeder weiteren ausgelesenen Zeile entsprechend an, weshalb das Bild in Ausleserichtung heller wird.

In Abb. 6-2 ist das Synchronisationsschema für „*Global Shutter*“ angegeben, welches gegenüber der Originaldarstellung [Rei16] um zusätzliche *Slots* zur digitalen Nachverarbeitung sowie um die Ausgabe von Meta-Daten erweitert wurde. Meta-Daten werden dazu als zusätzliche Zeilen dem eigentlichen Bild angehängt. Durch die externe Belichtungssteuerung („*Acquisition Control*“ in Abb. 6-2) erfolgt die Benachrichtigung bzgl. des Beginns der Belichtung (`expo`) bzw. des Beginns des Auslesens (`rdt`) über Signale am GPIO-Port des VSoC. Die Steuerung der Abläufe, ausgehend von GLOBALCTRL, erfolgt durch die Weiterleitung von Funktionsaufrufen zu LINECTRL bzw. SIMDCTRL (siehe 5.3.2) und dem Aufruf der entsprechenden, vom Benutzer verwendbaren *Slots* (in Abb. 6-2 hellblau hinterlegt). Erst bei der Verwendung des *Skeletons* werden diesen die tatsächlich auszuführenden Operationen zugewiesen.

Verändert sich die beobachtete Szene während der Belichtungszeit, so hat das zeitliche Verhalten der verschiedenen Auslese-Modi erhebliche Auswirkungen auf die Bildaufnahme. Unter Verwendung des Simulationssystems wurden verschiedene Auslese-Modi zur Beobachtung der gleichen Szene eingesetzt. Dazu wurde eine synthetische Bildsequenz erstellt, in der ein Windrad in 360 Schritten um jeweils $\varphi = 1^\circ$ gedreht dargestellt ist.

Die Bilder wurden mit einem zeitlichen Abstand $\tau_{\text{Dist}} = 50\,\mu\text{s}$ in den Ring-Puffer des Szenen-Modells (siehe 4.2.1) geladen. Das Ergebnis ist in Abb. 6-3 für „*Global Shutter*“ und „*Rolling Shutter*“ sowie für „*Global Reset*“ dargestellt. Während bei „*Global Shutter*“ (Abb. (6-3a)) aufgrund der gleichzeitigen Aufzeichnung in allen Pixeln keine zeitabhängigen Effekte sichtbar sind, erscheinen sowohl bei „*Rolling Shutter*“ (Abb. (6-3b)) als auch bei „*Global Reset*“ (Abb. (6-3c)) deutliche Artefakte.

6.2 Schrittweise Umsetzung einer Bildverarbeitungsaufgabe

6.2.1 Präsenzdetektion

Die Feststellung der Anwesenheit von Personen innerhalb eines Raumes wird als „Präsenzdetektion“ bezeichnet und ist ein wesentliches Element zur automatischen Steuerung von Beleuchtungs- und Klimatechnik. Die Verwendung Bildsensor-basierter Verfahren erlaubt gegenüber konventionellen PIR-Sensoren (*passive infrared*) auch die Bestimmung der Position sowie der Anzahl der Personen. Zur Wahrung der Privatsphäre dürfen derartige Systeme jedoch keinesfalls die Aufnahme von Bildern ermöglichen. Die Signalverarbeitung sollte bereits im Sensor-Knoten erfolgen, wodurch lediglich Textur-basierte Merkmale extrahiert und verarbeitet werden. Neben der Berücksichtigung des Blickwinkels bei der Bildaufnahme sowie der Geometrie des Raumes ist auch eine Unterscheidung von Personen von ihren Schatten und eine Erkennung von Reflexionen oder Verdeckungen erforderlich. Eine zweidimensionale Auswertung erlaubt keine gleichzeitige Auswertung von Objektgröße und Entfernung und ist somit zur Lokalisation von Personen in Räumen nicht ausreichend. Zudem können ggf. mehrere Sensorknoten erforderlich sein. Sowohl die Leistungsaufnahme der einzelnen Knoten als auch deren Kosten sollten möglichst gering sein. Zur Minimierung des Rechenaufwands kann daher auf die tatsächliche Erkennung von Personen verzichtet und stattdessen lediglich nach sich bewegenden Objekten gesucht werden, was einer Separation in Vorder- und Hintergrund entspricht. Jeder Sensorknoten soll selbstständig die Vordergrund-Objekte erkennen und eine 3d-Rekonstruktion zur Bestimmung von Position und Größe durchführen können.

Zur Untersuchung geeigneter Verfahren zur 3d-Rekonstruktion wurde zunächst ein modellhafter Stereo-Aufbau unter Verwendung gewöhnlicher Industriekameras realisiert [Rei15]. Aus den von beiden Kameras gleichzeitig aufgenommenen Bildern werden auf dem PC Merkmale in Form interessanter Punkte extrahiert. Nach der Aufteilung in Vorder- und Hintergrundpunkte durch ein selbst-lernendes Modell erfolgt die Zuordnung der Punkte des linken zu Punkten des rechten Bildes. Anstelle von Grauwertinformationen, kommen dabei als Deskriptor für jeden Punkt geometrische Merkmale (Winkel und Abstand) der Relationen zu umliegenden Nachbarpunkten zur Anwendung. Unter Verwendung des bekannten Abstandes beider Kameras wird schließlich die 3d-Position jedes Vordergrundpunktes berechnet. Zur Wahrung der Privatsphäre muss auf die Ausgabe von Realbildern vollständig verzichtet werden, weshalb auch ein Verfahren zur Berechnung der notwendigen Merkmale unter Verwendung des VSoC angegeben [Rei15] wird. Im Folgenden wird das Vorgehen zur Extraktion interessanter Punkte genauer betrachtet.

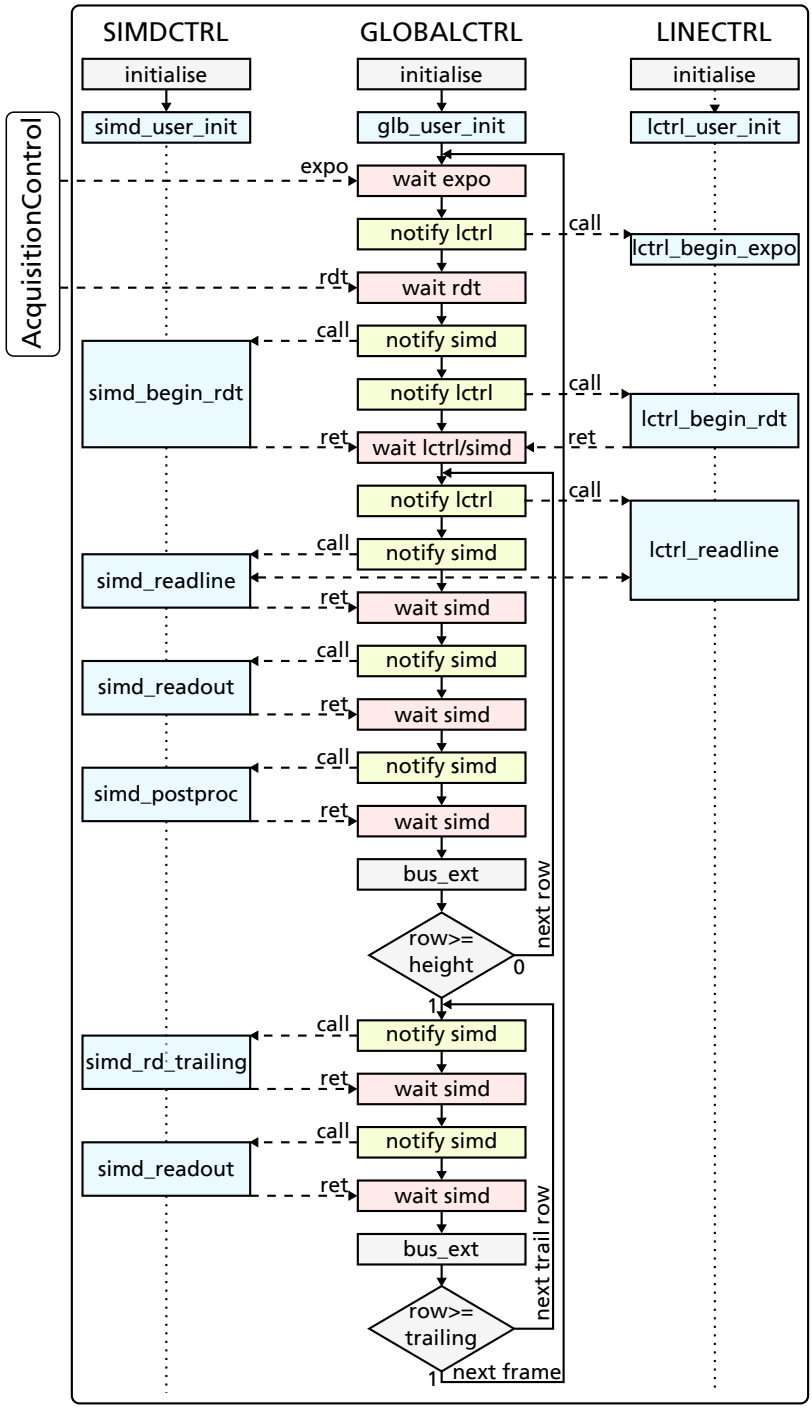
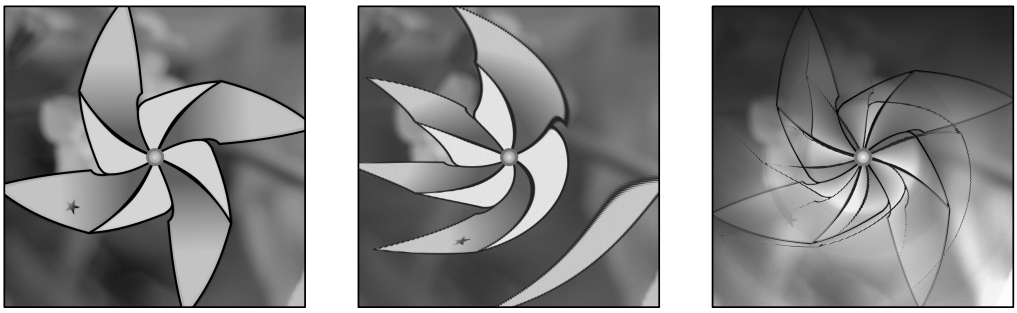


Abbildung 6-2: Vereinfachte Darstellung des Synchronisations-Schemas des Auslese-Modus „*Global-Shutter*“. Gegenüber der Originalarbeit [Rei16] um zusätzliche *Slots* sowie die Ausgabe zusätzlicher Meta-Daten am Ende des Bildes („*Trailing Lines*“) erweitert.



(a) „*Global Shutter*“ mit $\tau_{\text{Expo}} = 100\,\mu\text{s}$.

(b) „*Rolling Shutter*“ mit $\tau_{Zeile} = 10\,\mu\text{s}$ und einer Belichtungszeit von 10 Zeilen ($\tau_{Expo} = 100\ \mu s$).

(c) „*Global Reset*“ mit initialer Belichtungszeit $\tau_{Expo,0} = 1\,\mu\text{s}$ die zeilenweise jeweils um $\Delta\tau = 5\,\mu\text{s}$ ansteigt.

Abbildung 6-3: Darstellung der zeitabhängigen Effekte verschiedener Auslese-Modi bei Verwendung des Simulationssystems (aus [RDH+16]).

6.2.2 Extraktion interessanter Punkte

Interessante Punkte sind Bildmerkmale, die besondere Stellen, wie z.B. Eckpunkte oder Punkte entlang einer Kante, kennzeichnen. Zu deren Extraktion wurden verschiedene Operatoren beschrieben, wobei an dieser Stelle der sog. FAST-Operator (*Features from Accelerated Segment Test*) – vorgestellt von Rosten und Drummond [RD05, RD06] – zur Erkennung von Eckpunkten, eingesetzt werden soll. Zur Klassifikation eines Bildpunktes betrachtet der Operator um diesen alle u Pixel entlang eines Bresenham-Kreises mit bestimmtem Radius. Die einzelnen Punkte entlang der Kreisbahn werden jeweils mit dem Mittelpunkt des Kreises verglichen. Der Bildpunkt ist genau dann ein interessanter Punkt, wenn auf dem umgebenden Kreis ein zusammenhängendes Segment aus mindestens l Pixeln existiert, für das alle Pixel entweder kleiner als der Mittelpunkt abzüglich eines Offsets oder größer als dieser zuzüglich des Offsets sind. Direkt nebeneinanderliegende interessante Punkte werden anschließend unter Berücksichtigung eines geeigneten Maßes herausgefiltert (Nicht-Maxima-Unterdrückung). Abhängig vom Umfang u des Bresenham-Kreises und der Mindestlänge l der zu suchenden Segmente ergeben sich verschiedene Konfigurationen des Operators, die als FAST-l-u bezeichnet werden. Die Varianten FAST-5-8, FAST-7-12 und FAST-9-16 sind als Referenzimplementierungen in OpenCV [CV316] vorhanden.

Der zu realisierende Algorithmus (siehe Alg. 6.1) nimmt ein Bild vom Bildsensor mit bestimmter Belichtungszeit τ_{Expo} auf. Abweichend zur Original-Implementierung [Rei15], soll jedoch noch vor der Anwendung des FAST-Operators eine Korrektur defekter Pixel erfolgen. Dies wird durch Anwendung der Rangordnungs-Operation „*Median*“ [Ste08] erreicht, die als nicht-lineares Filter in der Bildverarbeitung häufig zur Reduzierung des „*Salt-and-Pepper*“ Rauschens eingesetzt wird. Im Gegensatz zur Mittelwert-Operation werden keine neuen Pixelwerte eingefügt. Zur Berechnung des Medians der 9 Eingabepixel

Algorithmus 6.1 Algorithmus zur Extraktion interessanter Punkte mit Korrektur defekter Pixel und automatischer Steuerung der Belichtungszeit.

1. Eingabe:
   - Rechteckige Region begrenzt durch die Punkte $P_1 = (x_1,y_1)$ und $P_2 = (x_2,y_2)$ zur Berechnung der mittleren Bildhelligkeit m_k,
   - das Soll-Intervall der mittleren Bildhelligkeit $m_{\text{soll}} = [m_{\text{soll,min}},m_{\text{soll,max}}]$ sowie
   - die Anfangsbelichtungszeit $\tau_{\text{Expo},0}$ für Bild zum Zeitpunkt $k = 0$.
2. Einzug eines Bildes B_k zum Zeitpunkt k mit Belichtungszeit $\tau_{\text{Expo},k}$.
3. Korrektur defekter Pixel durch Median-Filterung $B_{k,\text{Median}} = \text{Median}(B_k)$.
4. Berechnung der mittleren Helligkeit m_k der gewählten Region aus $B_{k,\text{Median}}$.
5. Berechnung der Belichtungszeit $\tau_{\text{Expo},k+1} = f_{\text{Expo}}(\tau_{\text{Expo},k}, m_k)$ für Folgebild B_{k+1} mit
$$f_{\text{Expo}}(\tau, m) = \begin{cases} \tau \cdot 1{,}25 & m = 0 \\ \tau \cdot \left(1 + \log \frac{m_{\text{soll,min}}}{m}\right) & m < m_{\text{soll,min}} \\ \tau \cdot \left(1 + \log \frac{m_{\text{soll,max}}}{m}\right) & m > m_{\text{soll,max}} \\ \tau & \text{sonst} \end{cases} \tag{6-1}$$
6. Extraktion interessanter Punkte im Bild $B_{k,\text{Median}}$ unter Verwendung des FAST-Operators mit Schwellwert $v_{\text{threshold}}$, danach weiter bei (2).

einer 3×3 Nachbarschaft, werden diese zunächst aufsteigend sortiert und schließlich der mittlere Wert ausgewählt.

Zur Adaption an sich ändernde Lichtverhältnisse erfolgt außerdem eine automatische Regelung der Belichtungszeit. Dazu wird eine rechteckige Region festgelegt, innerhalb derer der mittlere Wert der Bildhelligkeit berechnet wird. Unter Verwendung von Gl. 6-1 (siehe Alg. 6.1) wird die Belichtungszeit des Folgebildes, basierend auf dem berechneten Mittelwert, der aktuellen Belichtungszeit sowie einem Soll-Intervall der Bildhelligkeit, bestimmt.

6.2.3 Umsetzung auf Entwicklungsplattform

Der in 5.3 gezeigte Ansatz erlaubt die ganzheitliche Betrachtung eines *Vision Tasks* (siehe 5.3.3), bestehend aus VSoC-basierter Bildaufnahme und -vorverarbeitung sowie konventioneller Nachverarbeitung. Zur Lösung einer konkreten Aufgabenstellung können zunächst ein *Skeleton* zur Ausgabe vollständiger Bilder gewählt und die einzelnen Verarbeitungsschritte mit Hilfe konventioneller, digitaler Bildverarbeitung umgesetzt werden. Schritt für Schritt können daraufhin einzelne, VSoC-basierte Bildverarbeitungsoperationen aus der Bibliothek ausgewählt bzw. implementiert und in den *Slots* des *Skeletons* (siehe

6.1.2) registriert werden. Die Bildverarbeitung rückt somit näher an die Bildaufnahme heran. Durch die frühzeitige Merkmalsextraktion kann schließlich auf die Übertragung vollständiger Bilder verzichtet werden.

Zur Umsetzung von Alg. 6.1 unter Verwendung des VSoC auf der Kamera- und Entwicklungsplattform (siehe 5.1 und 6.1.1) wird ein entsprechender *Vision Task* implementiert. Zunächst werden vollständige Bilder eingezogen und unter Verwendung von OpenCV verarbeitet, weshalb die Registrierung von Funktionen zur Durchführung analoger bzw. digitaler Verarbeitungsschritte in den *Slots* des *Skeletons* nicht notwendig ist.

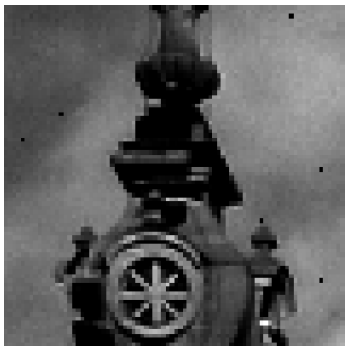
Zur Entwicklung des *Vision Tasks* wurde zunächst die Simulationsumgebung eingesetzt, weshalb eine Szene mit sich ändernden Beleuchtungsverhältnissen erforderlich ist. Dazu wurde ein Eingabebild (siehe Abb. 6-4a) ausgewählt und mit zwei unterschiedlichen Helligkeiten im Ring-Puffer des Szenen-Modells (siehe 4.2.1) abgelegt. Durch das gezielte Einfügen von Defekten und Nicht-Idealitäten (siehe 4.3.1) wurden zudem gleichverteilte Defektpixel in die Pixelmatrix eingebracht (siehe Bildausschnitt Abb. 6-4b).

In Listing 1 im Anhang A sind die wesentlichen Elemente des entsprechenden *Vision Tasks* dargestellt. Zunächst werden die Parameter zur automatischen Steuerung der Belichtungszeit sowie des FAST-Operators festgelegt (Zeilen 8 bis 17). Als *Skeleton* wurde der Auslese-Modus „*Global Shutter*“ gewählt. Nach dessen Konfiguration (Zeilen 20 und 21) erfolgt die Erstellung des Plattform-Objektes (Zeile 25) zur Kommunikation mit der Simulationsumgebung. Außerdem werden die zur Anzeige notwendigen OpenCV-Fenster initialisiert (Zeilen 28 bis 32) sowie der zur Extraktion interessanter Punkte verwendete FAST-Operator konfiguriert (Zeilen 35 bis 37). Die eigentliche Bildaufnahme und Verarbeitung erfolgt schließlich in einer Endlosschleife (Zeilen 41 bis 71). Dabei wird erst die neue Belichtungszeit gesetzt (Zeile 43), bevor die eigentliche Aufnahme eines Bildes veranlasst wird (Zeilen 44 und 45). Auf das aufgenommene Bild wird unter Verwendung von OpenCV die Median-Operation (Zeile 48) angewendet, wodurch sowohl die eingebrachten Defektpixel entfernt, als auch der Einfluss des Photonen-Rauschens minimiert wird (siehe Abb. 6-4c). Gleichzeitig verliert das Bild jedoch an Schärfe sowie an Details. Zur Berechnung der neuen Belichtungszeit unter Verwendung von Gl. 6-1 wird die mittlere Bildhelligkeit im gewählten Bildausschnitt berechnet (Zeilen 51 bis 55). Schließlich erfolgt die Anwendung des FAST-Operators zur Extraktion interessanter Punkte auf dem Median gefilterten Bild (Zeile 58) sowie, zum Vergleich, auf dem Originalbild (Zeile 63). Zum Schluss erfolgt die Darstellung von originalem und gefiltertem Bild sowie jeweils mit hervorgehobenen FAST-Punkten (Zeilen 68 bis 71). In Abb. 6-4d und 6-4e sind die entsprechenden Bilder mit hervorgehobenen FAST-Punkten dargestellt. Wird auf die Anwendung der Median-Operation verzichtet, so werden, bedingt durch die eingebrachten Defektpixel sowie das Photonen-Rauschen, deutlich mehr Punkte als FAST-Punkte erkannt.

6.2.4 SIMD-basierte Median-Berechnung

Zur optimierten Berechnung des Median-Wertes einer 3×3 Nachbarschaft stellen Vega-Rodríguez et al. [VRSPGP02] eine Architektur basierend auf einem *„minimum exchange*

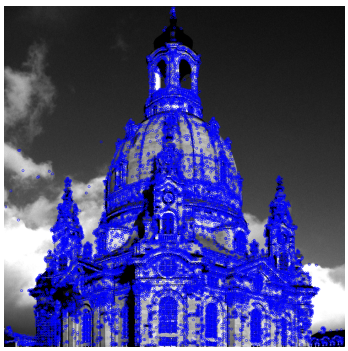
(a) Eingabebild mit Hervorhebung des in (b) und (c) betrachteten Bildausschnittes.



(b) Bildausschnitt mit eingefügten Defektpixeln (schwarze Punkte)

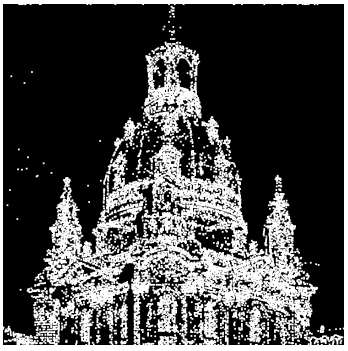
(c) Bildausschnitt nach Median-Operation

(d) FAST-Punkte (6174) nach Median



(e) FAST-Punkte (15323) ohne Median

(f) FAST-Punkte (5467) mit VSoC



(g) FAST-Punkte (12681) mit VSoC ohne Median

Abbildung 6-4: Ein- und Ausgabebilder des *Vision Tasks* zur Extraktion interessanter Punkte mit Hilfe des FAST-Operators.

Eingabebild der Simulation war „100130 150006 Dresden Frauenkirche winter blue sky-2“ von Netopyr, verfügbar unter CC BY-SA 3.0, `https://commons.wikimedia.org/wiki/File:100130_150006_Dresden_Frauenkirche_winter_blue_sky-2.jpg` (2016-03-11) – Diese Abbildungen werden veröffentlicht unter CC BY-SA 3.0.

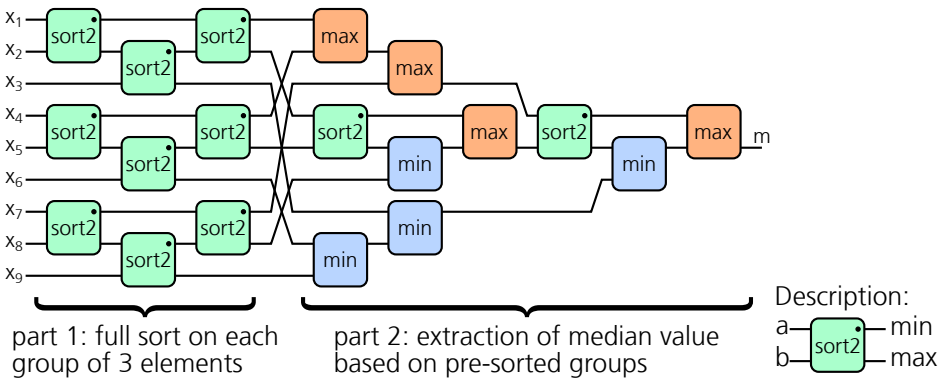


Abbildung 6-5: Darstellung der optimierten Median-Berechnung nach Vega-Rodríguez et al. [VRSPGP02].

network“ vor. Die Arbeit geht zunächst von einer Implementierung als systolisches Array aus, das, basierend auf paarweisen Sortieroperationen mit

$$\text{sort2}(a,b) = \begin{cases} (a,b) & a \geq b \\ (b,a) & a < b \end{cases}$$

die 9 Eingabepixel $X_1, \ldots, X_9$ aufsteigend sortiert. Dies erfordert insgesamt 41 sort2-Operationen. Da jedoch zur Median-Berechnung lediglich der mittlere Wert ausgewählt wird, kann das Netzwerk vereinfacht werden. Die schließlich vorgestellte Variante benötigt nur 19 Operationen, wovon lediglich 11 paarweise Sortierungen sowie jeweils 4 Operationen zur Auswahl des Minimums bzw. des Maximums ausreichen (siehe Abb. 6-5).

Zur Umsetzung im digitalen SIMD-Array wurde die von Vega-Rodríguez et al. vorgeschlagene Architektur ausgewählt. Dabei wurde das in Abb. 6-6 dargestellte Prinzip verwendet. Die letzten drei Eingabezeilen werden für alle k Spalten jeweils in den Registern r_1, r_2 und r_3 bereitgestellt und repräsentieren somit k Spaltenvektoren der Größe 3×1. Diese Spaltenvektoren werden zunächst vorsortiert (entspricht „part 1“ in Abb. 6-5), so dass $r_1 \leq r_2 \leq r_3$ gilt. Zur Extraktion des Median-Wertes in jeder Spalte, gehen jeweils der lokale, vorsortierte Spaltenvektor, als auch die Vektoren der linken und rechten Nachbarspalten ein (entspricht „part 2“ in Abb. 6-5). Da die Ergebnisse der Vorsortierung somit mehrfach verwendet werden, sind pro Spalte lediglich 5 paarweise Sortierungen sowie jeweils 4 Operationen zur Auswahl des Minimums bzw. des Maximums erforderlich, d.h. insgesamt 13 Operationen.

Die angegebene Architektur wurde direkt in Assembler implementiert und als Bibliotheksfunktion bereitgestellt. Für die Berechnungen einer Zeile werden jeweils 72 Takte benötigt, was bei den 1024 Spalten des VSoC ca. 0,07 $^{\text{Takte}}/_{\text{Pixel}}$ entspricht.

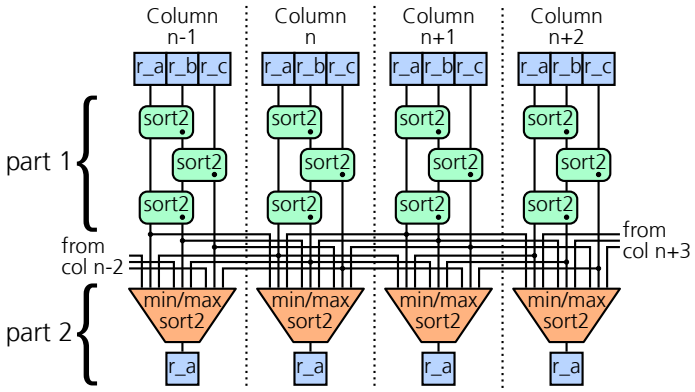


Abbildung 6-6: Darstellung zur Abbildung der Median-Berechnung nach Vega-Rodríguez et al. auf die SIMD-Architektur.

6.2.5 Abbildung unter Verwendung des VSoC

Die Median-Berechnung sowie die Extraktion interessanter Punkte sollen auf dem VSoC implementiert werden und somit näher an die Bildaufnahme heranrücken. Listing 2 auf Seite 97 repräsentiert den resultierenden *Vision Task*. Zunächst erfolgt die Parametrierung des Algorithmus (Zeilen 9 bis 18) analog zu Listing 1 auf Seite 96 sowie die Auswahl und Konfiguration eines *Skeletons* (Zeilen 21 und 22 in Listing 2). Daraufhin werden mehrere Routinen zur Integration in den Bildaufnahme und -verarbeitungsprozess erstellt und an den *Slots* des *Skeletons* registriert (Zeilen 27 bis 62).

Durch die Registrierung der Routine `simd_postproc()` (Zeilen 41 bis 62) können digitale Verarbeitungsoperationen unmittelbar nach der AD-Wandlung und noch vor der Datenausgabe integriert werden. Das Ergebnis der letzten AD-Wandlung wird im SIMD-Register r_1 bereitgestellt. Zunächst werden die Werte der innerhalb der FAST-Operation eingesetzten und in anderen Berechnungen ebenfalls benötigten Registern r_2, r_3 und r_4 gesichert. Daraufhin wird der für die Median-Berechnung notwendige Spaltenvektor in den Registern r_1, r_2 und r_3 abgelegt und im Anschluss die eigentliche Median-Operation (siehe 6.2.4) `simd_median()` ausgeführt, deren Ergebnis wiederum in r_1 steht.

Werden lediglich Merkmale anstelle vollständiger Bilder übertragen, so muss die Berechnung des zur automatischen Steuerung der Belichtungszeit erforderlichen Mittelwertes der gewählten Region bereits vor der Anwendung des FAST-Operators erfolgen. Da eine Addition aller Pixel einer Zeile im SIMD-Array jedoch sehr aufwändig ist, werden stattdessen lediglich spaltenweise Summen berechnet. Die Höhe der gewählten Region wird auf maximal 256 Zeilen beschränkt, wodurch für die Addition bei 8 Bit AD-Wandlung 16 Bit Register erforderlich sind. Diese müssen zu Beginn des Ausleseprozesses zunächst zurückgesetzt werden, was durch die *Slot*-Routine `simd_user_init_frame()` (Zeilen 27 bis 31) geschieht. Zur Übertragung der berechneten Spaltensummen werden den ausgelesenen Daten zwei weitere Zeilen angefügt. Dazu wird bei der Erstellung des *Skeleton*-Objektes der Parameter `trailing_lines=2` gesetzt (Zeile 21). Für jede zusätzliche Zeile wird die

Slot-Routine `simd_user_read_trailing()` (Zeilen 34 bis 38) aufgerufen, die beide Bytes der Spaltensumme nacheinander ausgibt.

Die verwendete Implementierung [Rei15] des FAST-Operators im digitalen SIMD-Array ist vom Typ FAST-7-12. Der in jeder Spalte in r_1 abgelegte Eingabewert wird zunächst an einen Spaltenvektor der Gestalt 5×1 der letzten 5 Eingabezeilen angefügt. Daraufhin wird für jede Spalte k das Pixel p_{center} im Mittelpunkt des Bresenham-Kreises, mit den $u = 12$ Pixeln p_i entlang der Kreisbahn im Uhrzeigersinn verglichen, wobei jeweils auch auf die nächste und übernächste Nachbarspalte zugegriffen wird. Wird ein zusammenhängendes Segment der Mindestlänge von $l = 7$ Pixeln gefunden, so wird der Punkt als Kandidatenpunkt markiert. Gleichzeitig erfolgt für alle Pixel zudem die Berechnung eines Maßes der „Eckigkeit“ mit

$$v = \min_{1 \leq i \leq u} \left(p_{\text{center}} - p_i \right).$$

Dieses dient schließlich zur Nicht-Maxima-Unterdrückung, indem ein Kandidatenpunkt genau dann als FAST-Punkt erkannt wird, wenn auch das Maß der Eckigkeit innerhalb einer 3×3 Nachbarschaft maximal ist, d.h. ein lokales Maximum gefunden wurde. Die Implementierung erfordert ca. 690 Takte, was bei 1024 Spalten 0,67 $^{\text{Takte}}/_{\text{Pixel}}$ entspricht. In Abb. 6-4f ist das resultierende Ausgabebild dargestellt, wobei die gefundenen FAST-Punkte weiß hervorgehoben wurden. Abb. 6-4g zeigt zudem die VSoC-basierte FAST-Berechnung ohne vorhergehende Anwendung der Median-Operation.

6.2.6 Bewertung

Durch die Extraktion der FAST-Punkte durch das VSoC wird zum Schutz der Privatsphäre auf die Ausgabe von Realbildern verzichtet. Defekte Pixel innerhalb homogener Regionen, die sich hinsichtlich ihres Grauwertes signifikant von ihrer Umgebung unterscheiden, werden abhängig vom gewählten Schwellwert immer als FAST-Punkte erkannt. Daher ist deren Korrektur noch vor der Merkmalsextraktion erforderlich. Durch die Anwendung der Median-Operation werden jedoch nicht nur defekte Pixel entfernt, sondern das Bild allgemein geglättet (vgl. Abb. 6-4b und Abb. 6-4c). Sowohl bei der OpenCV- als auch in der VSoC-Implementierung werden mit Median-Filterung deutlich weniger Punkte als FAST-Punkte erkannt. Durch die Berechnung der spaltenweisen Summen während der Ausgabe kann die Regelung der Belichtungszeit in beiden Fällen in Python implementiert werden.

Während zum Auffinden der Kandidatenpunkte in OpenCV- und VSoC-Implementierung der gleiche Algorithmus realisiert wurde, unterscheidet sich das Verfahren zur Nicht-Maxima-Unterdrückung hinsichtlich des verwendeten Maßes. Durch die OpenCV-Implementierung werden daher mehr Punkte als FAST-Punkte erkannt. Durch Abschaltung der Nicht-Maxima-Unterdrückung und Vergleich der Koordinaten der ausgegebenen Kandidaten-Punkte wurde die Korrektheit des Algorithmus nachgewiesen.

Da die Pixel am Rand der Sensormatrix des VSoC Teststrukturen enthalten bzw. abgedunkelt wurden, wurde die ROI gegenüber der vollen Auflösung von 1024×1024 Pixeln

Abbildung 6-7: Foto des Testaufbaus zur optischen Tonabnahme von einer E-Gitarre.

auf lediglich 960 × 960 Pixel beschränkt. Bei der Übertragung vollständiger Bilder entspricht dies 900 KiB/Bild. Durch die Berechnung der FAST-Punkte direkt auf dem VSoC muss lediglich deren Position ausgegeben werden. Für x– und y-Koordinate sind somit jeweils mind. 10 Bit notwendig. Aufgrund der Ausrichtung an Zweierpotenzen wird jedoch von jeweils 16 Bit ausgegangen, was 4 Byte pro FAST-Punkt entspricht. Zusätzlich erfolgt die Ausgabe der spaltenweisen Summe in zwei separaten Zeilen, wofür 1920 Byte benötigt werden. Gegenüber 900 KiB müssten so für das in Abb. 6-4f angegebene Beispiel lediglich $5467 \times 4\,\text{Byte} + 1920\,\text{Byte} = 23{,}2\,\text{KiB}$ übertragen werden. Dies entspricht einer Kompression von ca. 39:1. Die aktuelle Implementierung des Ausgabepfades des VSoC ist jedoch auf das Auslesen zusammenhängender Bereiche beschränkt und eine selektive Ausgabe basierend auf spaltenweisen Zuständen ist nicht möglich. Trotz der im Vorfeld extrahierten FAST-Punkte ist daher die Ausgabe von Daten für das gesamte Bild erforderlich. Die Ausgabe erfolgt daher als Binärbild, wobei gefundene FAST-Punkte als weiße Pixel auf schwarzem Untergrund erscheinen. Der reale Kompressionsfaktor der Implementierung beträgt somit lediglich 8:1.

6.3 Optische Tonabnahme

6.3.1 Funktionsweise und algorithmischer Ansatz

Als weiteres Anwendungsbeispiel des VSoC wurde ein Aufbau zur optischen Tonabnahme an einer E-Gitarre realisiert (siehe Abb. 6-7). Die Position der sechs Saiten der Gitarre wird subpixel-genau und mit hoher Bildwiederholrate bestimmt und zum PC zur Ausgabe übertragen (siehe Abb. 6-8). Zur Erhöhung des Kontrastes wird dabei „Durchlicht“ eingesetzt, weshalb der Untergrund hell und die Saiten dunkel erscheinen.

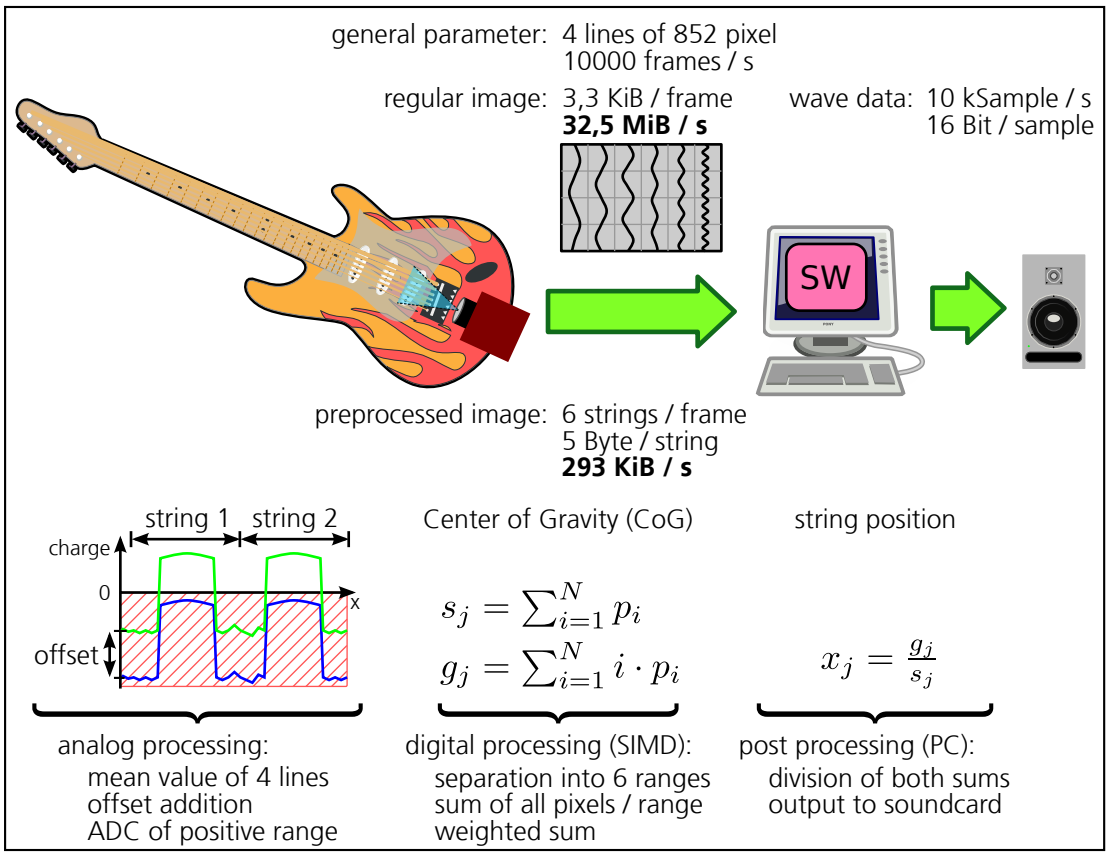


Abbildung 6-8: Darstellung des Funktionsprinzips der optischen Tonabnahme von einer E-Gitarre.

Die Umsetzung basiert ebenfalls auf dem Auslese-Modus „*Global Shutter*“, allerdings wird ein Bildausschnitt von lediglich vier aufeinander folgenden Zeilen betrachtet. Der Grauwert der Pixel wird zunächst in den SI-Zellen gespeichert, wodurch sich eine Umkehrung des Vorzeichens (siehe 3.1.3), d.h. ein negativer Wert ergibt (blaue Kurve in Abb. 6-8). Die Bildung des Mittelwertes erfolgt durch die Ausgabe der gespeicherten Werte mit einem entsprechenden Faltungskern für alle vier Zeilen gleichzeitig.

Durch die Ausgabe einer Ladung als Offset durch die sonst zur AD-Wandlung eingesetzten Stromquellen werden betragsmäßig kleine Werte in den positiven Bereich verschoben (grüne Kurve in Abb. 6-8). Während der helle Untergrund somit sein negatives Vorzeichen behält, gelangen die dunkel erscheinenden Saiten in den positiven Bereich. Zur Reduktion des Rauschens werden bei der anschließenden AD-Wandlung nur die Spalten mit positivem Vorzeichen betrachtet, wodurch nur innerhalb des Bereichs einer Saite ein gültiger Digitalwert vorliegt.

Für die weitere Verarbeitung im SIMD-Array wird die ausgelesene Zeile zunächst in sechs gleichgroße Bereiche der Breite N unterteilt, innerhalb derer jeweils eine Saite abgebildet wird. Für jeden Bereich j, wird sowohl die Summe aller Grauwerte $s_j = \sum_{i=1}^{N} p_i$,

als auch die gewichtete Summe der Grauwerte $g_j = \sum_{i=1}^{N} i \cdot p_i$ berechnet. Beginnend bei der jeweils letzten Spalte eines Bereichs, geschieht dies schrittweise für alle Bereiche parallel. Dabei erfolgt eine fortlaufende Addition des gewichteten und ungewichteten Grauwertes der jeweils aktuellen Spalte, zu den im vorhergehenden Schritt berechneten Zwischensummen. Anschließend werden die Zwischenergebnisse jeweils um eine Spalte weitergeschoben. Nach N Schritten ist die Berechnung beider Summen abgeschlossen, wobei für s_j 2 Byte genügen, für g_j jedoch 3 Byte notwendig sind. Diese werden in den ersten fünf Spalten eines jeden Bereichs abgelegt und schließlich ausgegeben.

Die tatsächliche Berechnung des Schwerpunktes („*Center of Gravity*“) der Saite j, erfolgt schließlich mit $x_j = \frac{g_j}{s_j}$ auf dem PC. Da die Ruheposition der Saite bedingt durch das Zupfen oder Dämpfen jedoch nicht konstant ist, wird zudem ein gleitender Mittelwert mit $\overline{x}_{j,k+1} = (1 - \alpha) \cdot \overline{x}_{j,k} + \alpha \cdot x_j$ für jeden Zeitschritt k nachgeführt. Als Basis zur Berechnung der Samples wird schließlich die Differenz $d_j = \overline{x}_{j,k} - x_j$ eingesetzt und zur Ausgabe an die Sound–Karte entsprechend skaliert.

6.3.2 Performance-Betrachtung

Die Größe des Beobachtungsbereichs einer Saite wurde mit $N = 142$ Pixel festgelegt. Zur Übertragung vollständiger Bilder mit einer Auflösung von entsprechend 4×852 Pixeln, sind bei 8 Bit AD-Wandlung, jeweils 3,3 KiB/Sample notwendig. Bei einer Abtastrate von 10000 Bilder/s, ergibt sich eine erforderliche Bandbreite von 32,5 MiB/s. Durch die Berechnung der Summen s_j und g_j auf dem VSoC sind jedoch insgesamt nur 30 Byte/Sample, d.h. 293 KiB/s erforderlich, was einem Kompressionsfaktor von ca. 114:1 entspricht.

Wie bereits in 6.2.6 beschrieben, erlaubt die aktuelle Implementierung des Ausgabepfades des VSoC ausschließlich zusammenhängende Bereiche. Die Begrenzung der Ausgabe auf 5 Byte pro Bereich ist daher nicht möglich. Stattdessen werden die berechneten Summen in die Grauwert-Bilder am Anfang des jeweiligen Bereichs eingeblendet. Die tatsächlichen Bilddaten stehen somit zur Visualisierung zur Verfügung. Alternativ könnten die Ergebnisse von bis zu $\lfloor \frac{142}{5} \rfloor = 28$ aufeinander folgenden Abtastwerte in einer gemeinsamen Ergebniszeile ausgegeben und somit auf die Übertragung von Grauwerten verzichtet werden. Die Position der Ergebnisse im jeweiligen Bereich einer Saite, muss dazu um jeweils 5 Spalten weiter verschoben werden. Durch die dafür erforderlichen zusätzlichen Operationen, nimmt jedoch die maximale Verarbeitungsgeschwindigkeit ab.

Bei der aktuellen Implementierung wird SIMDCTRL (siehe 3.1.1) mit einer Frequenz von 100 MHz getaktet, wodurch sich ein maximaler Durchsatz von ca. 15000 Samples/s ergibt. Die Ausgaberate wurde jedoch fest auf 10000 Samples/s gesetzt, um eine exakte Abstimmung mit der PC-basierten Nachverarbeitung zu erreichen.

6.4 Laser-Triangulation

6.4.1 Beschreibung des Verfahrens

Laser-Triangulation bzw. „Laserlichtschnitt“ („*Sheet-of-Light*“) [VM10] ist ein Verfahren zur Vermessung der Oberflächengeometrie eines Objektes und gehört zur Gruppe der

„*Structured Light*“-Verfahren. Mittels eines Lasers wird eine Linie auf die Oberfläche des zu untersuchenden Objektes projiziert, deren Abbildung mit einer Kamera beobachtet wird. Durch den bekannten Winkel zwischen Laser und Kamera kann anhand der Verformung der Linie auf die Höheninformation geschlossen werden. Zur tatsächlichen Rekonstruktion der Oberfläche sind mehrere derartige Höhenprofile erforderlich, weshalb das Objekt relativ zu Laser und Kamera verschoben wird. Dabei werden mit äquidistantem Abstand weitere Profile aufgenommen. Die Genauigkeit des Verfahrens hängt maßgeblich vom Winkel zwischen Laser und Kamera ab, d.h. vom Abbildungsverhältnis der Oberflächenhöhe zur korrespondierenden Position der projizierten Linie. Neben dem Kontrast der Oberfläche sowie verschiedenen Rauscheinflüssen (z.B. Photonenrauschen, Rauschen des Auslesepfades, Quantisierungsrauschen, ...) wirken die durch Reflexion des kohärenten Laserlichts an rauen Oberflächen verursachten Interferenzen („*Speckle*“) als dominierender, begrenzender Faktor. Die aus den mit der Kamera aufgenommenen Bildern extrahierte Information beschränkt sich jeweils auf die exakte Position der Abbildung der Laser-Linie. Diese Kompression kann mit Hilfe eines *Vision Chips* bereits im Sensor erfolgen. Anstelle der Ausgabe und Übertragung realer Bilddaten erfolgt somit eine frühzeitige Extraktion der Höhenprofile bereits im Sensor.

In [Rei16] wurden drei verschiedene Verfahren zum Auffinden der Linien-Position im VSoC implementiert und hinsichtlich der erreichbaren Genauigkeit und Geschwindigkeit miteinander verglichen. Während bei der „Maximum-Suche“ in jeder Spalte nach der Zeile mit maximaler Intensität gesucht und die korrespondierende Position zurückgeliefert wird, erfolgt bei der Schwerpunktsuche („*Center-of-Gravity*“, CoG) eine spaltenweise, subpixel-genaue Bestimmung der Position der Laserlinie. Für beide Verfahren wurde zudem eine Gauß-Filterung während der Ausgabe der Grauwerte auf die mit der Spaltenleitung verbundenen Kapazitäten angewendet und eine vollständige AD-Wandlung zur weiteren Verarbeitung in digitaler Domäne durchgeführt. Zur Vermeidung der zeitintensiven AD-Wandlung erfolgt beim dritten Verfahren – dem Ableitungsverfahren – die Kantendetektion basierend auf der ersten Ableitung des geglätteten Bildes entlang der Spalte, bei gleichzeitiger Anwendung einer Gauß-Filterung. An der Position des Maximums der Laserlinie befindet sich nach der ersten Ableitung somit ein Nulldurchgang. Zur Vermeidung falsch erkannter Positionen, bedingt durch Rauscheinflüsse, sucht das Verfahren in jeder Spalte nach der Position des ersten Nulldurchgangs nach dem einmaligen Überschreiten eines Schwellwertes.

6.4.2 Implementierung als Vision Task

Als Basis zur Implementierung der verschiedenen Algorithmen zum Auffinden der Laserlinie wurde der Auslesemodus „*Rolling Shutter*“ (siehe 6.1.2) verwendet. Im Folgenden soll beispielhaft auf die Implementierung des Ableitungsverfahrens eingegangen werden.

Wie die anderen Verfahren auch, erfordert das Ableitungsverfahren das Durchsuchen des Bildes. Anstelle der direkten Ausgabe des aktuellen Grauwertes jeder Zeile soll jedoch die Faltung mit einem Gauß-Kern sowie eine anschließende Differenzierung entlang

der Ausleserichtung erfolgen. Bedingt durch die Ableitungsregel der Faltungsoperation[1], können beide Operationen zusammengefasst werden, wodurch lediglich eine Faltung mit der ersten Ableitung des Gauß-Filters erforderlich ist.

Wie in 3.1.3 beschrieben, wird die diskrete Faltung durch Variation der Integrationszeiten bei der Ausgabe des Pixels auf die mit der Spaltenleitung verbundene Kapazität erreicht. Während positive Koeffizienten durch direkte Ausgabe des Wertes des Foto-FETs erzielt werden, erfolgt für negative Koeffizienten die Ausgabe des zuvor in der SI-Zelle abgelegten Wertes.

Die aus der ersten Ableitung der diskreten Gauß-Kurve resultierenden Vektoren $\underline{t_{\mathrm{Pix}}}$ und $\underline{t_{\mathrm{SI}}}$ (vgl. 3.1.3) werden jeweils zu Beginn des Ausleseprozesses eines Bildes in die oberste Position der Zeilensteuerung geladen. Der vom Auslesemodus vorgegebene, zeilenweise Ablauf veranlasst zunächst LINECTRL zur Ausgabe von Foto-FET und SI-Zelle an der aktuellen Position des Faltungskerns. Nach der Vorbereitung der nächsten Zeile, d.h. der Verschiebung von $\underline{t_{\mathrm{Pix}}}$ und $\underline{t_{\mathrm{SI}}}$ um eine Zeile nach unten, erfolgt die Benachrichtigung von SIMDCTRL. Durch SIMDCTRL wird nun nach der Position des ersten Nulldurchgangs jeder Spalte gesucht und diese gespeichert. Wurden alle zu betrachtenden Zeilen verarbeitet, so erfolgt schließlich die Ausgabe der Position des gefundenen Nulldurchgangs.

Im Nachverarbeitungsteil des *Vision Tasks* werden die vom VSoC berechneten Profile entgegengenommen und die Oberfläche des beobachteten Objektes unter Verwendung der bekannten Geometrie des Aufbaus berechnet und visualisiert.

6.4.3 Performance-Betrachtung

Für die verwendete Implementierung zur Erkennung der Position der Laserlinie werden 250 Zeilen mit je 1024 Pixeln betrachtet. Bei der Ausgabe vollständiger Bilder mit einer Auflösung von 8 Bit sind somit $250\,\mathrm{KiB/Bild}$ erforderlich. Durch die Berechnung der Profile im VSoC wird hingegen nur eine Zeile pro Bild, d.h. $1\,\mathrm{KiB/Bild}$ ausgegeben, was einem Kompressionsfaktor von 250:1 entspricht. Im Gegensatz zur optischen Tonabnahme (siehe 6.3.2) bzw. der Extraktion interessanter Punkte (siehe 6.2.6) spielt die Limitierung des Ausgabepfades bzgl. zusammenhängender Bereiche keine Rolle.

Die erreichbare Geschwindigkeit in Profilen pro Sekunde wird maßgeblich durch das verwendete *Skeleton* zur Umsetzung des Auslesemodus „*Rolling Shutter*“ begrenzt. Zur Verarbeitung einer Zeile bei einem Takt von 100 MHz, beträgt die erforderliche Zeit $T_{\mathrm{Zeile}} = 1{,}45\,\mu\mathrm{s}$ [Rei16], was ca. $2758\,\mathrm{Profile/s}$ entspricht. Die für ein Profil erforderliche Zeit wurde mit $T_{\mathrm{Bild}} = 0{,}733\,\mathrm{ms}$ bestimmt und liegt damit deutlich über den zunächst vermuteten $T_{\mathrm{Bild}} = 250 \cdot T_{\mathrm{Zeile}} = 0{,}362\,\mathrm{ms}$. Dabei wirken der für die von GLOBALCTRL initiierten Funktionsaufrufe erforderliche Mehraufwand sowie der generell sequentielle Ablauf der Verarbeitung als limitierende Faktoren. Zudem wurden wesentliche Teile des *Skeletons* in Python implementiert, der resultierende, generierte Code jedoch nicht optimiert. Da der Auslesemodus darüber hinaus keine überlappende Belichtungszeit un-

[1] Seien f und g Funktionen, D der Differenzierungs- und $*$ der Faltungsoperator, so besagt die Ableitungsregel der Faltungsoperation: $\mathrm{D}\,(f * g) = (\mathrm{D}f) * g = f * (\mathrm{D}g)$

terstützt, kommt diese zur Verarbeitungszeit eines Bildes jeweils hinzu. Die tatsächlich gemessene Profilrate bei einer Belichtungszeit von 150 µs wird schließlich mit 1080 Profile/s angegeben, was einer Datenrate von 1,1 MiB/s entspricht.

Da die Vorbereitung der nächsten Zeile durch LINECTRL überlappend zur Suche nach Nulldurchgängen in SIMDCTRL erfolgen kann, können beide ASIPs parallel arbeiten. Zudem erfolgt die Ausgabe des Profils erst nach der Durchsuchung des gesamten Bildes, weshalb eine Beteiligung von GLOBALCTRL erst nach dem Ende der Auswertung erforderlich ist. Zur Beschleunigung wurde daher ein spezieller Auslesemodus basierend auf „*Rolling Shutter*“ umgesetzt, bei dem die Bildaufnahme- bzw. -verarbeitungsfunktionen in LINECTRL und SIMDCTRL nur einmal pro Bild aufgerufen werden und selbstständig über die gewählten 250 Zeilen iterieren. Durch Verkürzung der vom Impulsgenerator vorgegebenen Integrationszeit sind bei einer für SIMDCTRL und LINECTRL gleichen Taktfrequenz von 100 MHz zur Verarbeitung einer Zeile ca. 42 Takte, d.h. 420 ns, erforderlich. Pro Bild entspricht dies somit 10500 Takten, wodurch sich ein theoretischer Maximaldurchsatz von 9524 Profile/s, bzw. 9,3 MiB/s ergibt. Der tatsächlich gemessene Durchsatz liegt mit ca. 9000 Profile/s, bzw. 8,8 MiB/s nur leicht darunter, was hauptsächlich durch die erforderliche Ausgabe sowie die Vorbereitung des nächsten Bildes begründet ist.

6.5 Ergebnisse

Das Multi-ASIP-basierte Steuerwerk ermöglicht die Ansteuerung und Verwendung der durch die Funktionseinheiten des VSoC in analoger und digitaler Domäne bereitgestellten Operationen und somit den Einsatz in Bildverarbeitungsaufgaben. Durch spezielle Instruktionen in den Befehlssätzen der ASIPs wird eine sehr hohe Flexibilität und Produktivität hinsichtlich der Umsetzung unterschiedlicher Algorithmen erreicht. Die Unterstützung mehrerer, paralleler Kontrollflüsse und deren Synchronisation erlaubt zudem die Ausnutzung der Parallelität verschiedener Funktionseinheiten.

Die Programmierung wird durch die Nutzung der Hochsprache Python deutlich vereinfacht, wobei eine ganzheitliche Betrachtung von VSoC-basierter Bildaufnahme und Vorverarbeitung sowie konventioneller Nachverarbeitung in Form von *Vision Tasks* erfolgt. Die automatische Transformation einzelner Programmabschnitte eines *Vision Tasks* in auf dem VSoC direkt ausführbaren Code ist insbesondere zur Umsetzung des Kontrollflusses sowie für erste, prototypische Implementierungen sinnvoll. Aufgrund der weitgehend fehlenden Optimierung während der Übersetzung der Python Bytecodes kann mit direkt in Assembler implementierten Abschnitten eine deutlich höhere Verarbeitungsgeschwindigkeit erzielt werden.

Bei der Entwicklung eines *Vision Tasks* kann zunächst mit konventioneller Bildverarbeitung gestartet werden, wobei schließlich ein schrittweiser Übergang zur Nutzung der VSoC-integrierten Verarbeitung erfolgt. Als Basis stehen hierfür spezielle *Skeletons* bereit, die verschiedene Auslese-Modi implementieren und die Synchronisation der einzelnen ASIPs vom Anwender verbergen. Sowohl eigene als auch durch Bibliotheken bereitgestellte Funktionen können in einzelnen *Slots* der *Skeletons* registriert und somit in den Bildaufnahme- bzw. Verarbeitungsprozess integriert werden. Dem somit eingebrachten

Komfort hinsichtlich der Vereinfachung der Programmierung stehen jedoch hohe Kosten durch den Mehraufwand für ASIP-übergreifende Funktionsaufrufe gegenüber. Durch die Realisierung anwendungsspezifischer *Skeletons* mit Hilfe miteinander interagierender Zustandsautomaten und den Verzicht auf eine zentrale Steuerung der Abläufe können bei Bedarf dennoch sehr hohe Verarbeitungsgeschwindigkeiten erreicht werden.

Die Nutzung des Simulators während der Entwicklung erlaubt die Untersuchung und Optimierung der implementierten Algorithmen unter reproduzierbaren Bedingungen sowie die Realisierung automatisierter Tests. So können Szenen jederzeit wiederholt und Rauscheinflüsse vollständig ausgeblendet werden. Der Einfluss von Nicht-Idealitäten und die Auswirkung von Filter- und Korrekturoperationen können direkt untersucht werden. Zudem ist eine Speicherung des Zustands des VSoC an beliebiger Stelle der Programmflüsse sowie die Ermittlung von Laufzeiteigenschaften möglich. Durch die Verwendung einheitlicher Schnittstellen wird die nahtlose Integration des Simulationssystems in den Entwurfsfluss und dessen Einsatz komplementär zum tatsächlichen DS ermöglicht. Der Programmcode eines implementierten *Vision Tasks* kann sowohl auf dem PC unter Einsatz des Simulators bzw. der Remote-Ansteuerung des DS als auch auf der in das DS integrierten CPU selbst ausgeführt werden. Letzteres erlaubt den autonomen Betrieb der Kamera- und Entwicklungsplattform als Sensorsystem, z.B. zum Einsatz in industriellen Anwendungen.

7 Zusammenfassung und Ausblick

Zunächst wurde der Stand der Technik bzgl. in der Literatur beschriebener *Vision Chips* dargestellt und diese hinsichtlich verschiedener Kriterien klassifiziert. Dabei wurden auch die zugehörigen Entwicklungsplattformen betrachtet. Es wurde deutlich, dass die meisten *Vision Chips* lediglich zur Demonstration des jeweiligen Verarbeitungsprinzips entwickelt wurden, jedoch bis auf wenige Ausnahmen nicht in realen Anwendungen eingesetzt werden. Zudem existieren zur Simulation von *Vision Chips* – im Gegensatz zur Simulation gewöhnlicher Bildsensoren – nur sehr wenige Arbeiten. Die gezielte Injektion von Fehlern und Nicht-Idealitäten, der im Zusammenhang mit frühzeitiger Merkmalsextraktion zur Validierung von Bildverarbeitungsalgorithmen eine besondere Bedeutung zukommt, wird dabei von keinem der Systeme unterstützt.

Nach der Betrachtung verschiedener Steuerwerksarchitekturen hinsichtlich der Art der Realisierung sowie möglicher Integrationsvarianten wurde ein integriertes Multi-ASIP-basiertes Steuerwerk und dessen Umsetzung im betrachteten VSoC vorgestellt. Der zur Ansteuerung der Funktionseinheiten erforderliche Decoder wird direkt in den Befehlssatz eines Prozessors integriert, wodurch eine enge Kopplung erreicht wird. Das Verhalten der durch die Funktionseinheiten bereitgestellten Operationen wird durch spezielle, applikationsspezifische Instruktionen abstrahiert. Als Basis dient ein Stack-basierter Prozessorkern, der eine einfache Schnittstelle zur Erweiterung des Befehlssatzes besitzt. Zudem werden Methoden zur Synchronisation und Kommunikation zwischen den verschiedenen ASIPs des Steuerwerks bereitgestellt. Die Zuordnung unabhängiger Funktionseinheiten zu verschiedenen, parallel arbeitenden ASIPs ermöglicht deren überlappende Arbeitsweise bzw. ein „*Pipelining*“ von analoger Bildaufnahme und -verarbeitung, digitalen Berechnungen und der Ausgabe der Ergebnisse. Durch den Verzicht auf fest vorgegebene Abläufe wird die Flexibilität der Architektur maximiert.

Zur Bereitstellung einer geeigneten Simulationsumgebung wurde ein strukturtreues Modell des VSoC entwickelt. Wesentliche Teile, darunter die digitale Steuerung sowie das Modell der Testumgebung, wurden unter Verwendung von SystemC realisiert. Der integrierte *Instruction Set Simulator* stellt dabei die Binärkompatibilität zum tatsächlichen VSoC her. Zur Umsetzung der besonders rechenintensiven Modelle der Pixel-Matrix und der zu beobachteten Szene wurde auf eine strukturtreue Realisierung verzichtet. Stattdessen erfolgte die Modellierung auf Verhaltensebene. Durch das gezielte Einbringen von Fehlern und Nicht-Idealitäten sowie die Berücksichtigung zeitlicher Effekte durch das Szenen-Modell wird die gemeinsame Betrachtung von Bildaufnahme- und -verarbeitung sowie die Untersuchung des Verhaltens entsprechender Algorithmen ermöglicht.

Für den Einsatz des VSoC in realen Bildverarbeitungsaufgaben wurde eine Kamera- und Entwicklungsplattform vorgestellt, die sowohl einen autonomen Betrieb als eigenständiges Sensorsystem ermöglicht, als auch für den PC-basierten Entwicklungsprozess

geeignet ist. Deren Basis bildet eine Kommunikationsinfrastruktur, die die einfache Ansteuerung eigener, im FPGA realisierter Komponenten durch Anwendungssoftware sowohl durch eine in die Plattform integrierte CPU als auch transparent über das Netzwerk erlaubt. Der Einsatz des Simulationssystems kann zudem komplementär zur Verwendung der tatsächlichen Entwicklungsplattform erfolgen, da durch einheitliche Schnittstellen eine nahtlose Integration in den Entwurfsfluss bereitsteht.

Die Programmierung des VSoC erfolgt primär durch eine eigene Assemblersprache, wird jedoch durch die Nutzung der Hochsprache Python deutlich vereinfacht. Dabei erfolgt eine ganzheitliche Betrachtung von VSoC-basierter Bildaufnahme und Vorverarbeitung sowie konventioneller Nachverarbeitung in Form von *Vision Tasks*. Einzelne, den ASIPs des Steuerwerks zugeordnete Programmabschnitte werden automatisch in direkt auf dem VSoC ausführbaren Code transformiert. Zur Entwicklung von *Vision Tasks* wird zunächst unter Verwendung konventioneller Bildverarbeitung begonnen und schrittweise zur Nutzung der VSoC-integrierten Verarbeitung übergegangen. Die komplexen Anforderungen bzgl. der Synchronisation und Kommunikation der einzelnen ASIPs untereinander, werden mit Hilfe vordefinierter *Skeletons* vom Benutzer verborgen. Die Adaption an individuelle Anforderungen der Bildaufnahme- und -verarbeitung erfolgt schließlich durch die Registrierung entsprechender Funktionen an von den *Skeletons* bereitgestellten *Slots*.

Aufbauend auf der grundlegenden Analyse der zum Einsatz von *Vision Chips* erforderlichen Umgebung wird durch die in dieser Arbeit beschriebene Methodik und ihre Umsetzung in Hard- und Softwarekomponenten das notwendige Bindeglied zwischen den durch die Funktionseinheiten bereitgestellten Verarbeitungsoperationen und deren Einsatz in tatsächlichen Anwendungen geschaffen. Sowohl das prinzipielle Vorgehen als auch die Funktionsweise der Methoden und Werkzeuge wurden durch verschiedene Anwendungsbeispiele gezeigt. Bei der Extraktion interessanter Punkte als Basis zur Präsenzdetektion steht der Schutz der Privatsphäre und der damit einhergehende Verzicht auf die Ausgabe realer Bilder im Vordergrund. Hingegen wird sowohl bei der optischen Tonabnahme von einer E-Gitarre als auch bei der Umsetzung der Laser-Triangulation zur 3D Oberflächenrekonstruktion von einer hohen Verarbeitungsgeschwindigkeit und einer frühzeitigen Datenreduktion profitiert. Bei der Realisierung der Anwendungsbeispiele zeigten sich neben den Stärken auch deutlich die Schwächen der implementierten Werkzeuge bzw. des VSoC selbst. Insbesondere die fehlende Optimierung während der Transformation der Python Bytecodes sowie die Beschränkung auf zusammenhängende Bereiche bei der Ausgabe von Ergebnissen aus dem VSoC wirken für verschiedene Anwendungen limitierend.

Eine mögliche Weiterführung der Arbeit kann – neben der Beseitigung der genannten Schwachstellen – durch eine Erweiterung des Konzepts des *„Vision Tasks“* hinsichtlich der FPGA-basierten Verarbeitung erfolgen. Analog der gezeigten Methodik zur Zuordnung von Programmabschnitten zu einzelnen ASIPs könnte unter Verwendung von *MyHDL* [VJB+11] eine Synthese von FPGA-Komponenten angestrebt werden. Die Integration von *Cython* [BBC+11] in den Übersetzungsprozess der „*Vision Tasks*“ würde zudem die Verarbeitungsgeschwindigkeit der auf dem PC auszuführenden Programmabschnitte deutlich erhöhen. Diese würden zunächst in die Programmiersprache C konvertiert

und schließlich compiliert werden, wodurch die aufwändige Interpretation des Python-Programms entfällt. Durch die Erweiterung der Simulationsumgebung hinsichtlich einer Abschätzung bzgl. der Verlustleistung kann die angestrebte Partitionierung und Abbildung von Algorithmen auch hinsichtlich der Leistungsaufnahme optimiert werden. Auch die direkte Integration von *Vision Chips* in Standard-Bibliotheken zur Bildverarbeitung sowie die automatische Abbildung und Optimierung von Algorithmen können als zukünftige Ziele genannt werden. Insbesondere bei aktuellen technologischen Trends, wie z.B. selbstfahrenden Autos, kann zukünftig von den Möglichkeiten der *Vision Chips* bzgl. einer frühzeitigen Datenreduktion sowie der Minimierung der Latenz profitiert werden.

A Programm Listings

Listing 1: Vision Task mit OpenCV-Implementierung zur Extraktion interessanter Punkte mit dem FAST-Operator.

```
import cv2
import numpy as np
from libch2dev import skeletons
#...

########
#Settings Auto-Exposure
ae_roi_x_start = int(960 / 2 - 50)
ae_roi_x_end   = int(960 / 2 + 50)
ae_roi_y_start = int(960 / 2 - 50)
ae_roi_y_end   = int(960 / 2 + 50)
ae_target_min  = (80 - 5)
ae_target_max  = (80 + 5)
ae_init_expo   = 1000        #in micro-seconds

#Settings FAST
fast_threshold = 20

#Create and configure the skeleton object
skel = skeletons.SkelCam2GlobalShutter(trailing_lines = 0)
skel.add_roi(...)
#...

#Create simulation platform
plat = Platform_Sim("localhost")

#Initialize OpenCV displays and images
cv2.startWindowThread()
cv2.namedWindow("disp_orig")
cv2.namedWindow("disp_median")
cv2.namedWindow("disp_points")
cv2.namedWindow("disp_points_orig")

#Create object of fast detector
fast = cv2.FastFeatureDetector_create(threshold=fast_threshold,
                                      nonmaxSuppression=True,
                                      type=cv2.FastFeatureDetector_TYPE_7_12)

#Endless loop for image acquisition and processing
cur_expo_time = ae_init_expo
while True:
    #set new exposure time, issue trigger and get image
    plat.acq_ctrl.set_exposure_time(cur_expo_time)
    plat.acq_ctrl.sw_trigger()
    img_orig = skel.acquire_img()

    #perform median calculation
    img_median = cv2.medianBlur(img_orig)
```

```
49
50      #get mean value for selected ROI
51      roi_img  = img_orig[ae_roi_y_start:ae_roi_y_end+1, ae_roi_x_start:ae_roi_x_end+1]
52      roi_mean = roi_img.mean()
53
54      #recalculate exposure time
55      cur_expo_time = f_expo(cur_expo_time, roi_mean)
56
57      #apply FAST to median image
58      kplist = fast.detect(img_median)
59      print("Count of FAST points in median image:   {}".format(len(kplist)))
60      img_points = cv2.drawKeypoints(img_median, kplist, img_points, color=(255,0,0))
61
62      #apply FAST to original image
63      kplist2 = fast.detect(img_orig)
64      print("Count of FAST points in original image: {}".format(len(kplist2)))
65      img_points_orig = cv2.drawKeypoints(img_orig, kplist2, img_points_orig, color=(255,0,0))
66
67      #show images
68      cv2.imshow("disp_orig", img_orig)                 #original image
69      cv2.imshow("disp_median", img_median)             #median filtered image
70      cv2.imshow("disp_points", img_points)             #points on median image
71      cv2.imshow("disp_points_orig", img_points_orig)   #points on original image
```

Listing 2: Vision Task mit VSoC-Implementierung zur Extraktion interessanter Punkte mit dem FAST-Operator.

```
1   import cv2
2   import numpy as np
3   from libch2dev import skeletons
4   from chrispy.chrisp2.cores import simd
5   #...
6
7   ########
8   #Settings Auto-Exposure
9   ae_roi_x_start = int(960 / 2 - 50)
10  ae_roi_x_end   = int(960 / 2 + 50)
11  ae_roi_y_start = int(960 / 2 - 50)
12  ae_roi_y_end   = int(960 / 2 + 50)
13  ae_target_min  = (80 - 5)
14  ae_target_max  = (80 + 5)
15  ae_init_expo   = 1000        #in micro-seconds
16
17  #Settings FAST
18  fast_threshold = 20
19
20  #Create and configure the skeleton object and register slots
21  skel = skeletons.SkelCam2GlobalShutter(trailing_lines = 2)
22  skel.add_roi(...)
23
24  ########
25  #Create some routine and register them as Slots within the Skeleton
26  ##
27  @simd.macro()
28  def simd_user_init_frame():
29      '''Called once per frame, reset column sum registers here.'''
30      #...
31  skel.reg_slot("simd_user_init_frame", simd_user_init_frame)
32
```

```
##
@simd.macro()
def simd_user_read_trailing():
    '''Called for each trailing line, first output column sum byte[0], than byte[1].'''
    #...
skel.reg_slot("simd_user_read_trailing", simd_user_read_trailing)

##
@simd.assign()
def simd_postproc(row, roi_yofs):
    '''Called after AD conversion, result is available in r_1.'''
    #1) backup registers needed below and used by FAST (r_2, r_3, r_4)
    simd.api.inline_asm(...)

    #2) prepare column vector in r_1, r_2 and r_3
    simd.api.inline_asm(...)

    #3) execute median
    simd_median("r_1", "r_2", "r_3", "r_4")

    #4) perform 16 bit addition for column sums when we're inside ROI
    if (row >= ae_roi_y_start + roi_yofs) and (row <= ae_roi_y_end + roi_yofs):
        simd.api.inline_asm(...)

    #5) restore registers used by FAST
    simd.api.inline_asm(...)

    #6) perform FAST operation, result is available in r_1 again
    FAST(fast_threshold)
skel.reg_slot("simd_user_postproc", simd_postproc)

########
#Create simulation platform
plat = Platform_Sim("localhost")

#Endless loop for image acquisition and processing
cur_expo_time = ae_init_expo
while True:
    #set new exposure time, issue trigger and get image
    plat.acq_ctrl.set_exposure_time(cur_expo_time)
    plat.acq_ctrl.sw_trigger()

    #acquire image, separate FAST data and trailing lines
    img_raw      = skel.acquire_img()
    img_fast     = img_raw[0:-2, 0:]
    img_trailing = img_raw[-2:, 0:]

    #get mean value for selected ROI
    colsums  = (img_trailing[0, ae_roi_x_start:ae_roi_x_end+1].astype(dtype=np.int) +
               256.0 * img_trailing[1, ae_roi_x_start:ae_roi_x_end+1].astype(dtype=np.int))
    roi_mean = colsums.sum() / ((ae_roi_x_end - ae_roi_x_start + 1) *
                                (ae_roi_y_end - ae_roi_y_start + 1))

    #recalculate exposure time
    cur_expo_time = f_expo(cur_expo_time, roi_mean)

    #calculate histogram, get count of FAST points
    hist = cv2.calcHist([img_fast],[0],None,[256],[0,256])
    print("Count of FAST points: {}".format(hist[255][0]))
```

Literaturverzeichnis

[ABD+91] Anderson, S ; Bruce, WH ; Denyer, PB ; Renshaw, D ; Wang, G: A single chip sensor and image processor for fingerprint verification. In: *Custom Integrated Circuits Conference, 1991., Proceedings of the IEEE 1991* IEEE, 1991, S. 12–1

[ACC+02] Albani, L ; Chiesa, Pietro ; Covi, Daniele ; Pedegani, G ; Sartori, A ; Vatteroni, M: VISoc: a Smart Camera SoC. In: *Solid-State Circuits Conference, 2002. ESSCIRC 2002. Proceedings of the 28th European* IEEE, 2002, S. 367–370

[Ana08] AnaFocus: *Eye-RIS VSoC OEM Kit Brochure*. 2008

[Ana12] AnaFocus: *Datasheet for Q-Eye Image Sensor*. 2012

[Bai11] Bailey, Donald G.: *Design for Embedded Image Processing on FPGAs*. John Wiley & Sons, 2011

[BBC+11] Behnel, Stefan ; Bradshaw, Robert ; Citro, Craig ; Dalcin, Lisandro ; Seljebotn, Dag S. ; Smith, Kurt: Cython: The best of both worlds. In: *Computing in Science & Engineering* 13 (2011), Nr. 2, S. 31–39

[BBY03] Bjørnsen, J. ; Bonnerud, T.E. ; Ytterdal, T.: Behavioral Modeling and Simulation of Mixed-Signal System-on-a-Chip using SystemC. In: *Analog Integrated Circuits and Signal Processing* 34 (2003), Nr. 1, S. 25–38

[BCD12] Barr, David R. ; Carey, Stephen J. ; Dudek, Piotr: Low power multiple object tracking and counting using a SCAMP cellular processor array. In: *Cellular Nanoscale Networks and Their Applications (CNNA), 2012 13th International Workshop on* IEEE, 2012, S. 1–2

[BCGRV10] Bakkali, M. ; Carmona-Galán, R. ; Rodríguez-Vázquez, A.: A Prototype Node for Wireless Vision Sensor Network Applications Development. In: *I/V Communications and Mobile Network (ISVC), 2010 5th International Symposium on*, 2010, S. 1–4

[BCLD06] Barr, David R. ; Carey, Stephen J. ; Lopich, Alexey ; Dudek, Piotr: A Control System for a Cellular Processor Array. In: *Cellular Neural Networks and Their Applications, 2006. CNNA'06. 10th International Workshop on* IEEE, 2006, S. 1–6

[BD08] Barr, David R. ; Dudek, Piotr: A Cellular Processor Array Simulation and Hardware Prototyping Tool. In: *Cellular Neural Networks and Their Applications, 2008. CNNA 2008. 11th International Workshop on* IEEE, 2008, S. 213–218

[BD09] Barr, David R. ; Dudek, Piotr: APRON: A Cellular Processor Array Simulation and Hardware Design Tool. In: *EURASIP Journal on Advances in Signal Processing* (2009)

[BDP11] Blanchard, Y. ; Dupret, A. ; Peizerat, A.: SystemC Modelization for fast Validation of Imager Architectures. In: *Design and Architectures for Signal and Image Processing (DASIP), 2011 Conference on*, 2011, S. 1–5

[BK08] Bradski, Gary ; Kaehler, Adrian: *Learning OpenCV: Computer vision with the OpenCV library.* O'Reilly Media, Inc., 2008

[Bos04] Bossel, Hartmut: *Systeme, Dynamik, Simulation.* BoD–Books on Demand, 2004

[CBD11] Carey, Stephen J. ; Barr, David Robert W. ; Dudek, Piotr: Demonstration of a Low Power Image Processing System using a SCAMP3 Vision Chip. In: *ICDSC*, 2011, S. 1–2

[CBD13] Carey, Stephen J. ; Barr, David R. ; Dudek, Piotr: Low power high-performance smart camera system based on SCAMP vision sensor. In: *Journal of Systems Architecture* 59 (2013), Nr. 10, S. 889–899

[CBW+12] Carey, Stephen J. ; Barr, David R. ; Wang, Bin ; Lopich, Alexey ; Dudek, Piotr: Locating High Speed Multiple Objects using a SCAMP-5 Vision-Chip. In: *Cellular Nanoscale Networks and Their Applications (CNNA), 2012 13th International Workshop on* IEEE, 2012, S. 1–2

[Che03] Chen, T.: *Digital Camera System Simulator and Applications*, Stanford University, Diss., 2003

[CJGLC+05] Carranza, Luis ; Jimenez-Garrido, Francisco ; Linan-Cembrano, Gustavo ; Roca, Elisenda ; Meana, Servando E. ; Rodríguez-Vázquez, Angel: ACE16k based stand-alone system for real-time pre-processing tasks. In: *Microtechnologies for the New Millennium 2005* International Society for Optics and Photonics, 2005, S. 872–879

[CLB+13] Carey, S.J. ; Lopich, A. ; Barr, D.R.W. ; Wang, Bin ; Dudek, P.: A 100,000 fps Vision Sensor with Embedded 535GOPS/W 256x256 SIMD Processor Array. In: *VLSI Circuits (VLSIC), 2013 Symposium on*, 2013, S. C182–C183

[CRKH05] Corbet, Jonathan ; Rubini, Alessandro ; Kroah-Hartman, Greg: *Linux Device Drivers*. O'Reilly Media, Inc., 2005

[CS04] Costantini, Roberto ; Susstrunk, Sabine: Virtual Sensor Design. (2004), 408-419. http://dx.doi.org/10.1117/12.525704. – DOI 10.1117/12.525704

[CSS11a] Cenni, F. ; Scotti, S. ; Simeu, E.: Behavioral modeling of a CMOS video sensor platform using SystemC AMS/TLM. In: *Forum on Specification and Design Languages (FDL)* IEEE, 2011

[CSS11b] Cenni, Fabio ; Scotti, Serge ; Simeu, Emmanuel: SystemC AMS behavioral modeling of a CMOS video sensor. In: *19th International Conference on VLSI and System-on-Chip (VLSI-SoC)* IEEE, 2011, S. 380–385

[CSS11c] Cenni, Fabio ; Scotti, Serge ; Simeu, Emmanuel: A SystemC AMS/TLM platform for CMOS video sensors. In: *Conference on Design and Architectures for Signal and Image Processing (DASIP)* IEEE, 2011

[CV316] *OpenCV API Reference – OpenCV 3.1.0 Open Source Computer Vision*. http://docs.opencv.org/3.1.0, November 2016

[CVB+09] Chen, Junqing ; Venkataraman, K. ; Bakin, D. ; Rodricks, B. ; Gravelle, R. ; Rao, P. ; Ni, Yongshen: Digital Camera Imaging System Simulation. In: *Electron Devices, IEEE Transactions on* 56 (2009), nov., Nr. 11, S. 2496 –2505. http://dx.doi.org/10.1109/TED.2009.2030995. – DOI 10.1109/TED.2009.2030995. – ISSN 0018–9383

[DC98] Downton, A. ; Crookes, D.: Parallel architectures for image processing. In: *Electronics Communication Engineering Journal* 10 (1998), Jun, Nr. 3, S. 139–151. http://dx.doi.org/10.1049/ecej:19980307. – DOI 10.1049/ecej:19980307. – ISSN 0954–0695

[DCERV+97] Dominguez-Castro, Rafael ; Espejo, Servando ; Rodríguez-Vázquez, Angel ; Carmona, Ricardo ; Földesy, Péter ; Zarándy, Ákos ; Szolgay, Péter ; Szirányi, Tamás ; Roska, Tamaá u. a.: A 0.8-μm CMOS Two-Dimensional Programmable Mixed-Signal Focal-Plane Array Processor with On-Chip Binary Imaging and Instructions Storage. In: *Solid-State Circuits, IEEE Journal of* 32 (1997), Nr. 7, S. 1013–1026

[DH01] Dudek, P. ; Hicks, P.J.: An analogue SIMD Focal-Plane Processor Array. In: *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on* Bd. 4, 2001, S. 490–493 vol. 4

[DHRP15] Döge, J. ; Hoppe, C. ; Reichel, P. ; Peter, N.: A 1 Megapixel HDR Image Sensor SoC with Highly Parallel Mixed-Signal Processing. In: *International Image Sensor Workshop (IISW)* IEEE, 2015

[DKN02] Dupret, Antoine ; Klein, Jacques-Olivier ; Nshare, Abdallah: A DSP-like analogue processing unit for smart image sensors. In: *International Journal of Circuit Theory and Applications* 30 (2002), Nr. 6, S. 595–609

[Dög08] Döge, Jens: *Ladungsbasierte analog-digitale Signalverarbeitung für schnelle CMOS-Bildsensoren.* Dresden : TUDpress, 2008. – ISBN 9783940046727

[DSAS11] Demant, Christian ; Streicher-Abel, Bernd ; Springhoff, Axel: *Industrielle Bildverarbeitung: Wie optische Qualitätskontrolle wirklich funktioniert.* Springer-Verlag, 2011

[DTV+11] Dupret, Antoine ; Tchagaspanian, Michael ; Verdant, Arnaud ; Alacoque, Laurent ; Peizerat, Arnaud: Smart Imagers of the future. In: *DATE*, 2011, S. 437–442

[Dud05] Dudek, Piotr: Implementation of SIMD Vision Chip with 128× 128 array of Analogue Processing Elements. In: *International Symposium on Circuits and Systems (ISCAS)* IEEE, 2005, S. 5806–5809

[Dud11] Dudek, Piotr: SCAMP-3: A Vision Chip with SIMD Current-Mode Analogue Processor Array. In: *Focal-plane sensor-processor chips.* Springer, 2011, S. 17–43

[Dur14] Durini, Daniel: *High Performance Silicon Imaging: Fundamentals and Applications of CMOS and CCD sensors.* Elsevier, 2014

[EBD+06] Elouardi, A ; Bouaziz, S ; Dupret, A ; Lacassagne, L ; Klein, JO ; Reynaud, R: A smart sensor-based vision system: implementation and evaluation. In: *Journal of Physics D: Applied Physics* 39 (2006), Nr. 8, S. 1694

[FBCGCG+08] Fernández-Berni, J ; Carmona-Galán, R ; Carranza-González, L u. a.: A vision-based monitoring system for very early automatic detection of forest fires. (2008)

[FBCGCG+10] Fernández-Berni, Jorge ; Carmona-Galán, Ricardo ; Carranza González, Luis ; Cano-Rojas, Alberto ; Martínez-Carmona, Juan F. ; Rodríguez Vázquez, Angel ; Morillas, Sergio u. a.: On-site forest fire smoke detection by low-power autonomous vision sensor. (2010)

[FBCGCG11] Fernández-Berni, J. ; Carmona-Galán, R. ; Carranza-González, L.: FLIP-Q: A QCIF Resolution Focal-Plane Array for Low-Power Image Processing. In: *Solid-State Circuits, IEEE Journal of* 46 (2011), March, Nr. 3, S. 669–680. `http://dx.doi.org/10.1109/JSSC.2010.2102591`. – DOI 10.1109/JSSC.2010.2102591. – ISSN 0018–9200

[FBCGLC+11] Fernández-Berni, J. ; Carmona-Galán, R. ; Linan-Cembrano, G. ; Zarandy, A. ; Rodríguez-Vázquez, A.: Wi-FLIP: A Wireless Smart Camera Based on a Focal-plane Low-power Image Processor. In: *Distributed Smart Cameras (ICDSC), 2011 Fifth ACM/IEEE International Conference on*, 2011, S. 1–6

[FBCGR+14] Fernández-Berni, J. ; Carmona-Galán, R. ; Río, Rocío del ; Kleihorst, R. ; Philips, W. ; Rodríguez-Vázquez, Á.: Demo: A Prototype Vision Sensor for Real-time Focal-plane Obfuscation through Tunable Pixelation. In: *Proceedings of the International Conference on Distributed Smart Cameras*. New York, NY, USA : ACM, 2014 (ICDSC '14). – ISBN 978–1–4503–2925–5, 42:1–42:2

[FBGRRV14] Fernández-Berni, J. ; Galán, R. C. ; Río, R. del ; Rodríguez-Vázquez, Á.: A QVGA Vision Sensor with Multi-functional Pixels for Focal-Plane Programmable Obfuscation. In: *Proceedings of the International Conference on Distributed Smart Cameras*. New York, NY, USA : ACM, 2014 (ICDSC '14). – ISBN 978–1–4503–2925–5, 12:1–12:6

[FCW12] Farrell, J.E. ; Catrysse, P.B. ; Wandell, B.A.: Digital camera simulation. In: *Applied Optics* 51 (2012), S. A80–A90

[FOP08] Farrell, J. ; Okincha, M. ; Parmara, M.: Sensor Calibration and Simulation. In: *Electronic Imaging* International Society for Optics and Photonics, 2008

[GDHP10] Ginhac, Dominique ; Dubois, Jérôme ; Heyrman, Barthélémy ; Paindavoine, Michel: A high speed programmable focal-plane SIMD vision chip. In: *Analog Integrated Circuits and Signal Processing* 65 (2010), Nr. 3, S. 389–398

[Geo16] George, Damien: *MicroPython Python for microcontrollers*. `https://micropython.org/`. Version: November 2016

[GMS07] Galup-Montoro, C. ; Schneider, M.C.: *MOSFET Modeling for Circuit Analysis and Design*. World Scientific, 2007 (International series on advances in solid state electronics and technology). – ISBN 9789812707598

[Gra16] Graphviz project team: *Graphviz - Graph Visualization Software*. `http://www.graphviz.org/`. Version: November 2016

[GRF+07] Gow, R.D. ; Renshaw, D. ; Findlater, K. ; Grant, L. ; McLeod, S.J. ; Hart, J. ; Nicol, R.L.: A Comprehensive Tool for Modeling CMOS Image-Sensor-Noise Performance. In: *Electron Devices, IEEE Transactions on* 54 (2007), june, Nr. 6, S. 1321 –1329. `http://dx.doi.`

org/10.1109/TED.2007.896718. – DOI 10.1109/TED.2007.896718. – ISSN 0018–9383

[Hal07] Hallinan, Christopher: *Embedded Linux Primer: A practical, Real-World Approach*. Pearson Education India, 2007

[Har08] Harboe, Øyvind: *ZPU - The worlds smallest 32 bit CPU with GCC toolchain*. http://opencores.org/project,zpu, November 2008

[HML07] Hoffmann, A ; Meyr, H ; Leupers, R: Optimized ASIP Synthesis from Architecture Description Language Models. (2007)

[Hon01] Hong, Canaan S.: *On-chip Spatial Image Processing with CMOS Active Pixel Sensors*, University of Waterloo, Diss., 2001

[IEE12] IEEE Standards Association and others: *Standard SystemC language reference manual*. Bd. IEEE1666-2011. 2012

[IK01] Ishikawa, M. ; Komuro, T.: Digital Vision Chips and High-Speed Vision Systems. In: *VLSI Circuits, 2001. Digest of Technical Papers. 2001 Symposium on*, 2001, S. 1–4

[IL06] Ienne, Paolo ; Leupers, Rainer: *Customizable Embedded Processors: Design Technologies and Applications*. Academic Press, 2006

[KAK+04] Kolehmainen, Timo T. ; Aikio, Janne ; Karppinen, Mikko ; Mattila, Antti-Jussi ; Makinen, Jukka-Tapani ; Kataja, Kari ; Tukkiniemi, Kari ; Karioja, Pentti: Simulation of imaging system's performance. (2004), 204-212. http://dx.doi.org/10.1117/12.504330. – DOI 10.1117/12.504330

[KDJS87] Kirkish, SD ; Daly, JC ; Jou, L ; Su, Shing-Fong: Optical characteristics of CMOS-fabricated MOSFET's. In: *IEEE Journal of Solid-State Circuits* 22 (1987), S. 299–301

[KKI04a] Kagami, S. ; Komuro, T. ; Ishikawa, M.: A High-Speed Vision System with In-Pixel Programmable ADCs and PEs for Real-Time Visual Sensing. In: *Advanced Motion Control, 2004. AMC '04. The 8th IEEE International Workshop on*, 2004, S. 439–443

[KKI04b] Komuro, T. ; Kagami, S. ; Ishikawa, M.: A Dynamically Reconfigurable SIMD Processor for a Vision Chip. In: *Solid-State Circuits, IEEE Journal of* 39 (2004), Jan, Nr. 1, S. 265–268. http://dx.doi.org/10.1109/JSSC.2003.820876. – DOI 10.1109/JSSC.2003.820876. – ISSN 0018–9200

[KKII02] Kagami, S. ; Komuro, T. ; Ishii, I. ; Ishikawa, M.: A Real-Time Visual Processing System using a General-Purpose Vision Chip. In: *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on* Bd. 2, 2002, S. 1229–1234

[KKIK05] Komuro, T. ; Kagami, S. ; Ishikawa, M. ; Katayama, Y.: Development of a Bit-level Compiler for Massively Parallel Vision Chips. In: *Computer Architecture for Machine Perception, 2005. CAMP 2005. Proceedings. Seventh International Workshop on*, 2005, S. 204–209

[Koo89] Koopman, Philip: *Stack Computers: The new wave (reprint).* Mountain View Press, 1989

[LD13] Lopich, Alexey ; Dudek, Piotr: A General-purpose Vision Processor with 160× 80 Pixel-Parallel SIMD Processor Array. In: *Custom Integrated Circuits Conference (CICC), 2013 IEEE* IEEE, 2013, S. 1–4

[LDV+12] Laforest, T ; Dupret, A ; Verdant, A ; Lattard, D ; Villard, P: Algorithm Architecture Co-Design for Ultra Low-Power Image Sensor. In: *IS&T/SPIE Electronic Imaging* International Society for Optics and Photonics, 2012, S. 829806–829806

[LEDCRV02a] Linan, G ; Espejo, S ; Domínguez-Castro, R ; Rodríguez-Vázquez, Angel: ACE4k: An analog I/O 64× 64 visual microprocessor chip with 7-bit analog accuracy. In: *International Journal of Circuit Theory and Applications* 30 (2002), Nr. 2-3, S. 89–116

[LEDCRV02b] Linan, G ; Espejo, S ; Dominguez-Castro, R ; Rodríguez-Vázquez, Angel: Architectural and Basic Circuit Considerations for a Flexible 128× 128 Mixed-Signal SIMD Vision Chip. In: *Analog Integrated Circuits and Signal Processing* 33 (2002), Nr. 2, S. 179–190

[Liu08] Liu, Dake: *Embedded DSP Processor Design: Application Specific Instruction Set Processors.* Bd. 2. Morgan Kaufmann, 2008

[LMJM05] Lindgren, Leif ; Melander, Johan ; Johansson, Robert ; Moller, B: A Multiresolution 100-GOPS 4-Gpixels/s Programmable Smart Vision Sensor for Multisense Imaging. In: *Solid-State Circuits, IEEE Journal of* 40 (2005), Nr. 6, S. 1350–1359

[LPL10] Lahdenoja, Olli ; Poikonen, Jonne ; Laiho, Mika: Extracting Local Binary Patterns with MIPA4k Vision Processor. In: *Cellular Nanoscale Networks and Their Applications (CNNA), 2010 12th International Workshop on* IEEE, 2010, S. 1–5

[Lyn93] Lynch, Michel A.: *Microprogrammed State Machine Design.* CRC press, 1993

[Lyo81] Lyon, Richard F.: The Optical Mouse, and an Architectural Methodology for Smart Digital Sensors. In: *Proceedings of CMU Conference on VLSI Structures and Computations*, Computer Science Press, 1981

[Mea89] Mead, Carver: Adaptive retina. In: *Analog VLSI implementation of neural systems*. Springer, 1989, S. 239–246

[Moc05] Mochel, Patrick: The sysfs Filesystem. In: *Linux Symposium*, 2005, S. 313

[Moi97] Moini, A: Vision Chips or Seeing Silicon. 240 (1997). `http://www.ohio.edu/people/starzykj/network/class/ee715/labs/systems/vision_chips.pdf`

[NG00] Ni, Yang ; Guan, Jianhong: A 256/spl times/256 Pixel Smart CMOS Image Sensor for Line-Based Stereo Vision Applications. In: *Solid-State Circuits, IEEE Journal of* 35 (2000), Nr. 7, S. 1055–1061

[Nio09] Nios, II: *Processor Reference Handbook*. 2009

[NNTF79] Nudd, G. R. ; Nygaard, P. A. ; Thurmond, G. D. ; Fouse, S. D.: A Charge-Coupled Device Image Processor For Smart Sensor Applications. In: *Proc. SPIE* 0155 (1979), 15-22. `http://dx.doi.org/10.1117/12.956726`. – DOI 10.1117/12.956726

[Oht10] Ohta, Jun: *Smart CMOS Image Sensors and Applications*. CRC Press, 2010

[OSC10] OSCI Open SystemC Initiative and AMS Working Group and others: *Standard SystemC-AMS extensions language reference manual*. 2010

[Pet13] Peter, Nico: *Entwicklung eines effizienten generischen Schnittstellenkonzepts fÃŒr Bildsensoren*, Diplomarbeit, Januar 2013

[PHZA11] Pietikäinen, Matti ; Hadid, Abdenour ; Zhao, Guoying ; Ahonen, Timo: *Computer Vision Using Local Binary Patterns*. Springer, 2011

[PJ15] P., Cambou ; Jaffard, Jean-Luc: Status of the CMOS Image Sensor Industry. In: *Yole Développement* (2015)

[PLB12] Pears, N. ; Liu, Y. ; Bunting, P.: *3D Imaging, Analysis and Applications*. Springer London, 2012. – ISBN 978–1447140634

[PLP09] Poikonen, Jonne ; Laiho, Mika ; Paasio, Ari: MIPA4k: A 64× 64 Cell Mixed-mode Image Processor Array. In: *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on* IEEE, 2009, S. 1927–1930

[PW09] Pilgrim, Mark ; Willison, Simon: *Dive Into Python 3.* Springer, 2009

[Pyt16] Python Software Foundation: *Python website.* `https://www.python.org`. Version: November 2016

[QK11] Quade, Jürgen ; Kunst, Eva-Katharina: *Linux-Treiber Entwickeln: Eine systematische Einführung in die Gerätetreiber-und Kernelprogrammierung.* Dpunkt. verlag, 2011

[RD05] Rosten, Edward ; Drummond, Tom: Fusing Points and Lines for High Performance Tracking. In: *IEEE International Conference on Computer Vision* Bd. 2, 2005, 1508–1511

[RD06] Rosten, Edward ; Drummond, Tom: Machine learning for high-speed corner detection. In: *European Conference on Computer Vision* Bd. 1, 2006, 430–443

[RD14a] Reichel, Peter ; Döge, Jens: Hardware/Software Infrastructure for ASIC Commissioning and Rapid System Prototyping. In: *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on* IEEE, 2014

[RD14b] Reichel, Peter ; Döge, Jens: Prototyping-Plattform zur Inbetriebnahme integrierter Schaltkreise. In: Schneider, Peter (Hrsg.) ; Dittrich, Michael (Hrsg.): *Dresdner Arbeitstagung Schaltungs- und Systementwurf, DASS 2014. Tagungsband.*, 2014, S. 50–55

[RDH+16] Reichel, Peter ; Döge, Jens ; Hoppe, Christoph ; Peter, Nico ; Reichel, Andreas ; Schneider, Peter: Simulation platform for application development on a Vision-System-on-Chip with integrated signal processing. In: *Journal of Electronic Imaging* 25 (2016), Nr. 4, 041004. `http://dx.doi.org/10.1117/1.JEI.25.4.041004`. – DOI 10.1117/1.JEI.25.4.041004. ISBN 1017–9909

[RDP+16] Reichel, Peter ; Döge, Jens ; Peter, Nico ; Hoppe, Christoph ; Reichel, Andreas ; Schneider, Peter: Software Environment for Holistic Vision-System-on-Chip Programming. In: *Electronic Imaging* 2016 (2016), Nr. 12. – ISSN 2470–1173

[RDPH15] Reichel, Peter ; Döge, Jens ; Peter, Nico ; Hoppe, Christoph: An ASIP-based Control System for Vision Chips with Highly Parallel Signal Processing. In: *The 24th IEEE International Symposium on Industrial Electronics (ISIE)* IEEE, 2015

[Rei15] Reichel, Andreas: Motion-based 3-D Localisation in Natural Scenes Using a Novel Sensor-Processor-Architecture / Technische Universität Dresden. 2015. – Research Paper

[Rei16] Reichel, Andreas: *Analysing the Algorithmic Performance of the Analogue Image Pre-Processing of a VSoC for 1-D and 2-D Feature Extraction*, Technische Universität Dresden, Diploma Thesis, 2016

[RHDP15] Reichel, Peter ; Hoppe, Christoph ; Döge, Jens ; Peter, Nico: Simulation Environment for a Vision-System-on-Chip with Integrated Processing. In: *Proceedings of the International Conference on Distributed Smart Cameras (ICDSC)*, 2015

[RVDCJG+08] Rodríguez-Vázquez, Ángel ; Domínguez-Castro, Rafael ; Jiménez-Garrido, Francisco ; Morillas, Sergio ; Listán, Juan ; Alba, Luis ; Utrera, Cayetana ; Espejo, Servando ; Romay, Rafael: The Eye-RIS CMOS Vision System. In: *Analog circuit design*. Springer, 2008, S. 15–32

[RVDCJG+10] Rodríguez-Vázquez, Angel ; Domínguez-Castro, Rafael ; Jiménez-Garrido, Francisco ; Morillas, Sergio ; García, Alberto ; Utrera, Cayetana ; Pardo, Ma D. ; Listan, Juan ; Romay, Rafael: A CMOS Vision System On-Chip with Multi-Core, Cellular Sensory-Processing Front-End. In: *Cellular nanoscale sensory wave computing*. Springer, 2010, S. 129–146

[RVLCC+04] Rodríguez-Vázquez, Angel ; Liñán-Cembrano, Gustavo ; Carranza, L ; Roca-Moreno, Elisenda ; Carmona-Galán, Ricardo ; Jiménez-Garrido, Francisco ; Domínguez-Castro, Rafael ; Meana, S E.: ACE16k: The Third Generation of Mixed-Signal SIMD-CNN ACE Chips Toward VSoCs. In: *Circuits and Systems I: Regular Papers, IEEE Transactions on* 51 (2004), Nr. 5, S. 851–863

[RZZ+99] Roska, Tamas ; Zarandy, Akos ; Zold, Sandor ; Foldesy, Peter ; Szolgay, Peter: The Computational Infrastructure of Analogic CNN Computing – Part I: The CNN-UM Chip Prototyping System. In: *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on* 46 (1999), Nr. 2, S. 261–268

[Sch12] Schaumont, Patrick R.: *A Practical Introduction to Hardware/Software Codesign*. Springer, 2012

[Sem15] Semiconductor, ON: *AS0260 Datasheet 1/6-Inch 1080P High-Definition (HD) System-On- A-Chip (SOC) Digital Image Sensor*. May 2015

[SFRZ02] Szatmari, ISTVÁN ; Földesy, PÉTER ; Rekeczky, CSABA ; Zarándy, ÁKOS: Image processing library for the Aladdin Visual Computer. In: *Proceedings of the CNNA-2002 Frankfurt, Germany* (2002)

[Sie99] Siemers, Christian: *Prozessorbau: Eine konstruktive Einführung in das Hardware/Software-Interface*. Hanser, 1999

[Smi03] Smith, Kevin D.: PEP 0318 – Decorators for Functions and Methods. (2003), Juni. `https://www.python.org/dev/peps/pep-0318/`

[Ste08] Steinmüller, Johannes: *Bildanalyse: Von der Bildverarbeitung zur räumlichen Interpretation von Bildern*. Springer-Verlag, 2008

[Tan09] Tanenbaum, Andrew S.: *Modern Operating Systems*. Bd. 3. Pearson Prentice Hall, 2009

[Tei13] Teich, Jürgen: *Digitale Hardware/Software-Systeme: Synthese und Optimierung*. Springer-Verlag, 2013

[Vah10] Vahid, Frank: *Digital Design with RTL Design, Verilog and VHDL*. John Wiley & Sons, 2010

[VDV+13] Verdant, Arnaud ; Dupret, Antoine ; Villard, Patrick ; Alacoque, Laurent ; Mathias, Hervé ; Delgehier, Flavien: A 120μW 240$\times$ 110@ 25fps vision chip with ROI detection SIMD processing unit. In: *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on* IEEE, 2013, S. 2412–2415

[VJB+11] Villar, JI ; Juan, J ; Bellido, MJ ; Viejo, J ; Guerrero, D ; Decaluwe, J: Python as a hardware description language: A case study. In: *Programmable Logic (SPL), 2011 VII Southern Conference on* IEEE, 2011, S. 117–122

[VM10] Vosselman, G. ; Maas, H.G.: *Airborne and Terrestrial Laser Scanning*. Whittles Publishing, 2010. – ISBN 978–1439827987

[VRSPGP02] Vega-Rodríguez, Miguel A. ; Sánchez-Pérez, Juan M. ; Gómez-Pulido, Juan A.: An FPGA-based Implementation for Median Filter meeting the Real-Time Requirements of Automated Visual Inspection Systems. In: *Proc. 10th Mediterranean Conf. Control and Automation*, 2002

[VVDM07] Verdant, A. ; Villard, P. ; Dupret, A. ; Mathias, H.: SystemC Validation of A Low Power Analog CMOS Image Sensor Architecture. In: *IEEE Northeast Workshop on Circuits and Systems* IEEE, 2007, S. 903–906

[Xil16] Xilinx Inc.: *Zynq-7000 All Programmable SoC*. `http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/`. Version: November 2016

[YPEC07] Yadid-Pecht, Orly ; Etienne-Cummings, Ralph: *CMOS Imagers: From Phototransduction to Image Processing.* Springer Science & Business Media, 2007

[Zar01] Zarándy, Ákos: ACE Box: High-performance Visual Computer based on the ACE4k Analogic Array Processor Chip. In: *Proc. of ECCTD* Bd. 1, 2001

[Zar11] Zarándy, Ákos: *Focal-Plane Sensor-Processor Chips.* Springer, 2011

[ZDCE02] Zarandy, A. ; Dominguez-Castro, R. ; Espejo, S.: Ultra-High Frame Rate Focal Plane Image Sensor and Processor. In: *Sensors Journal, IEEE* 2 (2002), Dec, Nr. 6, S. 559–565. `http://dx.doi.org/10.1109/JSEN.2002.806895`. – DOI 10.1109/JSEN.2002.806895. – ISSN 1530–437X

[ZFW11] Zhang, Wancheng ; Fu, Qiuyu ; Wu, Nan-Jian: A Programmable Vision Chip Based on Multiple Levels of Parallel Processors. In: *Solid-State Circuits, IEEE Journal of* 46 (2011), Nr. 9, S. 2132–2147

[ZR05] Zarandy, A. ; Rekeczky, C.: Bi-i: A Standalone Ultra High Speed Cellular Vision System. In: *Circuits and Systems Magazine, IEEE* 5 (2005), Nr. 2, S. 36–45. `http://dx.doi.org/10.1109/MCAS.2005.1438738`. – DOI 10.1109/MCAS.2005.1438738. – ISSN 1531–636X

[ZRFS03] Zarándy, Ákos ; Rekeczky, Csaba ; Földesy, Péter ; Szatmári, István: The new Framework of Applications: The ALADDIN System. In: *Journal of Circuits, Systems, and Computers* 12 (2003), Nr. 06, S. 769–781

[ZRS03] Zarandy, A ; Rekeczky, Csaba ; Szatmari, L: Vision systems based on the 128× 128 focal plane cellular visual microprocessor chips. In: *Circuits and Systems, 2003. ISCAS'03. Proceedings of the 2003 International Symposium on* Bd. 3 IEEE, 2003, S. III–518