Effizienter Einsatz von Bildsensoren mit integrierter Signalverarbeitung

— Kurzfassung der Dissertation —

Peter Reichel

peter.reichel@eas.iis.fraunhofer.de

TECHNISCHE UNIVERSITÄT DRESDEN
FAKULTÄT INFORMATIK
und
FRAUNHOFER INSTITUT FÜR INTEGRIERTE SCHALTUNGEN (IIS)
Institutsteil Entwicklung Adaptiver Systeme (EAS)

1 Einleitung

Bildsensoren mit integrierter Signalverarbeitung – sog. *„Vision Chips“* [Moi97, Zar11] – ermöglichen die Ausführung ansonsten rechenintensiver Verarbeitungsschritte während oder unmittelbar nach der Bildaufnahme. Gegenüber konventionellen CMOS-Bildsensoren [Dur14, Oht10], die sich vor allem durch eine gute Bildqualität auszeichnen, werden die auszugebenden Daten bereits auf dem Chip auf relevante Informationen beschränkt. *Vision Chips* ermöglichen somit sehr hohe Bildwiederholraten bei gleichzeitig deutlich niedrigeren Anforderungen bzgl. der Übertragungsbandbreite, was sie besonders attraktiv für die Beobachtung sehr schneller Prozesse macht.

Für spezielle Aufgabenstellungen wurden *Vision Chips* mit festem Funktionsumfang entwickelt, die genau für eine Anwendung zugeschnitten sind. Als Beispiel sei der in der optischen Maus eingesetzte Sensor genannt, der die Verschiebung relativ zum Untergrund berechnet und lediglich 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 jedem Pixel.

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 nur für sehr wenige Sensoren Hinweise auf einen kommerziellen Einsatz. Neben einer geringen optischen Auflösung und einer eingeschränkten Empfindlichkeit können das Fehlen von integrierten Steuerwerken und die erhebliche Komplexität bzgl. der Programmierung als wesentliche Hindernisse für einen breiten Einsatz genannt werden.

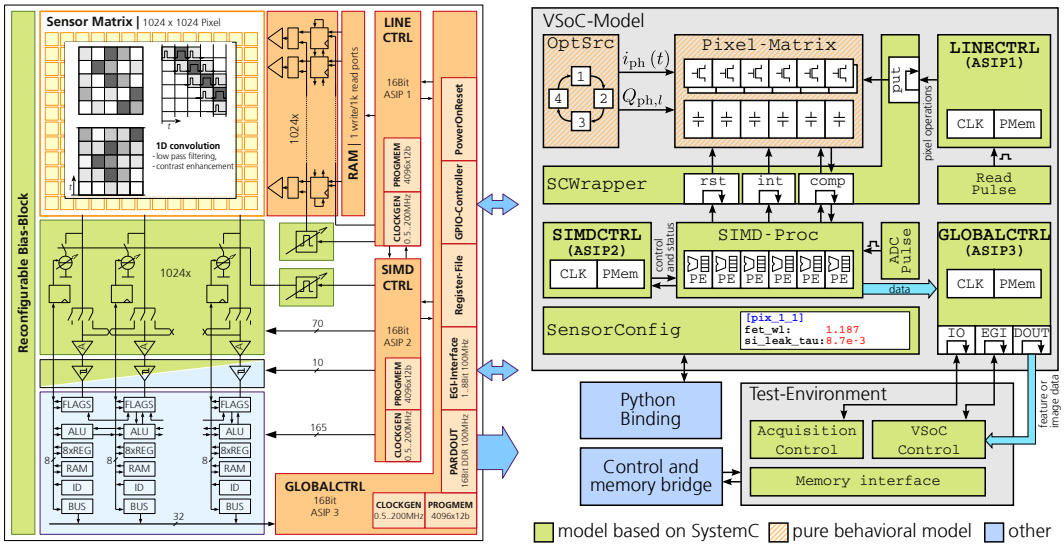
Zielstellung dieser Arbeit ist die grundlegende Analyse der für den Einsatz von *Vision Chips* notwendigen Umgebung und, darauf aufbauend, Untersuchungen zur Entwicklung der zum Entwurf, zur Realisierung sowie zur Validierung von Bildverarbeitungsaufgaben erforderlichen Infrastruktur. Mittels eines durchgängigen Systems, bestehend aus Hardware- und Software-Komponenten, soll das fehlende Bindeglied zwischen der on-chip Verarbeitung im VSoC und dessen Integration in konkrete Anwendungen geschaffen werden. Insbesondere entstanden die folgenden wesentliche Beiträge:

1. Zur Ansteuerung der Funktionseinheiten (*Functional Unit*, FU) von *Vision Chips* wurde das Konzept eines Multi-ASIP (*Application Specific Instruction-set Processor*) basierten Steuerwerks entwickelt [RDPH15]. Dieses dient zur Abstraktion des Verhaltens der FU und erlaubt mehrere parallele Kontrollflüsse. Die praktische Umsetzung des Konzepts erfolgte im Rahmen des von Döge et al. vorgestellten VSoC [DHRP15].
2. Eine zum VSoC binärkompatible Simulationsumgebung [RHDP15, RDH$^+$16] erlaubt die Betrachtung zeitabhängiger Effekte und die gezielte Injektion von Defekten und Nicht-Idealitäten sowie Untersuchungen bzgl. deren Auswirkungen auf den Prozess der Bildaufnahme und -verarbeitung.
3. Zur Ansteuerung eigener im FPGA (*Field-Programmable Gate Array*) realisierter Hardware-Komponenten wurde eine spezielle Kommunikations-Infrastruktur [RD14] entwickelt. Die gleiche Ansteuersoftware kann sowohl lokal bei Verwendung einer integrierten CPU als auch über eine Netzwerkverbindung ausgehend von einem entfernten PC bzw. basierend auf einem Modell der Hardware eingesetzt werden.
4. Zur Abbildung von Bildverarbeitungsaufgaben wurde das Konzept des *Vision Tasks* [RDP$^+$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.

2 Vision System-on-Chip

Architekturkonzept

In Abb. 1a ist die Architektur des von Döge et al. [DHRP15] vorgestellten VSoC als Übersicht dargestellt. Neben der eigentlichen, 1024×1024 Pixel umfassenden Sensor-Matrix und deren Ansteuerung besitzt dieses einen spaltenparallelen, mixed-signal Datenpfad, integrierte Takt- und Impulsgeneratoren sowie verschiedene Schnittstellen zur Dateneinund -ausgabe. Die Steuerung der einzelnen Komponenten erfolgt durch das integrierte, drei ASIPs umfassende Steuerwerk. Während LINECTRL (ASIP 1 in Abb. 1a) 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.



(a) Architektur des Bildsensor-SoC (Quelle: [DHRP15]).

(b) Struktur des Simulationssystems (Quelle: [RDH+16]).

Abbildung 1: Gegenüberstellung der Architekturen von VSoC und Simulationssystem.

Durch Verwendung ladungsbasierter Schaltungstechnik [Dög08] können Faltungsoperationen in der analogen Domäne entlang einer Spalte durchgeführt werden. Zudem erlauben die Pixel-Zellen neben einem linearen auch einen logarithmischen Betriebsmodus mit besonders hohem Dynamikumfang.

Steuerwerke für Vision Chips

Für das zur Ansteuerung von *Vision Chips* erforderliche Steuerwerk sind sowohl verschiedene Integrationsvarianten als auch unterschiedliche Realisierungen möglich. Während bei einem externen Steuerwerk alle Steuersignale der FU nach außen geführt werden, wird deren Anzahl durch einen integrierten Decoder reduziert. Wird neben dem Decoder auch der Kontrollfluss integriert, so vereinfacht sich die externe Schnittstelle deutlich. Diese dient lediglich zur Programmierung und Parametrierung bzw. zum Abrufen der Ergebnisse. Zur Realisierung des Steuerwerks werden neben endlichen Automaten und Mikroprogrammsteuerwerken zumeist (externe) Mikrocontroller eingesetzt.

Multi-ASIP basierte Steuerwerks-Architektur

Applikationsspezifische Prozessoren besitzen spezielle Eigenschaften, durch die der Einsatz in bestimmten Anwendungsgebieten wesentlich erleichtert oder überhaupt erst ermöglicht wird. Im Gegensatz zum klassischen ASIP-Ansatz zur Beschleunigung rechenintensiver Programmabschnitte, steht bei der Verwendung eines ASIPs als Steuerwerk die Integration der anzusteuernden FU in den Befehlssatz im Vordergrund.

Die Basis der im Rahmen dieser Arbeit entwickelten Steuerwerksarchitektur [RDPH15] bildet ein Stack-basierter 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 zum Laden der für Berechnungen notwendigen Operanden jedoch mehr Instruktionen erforderlich. Der Verzicht auf die explizite Adressierung ermöglicht eine einfache Schnittstelle für Befehlssatzerweiterungen. Dazu werden dem Prozessorkern zusätzliche externe Decoder (EDEC) zur Seite gestellt. Wird ein externer Befehl erkannt, so wird die Kontrolle an den entsprechenden EDEC übergeben. Dem EDEC werden neben dem aktuellen Befehlswort auch die oberen beiden Stack-Einträge sowie die Statusregister der ALU als Parameter zur Verfügung gestellt. Mit Abschluss einer Instruktion kann der EDEC den Stack manipulieren und z.B. Ergebnisse ablegen.

Zur Unterstützung mehrerer Kontrollflüsse zur Ansteuerung parallel arbeitender FU, werden anstelle eines einzelnen, großen Decoders, der die Steuersignale aller FU bereitstellt, die FU in Gruppen zusammengefasst. Alle FU einer Gruppe werden durch einen gemeinsamen Decoder angesteuert, der jeweils in einen separaten Prozessor integriert wird. Zur Berücksichtigung von Abhängigkeiten der Kontrollflüsse bzw. der FU untereinander stehen Methoden zur Synchronisation sowie zum Datenaustausch bereit.

3 Simulationsumgebung

Architekturübersicht

Bedingt durch die Heterogenität der Architektur des VSoC sind Simulation und Test unter reproduzierbaren Bedingungen ein wesentlicher Bestandteil des Entwurfs sowie der Implementierung von Bildverarbeitungsalgorithmen. Beim klassischen, Kamera-PC-basierten Ansatz wird dazu die Kamera durch eine Bilddatenbank ersetzt. In einem *Vision Chip* ist hingegen die Betrachtung von Bildaufnahme und Bildverarbeitung nicht unabhängig voneinander möglich.

In Abb. 1b ist die Architektur des strukturtreuen Modells des VSoC als Übersicht dargestellt, das die Grundlage des Simulationssystems [RHDP15, RDH+16] bildet. Analog zum VSoC selbst (vgl. Abb. 1a) umfasst dieses ein Modell der Pixelmatrix sowie Modelle der einzelnen ASIPs. Zudem wird die beobachtete Szene durch ein Szenen-Modell (`OptSrc`) beschrieben. Zur Steigerung der Simulationsgeschwindigkeit wurde auf die strukturtreue Modellierung der Pixelmatrix auf Schaltungsebene verzichtet, weshalb durch die Komponente `SCWrapper` eine Ankopplung an den in SystemC implementierten Teil des Modells erfolgt (siehe Abb. 1b). Die spezielle Komponente `SensorConfig` 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.

Modellierung von Bildaufnahme und -verarbeitung

Grundlage des Szenen-Modells bildet ein Ring-Puffer, in welchen eine Sequenz aufeinanderfolgender Bilder mit gegebenem zeitlichem Abstand abgelegt wird, die sich daraufhin periodisch wiederholt. Dabei werden die RGB-Helligkeitswerte jedes Pixels l als lokale Bestrahlungsstärke in drei Spektralbereichen interpretiert. Die Berechnung des Fotostromes $i_{\mathrm{ph},l}(t)$ als Funktion der Zeit erfolgt unter Verwendung der abgelegten Bilder für jedes Pixel anhand der jeweiligen spektralen Empfindlichkeit, d.h. der Anzahl erzeugter Elektronen pro Photon. Durch stückweise Integration kann die während eines Zeitintervalls $[t_1, t_2]$ akkumulierte Ladung $Q_{\mathrm{ph},l} = \int_{t_1}^{t_2} i_{\mathrm{ph},l}(t)\, dt$ berechnet werden. 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.

Abhängig von den durch LINECTRL vorgegebenen Steuerworten wird der resultierende Ausgabestrom des Pixels als Funktion von $i_{\mathrm{ph},l}(t)$ und $Q_{\mathrm{ph},l}$ berechnet. Dabei wird eine vereinfachte Ersatzschaltung der Pixel-Zellen angenommen, deren Simulation zeitdiskret erfolgt, wodurch auf die Lösung der sonst notwendigen Differenzialgleichungen verzichtet werden kann.

Einsatz in Anwendungsentwicklung

Sowohl zur Parametrierung als auch zur Injektion von Defekten und Nicht-Idealitäten werden sämtliche Parameter der vereinfachten Ersatzschaltung der Pixelzellen durch `SensorConfig` vorgegeben. So können z.B. durch Veränderung der Transistorparameter gezielt defekte Pixel injiziert oder *Fixed-Pattern Noise* (FPN) eingebracht und somit die Auswirkung auf Bildaufnahme und -verarbeitung untersucht werden.

Im Gegensatz zum Test auf der realen Hardware, bei dem lediglich die ausgegebenen Ergebnisse evaluiert werden können, erlaubt die Validierung unter Verwendung des Simulationssystems die Betrachtung des internen Zustands der digitalen und analogen Komponenten zu einem beliebigen Zeitpunkt. So kann auch der Einfluss von Defekten und Nicht-Idealitäten bestimmt und z.B. mit dem idealen Verhalten verglichen werden. Der Zustand einzelner Komponenten wird dazu zu einem bestimmten Zeitpunkt gespeichert („*Snapshot*“), d.h. beispielsweise bei Erreichen bestimmter Programmabschnitte.

4 Bildaufnahme- und Verarbeitungsplattform

Entwicklungs- und Kamerasystem

Zur Vereinfachung der Ansteuerung eigener, im FPGA realisierter Hardware-Komponenten durch Linux-basierte Anwendungsprogramme wurde eine spezielle Kommunikations-Infrastruktur [RD14] entwickelt. Diese stellt elementare Komponenten, wie z.B. einzelne Register oder Registerblöcke, bereit, die durch eine baumartige Struktur mit einer integrierten CPU verbunden werden. Jede Basis-Komponente ermöglicht zudem die asynchrone Benachrichtigung der Steuersoftware durch die umgebende Logik. Zur Ansteuerung der Komponenten wurde ein spezieller Kernel-Treiber entwickelt, der den Zugriff direkt aus

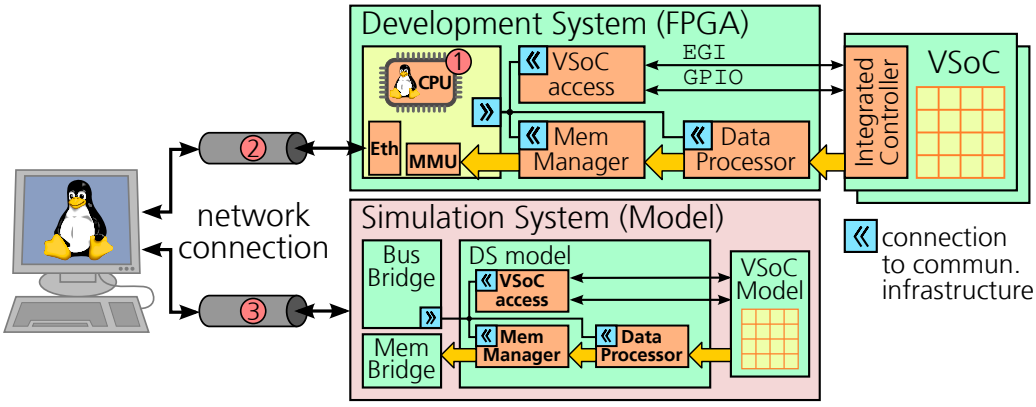


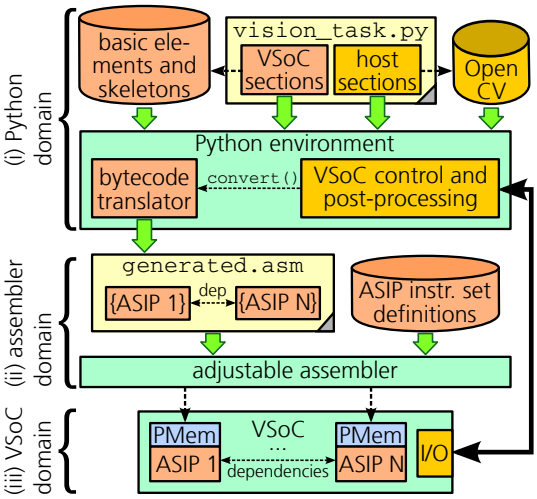
Abbildung 2: Darstellung der Interaktion zwischen Anwendungssoftware und Entwicklungs- bzw. Simulationsumgebung (aus [RDH+16]).

dem *Userspace* erlaubt sowie eine objektorientierte Bibliothek zur Anwendungsentwicklung in C/C++ bzw. Python. Anstelle des direkten Zugriffs auf die Hardware mit Hilfe des Kernel-Treibers können Datenzugriffe auch über eine Netzwerkverbindung (LAN), ausgehend von einem entfernten PC, erfolgen. Auch die Ansteuerung von Modellen der Hardware ist auf dem gleichen Weg möglich.

In Abb. 2 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 eine Schnittstelle zur Ansteuerung des VSoC, eine Bildspeicherverwaltung sowie ggf. weitere Verarbeitungsfunktionen. Zur Verwendung der Simulationsumgebung – komplementär zum DS – wurden dessen wesentliche Komponenten ebenfalls modelliert. Durch den Einsatz der entwickelten Kommunikations-Infrastruktur kann die gleiche Ansteuersoftware sowohl auf einer im DS integrierten CPU (1) als auch auf einem entfernten PC (2) sowie in Kombination mit dem Simulationssystem (3) verwendet werden.

Programmierung des ASIP-basierten Steuerwerks

Zur Umsetzung konkreter Bildverarbeitungsaufgaben unter Verwendung des VSoC ist die Partitionierung der Aufgabenstellung in VSoC-basierte Verarbeitungsschritte und möglicherweise konventionelle Nachverarbeitung erforderlich. Der Entwurfsfluss des im Rahmen dieser Arbeit entwickelten Konzepts [RDP+16] zur ganzheitlichen Betrachtung von Bildverarbeitungsaufgaben ist in Abb. 3a als Übersicht dargestellt. Dieser lässt sich grob in drei Domänen unterteilen: Die untere Domäne (Abb. 3a (iii)) repräsentiert das VSoC mit dem integrierten Steuerwerk. Die zweite, mittlere Domäne (ii) 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 beschrieben werden, um Abhängigkeiten zwischen den ASIPs auszudrücken, aber auch um Optimierungen über mehrere ASIPs zu ermöglichen. Die obere Domäne (i) enthält schließlich eine auf Python basieren-



(a) Zusammenhängende Darstellung des Entwurfsflusses zur Programmierung des Multi-ASIP-basierten Steuerwerks (aus [RDP+16]).

```
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   ...
```

(b) Beispiel der Python-Beschreibung eines *Vision Tasks* mit Nachverarbeitung basierend auf OpenCV.

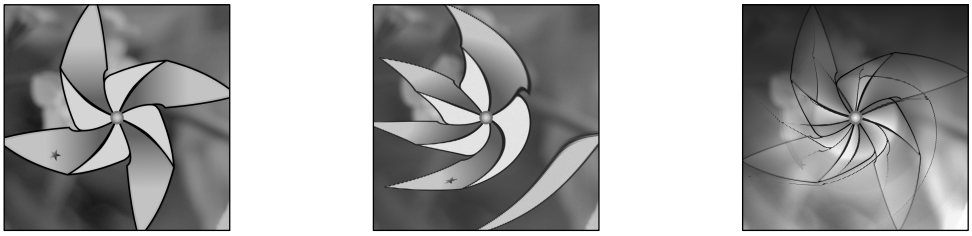
Abbildung 3: High-Level Programmierung des VSoC.

de 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 Bibliotheken mit Funktionen zur Bildverarbeitung sowie zur Steuerung der Bildaufnahme bereit. 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 zum Einsatz kommen.

Abbildung von Bildverarbeitungsaufgaben

Neben der Partitionierung in VSoC-basierte Vor- sowie konventionelle Nachverarbeitung ist auch die Parametrierung der Entwicklungsplattform sowie die eigentliche Programmierung des VSoC erforderlich. Die gemeinsame Betrachtung dieser Aufgabenstellungen soll im Folgenden als „*Vision Task*“ bezeichnet werden.

Die für Bildaufnahme und -verarbeitung auf dem VSoC notwendigen Abläufe erfordern ein komplexes Synchronisationsschema zwischen den einzelnen, parallelen Kontrollflüssen des Steuerwerks sowie der Entwicklungsplattform und ggf. weiteren externen Sensoren und Aktoren. Zur Vereinfachung der Implementierung werden parametrierbare und erweiterbare *Skeletons* bereitgestellt, in denen generelle Abläufe sowie die erforderliche



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

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

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

Abbildung 4: Darstellung der zeitabhängigen Effekte verschiedener Auslese-Modi bei Verwendung des Simulationssystems (aus [RDH+16]).

Synchronisation und Kommunikation hinterlegt werden. Zur Anpassung des vorgegebenen Schemas stehen *Slots* zur Registrierung von Routinen an bestimmten Positionen der Verarbeitung bereit. Zudem wird eine Schnittstelle zur Parametrierung vorgegeben.

In Abb. 3b 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. Zunächst erfolgt die Auswahl und Konfiguration der Plattform (Abb. 3b (1) und (2)) und eines *Skeletons* (Abb. 3b (4)) sowie die Registrierung einer eigenen Verarbeitungsroutine (3). Nach der Programmierung des VSoC (5) und der Parametrierung des *Skeletons* (6) werden die ausgegebenen Daten mit Hilfe von OpenCV weiterverarbeitet (7).

5 Anwendungsbeispiele

Als Basis zur Implementierung von Bildaufnahme- und -verarbeitungsfunktionen wurden die Auslese-Modi „*Global Shutter*“, „*Rolling Shutter*“ und „*Global Reset*“ als *Skeletons* realisiert. Verändert sich die beobachtete Szene während der Belichtungszeit, so hat deren zeitliches Verhalten erhebliche Auswirkungen auf die Bildaufnahme. Unter Verwendung des Simulationssystems wurden alle drei Auslese-Modi zur Beobachtung der gleichen Szene eingesetzt. Dazu wurde eine synthetische Bildsequenz erstellt, in der ein Windrad in 360 Schritten um jeweils $\varphi = 1°$ gedreht dargestellt ist. Die Bilder wurden mit einem zeitlichen Abstand $\tau_{\mathrm{Dist}} = 50\,\mu\mathrm{s}$ in den Ring-Puffer des Szenen-Modells geladen. Während bei „*Global Shutter*“ (Abb. (4a)) aufgrund der gleichzeitigen Aufzeichnung aller Pixel keine zeitabhängigen Effekte sichtbar sind, erscheinen sowohl bei „*Rolling Shutter*“ (Abb. (4b)) als auch bei „*Global Reset*“ (Abb. (4c)) deutliche Artefakte.

Mithilfe der verschiedenen Auslese-Modi wurden drei *Vision Tasks* realisiert:

1. Als Teil eines Präsenzdetektionssystems werden interessante Punkte mit Hilfe des FAST-Operators (*Features from Accelerated Segment Test*) extrahiert und ausgegeben.

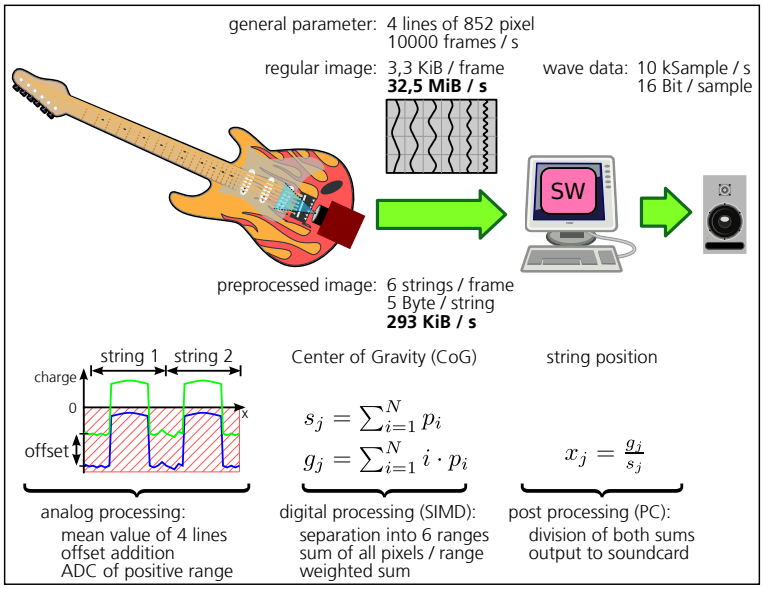


Abbildung 5: Funktionsprinzip der optischen Tonabnahme von einer E-Gitarre.

Zusätzlich werden defekte Pixel im Vorfeld mit Hilfe eines Median-Filters entfernt sowie die Belichtungszeit automatisch nachgeführt. Die resultierende Kompressionsrate entspricht ca. 39:1.

2. Zur optischen Tonabnahme an einer E-Gitarre wird die Position der sechs Saiten subpixel-genau und mit einer Bildwiederholrate von 10000 $^{\text{Bilder}}/_{\text{s}}$ bestimmt und zum PC zur Ausgabe übertragen (siehe Abb. 5). Es kommen sowohl analoge als auch digitale Operationen des VSoC zum Einsatz. Durch die Berechnung der Positionen auf dem VSoC ist anstelle von $32,5\,^{\text{MiB}}/_{\text{s}}$ lediglich eine Bandbreite von $293\,^{\text{KiB}}/_{\text{s}}$ erforderlich, was einer Kompressionsrate von 114:1 entspricht.
3. Bei der Laser-Triangulation zur Vermessung der Oberflächengeometrie eines Objektes wird eine Linie auf dessen Oberfläche projiziert und deren Abbildung mit einer Kamera beobachtet. Durch den bekannten Winkel zwischen Laser und Kamera kann anhand der Verformung der Linie auf die Höheninformation geschlossen werden. Die aus den mit der Kamera aufgenommenen Bildern extrahierte Information beschränkt sich auf die exakte Position der Abbildung der Laser-Linie. Durch die Umsetzung unter Verwendung analoger Faltungsoperationen wird die Ausgabe auf die tatsächlichen Profildaten beschränkt und ein Kompressionsfaktor von ca. 250:1 erreicht.

6 Zusammenfassung

Zur Abstraktion des Verhaltens der Funktionseinheiten wurde ein integriertes Multi-ASIP-basiertes Steuerwerk und dessen Umsetzung im betrachteten VSoC vorgestellt. Neben der Bereitstellung mehrerer Kontrollflüsse wird durch den Verzicht auf fest vor-

gegebene Abläufe die Flexibilität der Architektur maximiert. Zur Bereitstellung einer geeigneten Simulationsumgebung wurde ein strukturtreues, binärkompatibles Modell des VSoC entwickelt. Durch das gezielte Einbringen von Fehlern und Nicht-Idealitäten sowie die Berücksichtigung zeitlicher Effekte wird die Untersuchung des Verhaltens implementierter Algorithmen ermöglicht. Für den Einsatz des VSoC in realen Bildverarbeitungsaufgaben wurde eine Kamera- und Entwicklungsplattform vorgestellt. Deren Basis bildet eine Kommunikationsinfrastruktur, die die einfache Ansteuerung eigener, im FPGA realisierter Komponenten durch Anwendungssoftware erlaubt. Diese ist auch die Grundlage zum Einsatz des Simulationssystems komplementär zur Verwendung der tatsächlichen Entwicklungsplattform. 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.

Literatur

[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

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

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

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

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

[RD14] 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

[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

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