Lehrstuhl für Integrierte Schaltungen

Technische Universität München

Entwurfsmethodik für eine flexible Architektur zur Videoobjekt-Segmentierung

Hubert Mooshofer

Lehrstuhl für Integrierte Schaltungen

Technische Universität München

Entwurfsmethodik für eine flexible Architektur zur Videoobjekt-Segmentierung

Hubert Mooshofer

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. rer nat. M. Lang
Prüfer der Dissertation:	
	1. Univ.-Prof. Dr.-Ing. I. Ruge
	2. Univ.-Prof. Dr.-Ing. G. Färber

Die Dissertation wurde am 02.11.2001 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 06.06.2002 angenommen.

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Angestellter am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München.

Mein erster Dank gilt meinem Doktorvater, Herrn Prof. Dr.-Ing. I. Ruge, der diese Arbeit ermöglicht hat und mir die Möglichkeit gab, in den hochinteressanten Gebieten Videosignalverarbeitung und Entwurf von Hardware-Architekturen zu forschen.

Daß mir Gelegenheit gegeben wurde, im Rahmen des COST211-Projektes mit Experten im Bereich Bildanalyse zusammenzuarbeiten, weiß ich sehr zu schätzen. Auch die Teilnahme an dem europäischen Forschungsprojekt MPEG F(o)ur Mobiles, ermöglichte mir viele interessante Kontakte.

Zu Dank bin ich auch der Siemens AG verpflichtet, die mich durch die Gewährung des Ernst von Siemens Stipendiums bei meiner Arbeit unterstützte. Namentlich möchte ich an dieser Stelle besonders Herrn Prof. Dr. M. Groß und Herrn Dr. E. Hundt danken.

Herrn Prof. Dr.-Ing. G. Färber, dem Inhaber des Lehrstuhls für Realzeit-Computersysteme an der Technischen Universität München, danke ich für sein Interesse an der Arbeit und die Übernahme des Zweitberichtes.

Weiterer Dank gilt Herrn Prof. F. Meyer, Ph.D. und Herrn Prof. Dr. J.-C. Klein von der Ecole des Mines de Paris für den intensiven fachlichen Austausch und viele interessante Diskussionen.

Dem Leiter der Arbeitsgruppe Videosignalverarbeitung, Herrn Dr.-Ing. W. Stechele, danke ich für sein Vertrauen und die stets gewährte fachliche und administrative Unterstützung die wesentlich zum Gelingen dieser Arbeit beitrug.

Für die immer sehr gute Zusammenarbeit, viele fruchtbaren Diskussionen und das sehr angenehme Arbeitsklima möchte ich mich bei meinen jetzigen und ehemaligen Kollegen am Lehrstuhl bedanken. Dabei möchte ich Georg Diebel, Michael Eiermann, Stephan Herrmann, Dr. Andreas Hutter, Peter Kindsmüller, Torsten Mahnke, Nuria Pazos-Escudero, Thomas Wild und Armin Windschiegl namentlich hervorheben, die mir mit ihrem Wissen, ihrem Einsatz und als Diskussionspartner eine große Hilfe waren.

Nicht zuletzt gilt meinen Eltern großer Dank, für das Korrekturlesen des Manuskripts und die moralische Unterstützung.

München, im Oktober 2001

Zusammenfassung

In zukünftigen Videodiensten wird objektbasierte Videokodierung eine wichtige Rolle spielen, da objektbasierte Funktionalität interaktive Anwendungen, objektbasierte Manipulation und inhaltsadaptive Kodierung ermöglicht. Der objektbasierten Videokodierung wurde durch die Standardisierung von MPEG-4 bereits entscheidender Vorschub geleistet, da damit eine einheitliche Plattform zur Kodierung beliebig geformter Videoobjekte bereitsteht. Voraussetzung für objektorientierte Videokodierung ist jedoch die Abgrenzung von Videoobjekten gegeneinander bzw. gegenüber dem Hintergrund, so daß mit zunehmender Nutzung objektorientierter Videokodierung, die Segmentierung von Videoobjekten an Bedeutung gewinnt. Neben diesem Hauptanwendungsgebiet ergeben sich im Bereich Multimediadatenbanken weitere Anwendungsmöglichkeiten, wie beispielsweise die Suche nach Personen, Gegenständen oder anderen Videoobjekten.

Operationen, die direkt Bilddaten verarbeiten, bedingen den größten Teil des Rechenaufwandes von Videoobjekt-Segmentierungsalgorithmen. Häufig benutzte Operationen sind dabei lineare und nichtlineare Filter (insbesondere Kantenoperatoren), morphologische Operationen, Operationen zur Bildverknüpfung, Ausbreitungsprozesse und die Berechnung von Segmenteigenschaften. Da die Adressierung der Daten bei vielen Operationen auf ähnliche Weise erfolgt, läßt sie sich auf vier grundlegende Adressierungsarten zurückführen: Die Adressierung bei Operationen, die zwei Bildpunkte aus verschiedenen Bildern als Eingangsdaten benutzen, wird in Anlehnung an die Nomenklatur bei Videokodierung als Inter-Adressierung bezeichnet. Operationen, die räumlich benachbarte Bildpunkte innerhalb eines Bildes als Eingangsdaten benutzen, verwenden Intra-Adressierung und Operationen, die beliebig geformte Bereiche entsprechend dem Bildinhalt bearbeiten, verwenden Segment-Adressierung. Zur Verwaltung von Segmenteigenschaften wird schließlich die segmentindizierte Adressierung eingesetzt.

Segmentierungsverfahren zur Videoobjektgenerierung sind sehr komplex und erfordern daher eine hohe Verarbeitungsleistung. Demgegenüber stehen die Echtzeitanforderungen von Anwendungen im Kommunikations- und Überwachungsbereich und der Bedarf an schneller Verarbeitung bei anderen Anwendungen. Im Hinblick auf mobile Anwendungen ist außerdem die kompakte Implementierung mit niedrigem Leistungsverbrauch gefordert. Universalprozessoren bieten trotzt hohem Flächenaufwand derzeit keine ausreichende Rechenleistung und besitzen bei vollständiger Auslastung einen sehr hohen Leistungsverbrauch. Aus diesem Grund sind auf Videosegmentierung ausgerichtete Lösungen erforderlich, die durch ein hohes Maß an Parallelität eine hohe Verarbeitungsleistung erzielen. Eine weitere Anforderung ist es, ausreichende Verarbeitungsleistung bei niedriger Taktfrequenz zu erreichen, damit der Leistungsverbrauch durch Absenkung der Versorgungsspannung reduziert werden kann. Da Videoobjekt-Segegmentierungsverfahren viele unterschiedliche Operationen beinhalten, und um ein breites Einsatzspektrum abzudecken, müssen derartige Lösungen außerdem ein hohes Maß an Flexibilität bieten.

Bekannte Architekturen, die zur Implementierung in Segmentierungsalgorithmen enthaltener Operationen in Frage kommen, lassen sich in universelle, problemangepaßte und dedizierte Architekturen unterteilen. Dedizierte Architekturen erweisen sich als nicht ausreichend flexibel, da sie nur einzelne Operationen unterstützen. Zu den universellen Architekturen zählen auf Videosignalverarbeitung ausgerichtete Prozessoren, Prozessorsysteme und konfigurierbare Systeme. Während diese hohe Flexibilität bieten, bedingt ihre Universalität einen großen flächenmäßigen und zeitlichen Zusatzaufwand. Am besten geeignet sind SIMD-Videoprozessoren, da sie Datenparallelität mit verhältnismäßig geringem Aufwand nutzen können. Sie eignen sich jedoch nicht besonders gut für auf Segment-Adressierung basierende Operationen, bei denen geringe Datenparallelität vorhanden ist. Problemangepaßte Architekturen unterteilen sich in datenparallele und befehlsparallele Architekturen. Datenparallele problemangepaßte Architektu-

ren kommen für die Videoobjekt-Segmentierung nicht in Frage, da bei Operationen, die auf Ausbreitungsprozessen basieren, nur wenige Prozessorelemente tatsächlich genutzt werden. Befehlsparallele problemangepaßte Architekturen unterstützen entweder Operationen mit regulärer Adressierung oder solche, die auf Ausbreitungsprozessen beruhen. Da jedoch in Algorithmen zur Videoobjekt-Segmentierung Operationen mit Inter-, Intra- und Segment-Adressierung eingesetzt werden, ist sowohl die Unterstützung regulärer Adressierung als auch die Unterstützung von Ausbreitungsprozessen erforderlich, bei Operationen mit Segment-Adressierung sogar beides in schnellem Wechsel.

Im Rahmen dieser Arbeit wird eine Architektur vorgeschlagen, die beide Arten der Adressierung unterstützt. Da dem Bedarf einer Datenzufuhr mit großer Bandbreite bei kleiner Anzahl an Adressierungsarten, die hohe Anzahl unterschiedlicher zu unterstützender Operationen gegenübersteht, wurde die Trennung von Adressierung und Verarbeitung als Ansatzpunkt für das Konzept der vorgeschlagenen Architektur gewählt. Das Konzept sieht die dedizierte Implementierung der generischen Adressierungsarten und die konfigurierbare Implementierung der Verarbeitung vor, um gleichzeitig eine effiziente Datenzufuhr und ein hohes Maß an Flexibilität zu erzielen. Im Hinblick auf die geforderte hohe Verarbeitungsleistung bei niedriger Taktfrequenz wird die innerhalb der Bildpunktoperationen vorhandene Parallelität vollständig genutzt. Auf die Parallelisierung von Bildpunktoperationen, die die einsetzbaren Adressierungsarten einschränken würde, wird zugunsten hoher Flexibilität verzichtet. Zur Steigerung der Verarbeitungsleistung wird die Möglichkeit zur Parallelisierung von Bild- und Segmentoperationen durch Zusammenschaltung mehrerer Instanzen der Architektur vorgesehen.

Um von dem vorgeschlagenen Konzept zu einer effizienten Architektur zu gelangen, sind jedoch entscheidende Fragestellungen im Hinblick auf Balancierung von Adressierung und Verarbeitung, Ort der Datenspeicherung, Datenpufferung, Speicherorganisation und -technologie zu klären. Dazu müssen Architekturalternativen bewertet und verglichen werden. Die Schwierigkeit dabei liegt insbesondere darin, daß sich bei komplexen Algorithmen mit datenabhängigem Verhalten - wie in diesem Fall gegeben - keine Bewertung auf analytischem Weg vornehmen läßt. Eine Architektursimulation unterstützt den Entwurfsprozeß jedoch nur, wenn sich Simulationsmodelle auf einfache Weise erstellen lassen, ohne daß der Architekturentwurf tatsächlich durchgeführt werden muß. Aus diesem Grund wird eine Simulationsmethodik entwickelt, die auf der getrennten Modellierung von Algorithmus und Architektur beruht und eine gekoppelte Simulation beider Modelle gestattet. Die im Algorithmenmodell durchgeführten Operationen und Datenzugriffe steuern das Architekturmodell so, daß datenabhängige Abläufe, statistisches Verhalten, Ressourcenkonflikte und -auslastung simuliert werden können. Durch die getrennte Modellierung wird der Einsatz abstrakter Architekturmodelle ermöglicht und der Modellierungsaufwand verringert. Außerdem wird die schrittweise Verfeinerung des Architekturmodells ermöglicht und eine hohe Simulationsgeschwindigkeit erreicht, so daß komplexe Architekturen simuliert werden können.

Auf der Basis der entwickelten Methodik werden Simulationsmodelle für das vorgeschlagene Architekturkonzept und mögliche Architekturvarianten erstellt und deren Verarbeitungsleistung simuliert. Die Untersuchung einer Referenzarchitektur, bei der zu verarbeitende Daten direkt aus einem - aufgrund seiner Größe externen - Speicher geladen bzw. in diesen geschrieben werden, zeigt, daß ohne spezielle architekturelle Maßnahmen die Speicherschnittstelle die Verarbeitungsgeschwindigkeit des Systems bestimmt, während die Verarbeitungseinheit nicht ausgelastet ist. Um eine Steigerung der Verarbeitungsgeschwindigkeit und eine ausgeglichene Auslastung von Adressierungs- und Verarbeitungseinheit zu erzielen, wird daher untersucht, wie die effektive Speicherbandbreite durch Einsatz eines Pufferspeichers vergrößert werden kann. Die Simulation eines Zeilenspeichers und eines zweidimensionalen Caches zeigt, daß der Zeilenspeicher bei regulären Zugriffen und der Cache bei irregulären Zugriffen effektiv arbeitet. Um zu einem Pufferungskonzept zu gelangen, das für reguläre und irreguläre Zugriffe gleichermaßen geeignet ist,

wurde die Erweiterung des Caches durch vorausschauendes Laden zur Verbesserung der Trefferrate bei regulärer Adressierung untersucht. Der Vergleich zeigt, daß damit bei regulärer Adressierung eine dem Zeilenspeicher vergleichbare Pufferwirkung erreicht wird. Beim Vergleich unterschiedlicher Cache-Parameter erwies sich eine Blockgröße von 4x4 Bildpunkten und eine direkte Abbildung der Cache-Blöcke bei impliziter Verdrängungsstrategie als optimal. Auf der Basis weiterer Simulationen wurde die Cache-Größe bestimmt, und nachgewiesen, daß der Cache auch mit dynamischem Speicher eine schnelle Datenversorgung gestattet. Die Untersuchung der Pufferung segmentindizierter Daten zeigt schließlich, daß bereits durch ein einzelnes Pufferregister eine deutliche Bandbreitenreduktion erzielt werden kann, während ein Cache nur eine geringe weitere Reduktion bewirkt. Weitere Simulationen ergaben, daß die Architektur des Koordinatenspeichers, der bei Segment-Adressierung innerhalb einer zeitkritischen Schleife liegt, nicht auf paralleles Abspeichern mehrerer Koordinaten ausgerichtet werden braucht, sondern im Hinblick auf schnellen Zugriff optimiert werden muß. Außerdem wurde gezeigt, daß der Koordinatenspeicher im Verglich zur Gesamtzahl der Bildpunkte deutlich weniger Koordinaten zu fassen braucht und somit kompakt implementiert werden kann.

Auf der Basis dieser Untersuchungen wird eine Architektur vorgeschlagen, die im Hinblick auf weitreichendere Einsatzmöglichkeiten im Bereich der Bildanalyse mit dem Akronym CONIAN benannt wird, das für *CONfigurable Image ANalysis* Koprozessor steht. Die CONIAN-Architektur besteht aus dem Speichersystem, das sich aus Adreßeinheit und Koordinatenspeicher zusammensetzt, einer konfigurierbaren Verarbeitungseinheit und der Steuerungseinheit.

Die Adreßeinheit unterstützt Inter-, Intra-, Segment- und segmentindizierte Adressierung. Sie ermöglicht den Zugriff auf Bilddaten und segmentindizierte Daten über zwei intern getrennte Datenpfade, die zu einer externen Speicherschnittstelle führen. Ein Pufferregister vermeidet wiederholte Zugriffe auf dieselben segmentindizierten Daten. Die Adreßeinheit enthält einen acht Speicherbänke umfassenden Cache für Bilddaten, der zweidimensional in Form von 4x4 Blöcken organisiert ist. Durch den Cache können gleichzeitig alle Farb-, Helligkeits- und sonstigen Komponenten eines Bildpunktes und seiner vier direkten Nachbarpunkte in nur einem Takt zugegriffen werden. Eine komplette 3x3 Nachbarschaft kann in 2 Takten geladen bzw. gespeichert werden. Das Laden und Zurückschreiben von Cache-Blöcken erfolgt über die 128 bit breite Speicherschnittstelle, die pro Takt alle Komponenten von zwei Bildpunkten übertragen kann. Bei regulärer Adressierung erfolgt vorzeitiges Laden bzw. Zurückschreiben von Cache-Blöcken im Hintergrund, so daß Zugriffe auf Bilddaten ohne Verzögerung stattfinden.

Der Koordinatenspeicher speichert bei Ausbreitungsprozessen die Position von Punkten zwischen, bis diese zu einem späteren Zeitpunkt verarbeitet werden, und liefert beim Auslesen gleichzeitig mit den Koordinaten eines Punktes dessen Distanzwert gegenüber den Startpunkten. Der Koordinatenspeicher ist hierarchisch organisiert, so daß Operationen unterstützt werden, die auf Ausbreitungsprozessen mit adaptiver Geschwindigkeit beruhen.

Die Verarbeitungseinheit besteht aus mehreren Einheiten, die parallele Verarbeitung von Helligkeits- und Farbkomponenten gestatten. Durch „grobkörnige“ Konfiguration wird der Flächenaufwand und die zusätzliche Zeitdauer für Konfigurationselemente klein gehalten, während durch unabhängige Konfiguration von Teiloperationen eine hohe Flexibilität bereitgestellt wird. Die Vor- und Nachverarbeitung, wie Zahlenformat-Konvertierung, Kanalkombination, Maskierung entsprechend einem Strukturelement, Wertebereichsbegrenzung und Nachbarschaftsprüfung werden durch eigene Einheiten übernommen. Die Teiloperationen von Inter-, Intra-, Segment- und segmentindizierten Operationen werden durch parametrisierbare Module implementiert, deren Verschaltung konfiguriert werden kann. Das Bindeglied zwischen Adreßeinheit und Verarbeitungseinheit stellt die Registermatrix dar. Sie enthält die Bilddaten eines ein- bzw. zweidimensionalen Ausschnitts, der maximal 3x3 bzw. 9x1 Bildpunkte umfaßt und die Eingangsdaten für die Verarbeitungseinheit enthält. Vorladeregister ermöglichen die vollständig überlappende Ope-

ration von Adress- und Verarbeitungseinheit. Durch Bypass-Pfade können rekursive Operationen effizient implementiert werden, bei denen das Verarbeitungsergebnis unter Umgehung des Caches direkt im nächsten Schritt wieder als Eingangsdatum benutzt wird. Die Registermatrix unterstützt Schiebeoperationen, so daß bei regulärer Adressierung lediglich 1 Takt zum Laden einer 3x3 Nachbarschaft erforderlich ist.

Die Steuerungseinheit besteht aus zwei Untereinheiten, die den Ablauf des Verarbeitungszyklus eines Bildpunktes bzw. die Reihenfolge der Verarbeitung von Bildpunkten steuern. Der Ablauf eines Verarbeitungszyklus wird flexibel entsprechend der unterschiedlichen Verarbeitungsdauer und Anzahl an Zugriffen gesteuert, während die Taktfrequenz an der Speichergeschwindigkeit ausgerichtet ist.

Die CONIAN-Architektur wurde in der Hardware-Beschreibungssprache VHDL modelliert und verifiziert. Eine Synthese und Abschätzung für eine 0,25 µm Standardzellen-Technologiebibliothek ergab inklusive Verdrahtung einen Flächenbedarf von insgesamt 47,7 mm^2. Anhand des VHDL-Modells wurde die Verarbeitungsdauer für die unterstützten Operationen durch eine statische Analyse des Zeitverhaltens ermittelt. Abgesehen von der Relaxation, die aufgrund ihrer Komplexität zwei Takte länger dauert, ergibt sich für alle anderen Operationen eine Dauer von 1 bis 5 Taktzyklen pro Bildpunkt bei einer maximal erreichbaren Taktfrequenz von 200 MHz. Die Simulation der Geschwindigkeit eines komplexen iterativen Farbesegmentierungsalgorithmus belegt die Eignung für Echtzeitimplementierungen. Für das QCIF Bildformat wird eine Verarbeitungsgeschwindigkeit von 29,3 Bilder pro Sekunde erreicht, wenn jedes Bild segmentiert wird, bzw. 65,9 Bilder pro Sekunde, wenn die Segmentierung für jedes vierte Bild erfolgt und dazwischen eine Bewegungskompensation vorgenommen wird.

Zur Beurteilung der Architektur wurde ein Vergleich mit dem Celeron Prozessor durchgeführt, einem Universalprozessors, dessen Implementierung auf einer Technologie mit gleicher Strukturgröße beruht. Während für den Celeron 11,5 Sekunden Verarbeitungszeit für einen Segmentierungsalgorithmus gemessen wurden, ergab die Simulation für den CONIAN lediglich 0,48 Sekunden. Die vorgeschlagene Architektur erreicht somit bei einer um 64% niedrigeren Fläche eine deutlich höhere Geschwindigkeit. Gegenüber der Verarbeitungszeit, die für einen mit 2 GHz getakteten Pentium IV bei Nutzung von MMX geschätzt wurde, erreicht die CONIAN-Architektur eine um den Faktor 4,6 höhere Geschwindigkeit. Die Taktfrequenz des CONIAN beträgt ein zehntel der des Pentium IV und weniger als die Hälfte der des Celeron, so daß die Forderung nach hoher Verarbeitungsleistung bei niedrigerer Taktfrequenz erfüllt wird. Der Vergleich der CONIAN-Architektur mit bekannten für Videoobjekt-Segmentierung einsetzbaren Architekturen ergab, daß die Verarbeitungsgeschwindigkeit der CONIAN-Architektur bei Operationen mit regulärer Adressierung mit sehr leistungsfähigen bekannten Architekturen vergleichbar ist, während sie bei Operationen, die auf Ausbreitungsprozessen beruhen, diese deutlich übertrifft.

Im Rahmen der Arbeit wurde außerdem die Simulationsgeschwindigkeit der zum Architekturentwurf eingesetzten Methodik bestimmt. Bezogen auf die Architekturkomplexität ergab sich mit $177 \cdot 10^6$ kGatter $\cdot$ Clk pro Sekunde eine Simulationsgeschwindigkeit die um Faktor 10^4 bis 10^5 über der Geschwindigkeit einer ereignisgesteuerten VHDL-Simulation liegt. Gegenüber der Simulationsdauer des Algorithmus, lag die Simulationsdauer der Architektur damit im Mittel um den Faktor 7,7 höher. Das Verhältnis zwischen simulierter Zeit und Dauer für die Architektursimulation betrug im Mittel 1:596. Somit wurde anhand eines Fallbeispiels gezeigt, daß mit der vorgeschlagenen Methodik Modelle mit geringem Modellierungsaufwand erstellt werden können, die hohe Simulationsgeschwindigkeit gestatten. Die vorgeschlagene Simulationsmethodik kann erweitert werden, so daß Architekturen mit von Algorithmus abweichendem zeitlichem Ablauf (Schedule) untersucht werden können, und bietet großes Potential für den Vergleich und die Optimierung von Hardware-Architekturen.

Inhaltsverzeichnis

Vorwort ..... **i**

Zusammenfassung ..... **iii**

Inhaltsverzeichnis ..... **vii**

Abbildungsverzeichnis ..... **xiii**

Tabellenverzeichnis ..... **xv**

Abkürzungsverzeichnis ..... **xvii**

1. Einführung **1**

1.1 Trends bei Multimedia-Diensten 1

1.2 Implementierungsaspekte der Videoobjekt-Segmentierung 4

1.3 Aufgabenstellung 5

1.4 Aufbau der Arbeit 6

2. Algorithmen für Videoobjekt-Segmentierung **9**

2.1 Aufgabenstellung 9

2.2 Bildmerkmale zur Objektabgrenzung 10

2.3 Überblick über Verfahrenstypen 11

2.3.1 Objektabgrenzung 11

2.3.1.1 Punktorientierte Verfahren 11

2.3.1.2 Regionenorientierte Verfahren 12

2.3.1.3 Konturorientierte Verfahren 12

2.3.2 Einsatz mehrerer Bildmerkmale 13

2.3.3 Objektverfolgung 14

2.4 Grundlegende Operationen 14

2.4.1 Klassifikation 14

2.4.2 Nachbarschaftsbasierte Operationen 15

2.4.2.1 Filter-Operationen 15

2.4.2.1.1 Lineare Faltungsfilter 15

2.4.2.1.2 Nichtlineare Filter 18

2.4.2.2 Morphologische Operatoren 20

2.4.2.2.1 Dilatation und Erosion 20

2.4.2.2.2 Morphologische Kantenoperatoren 22

2.4.2.3 Relaxation 23

2.4.3 Bildverknüpfende Operationen 24

2.4.3.1 Bilddifferenz 24

2.4.3.2 Akkumulation der Differenzbeträge 24

2.4.3.3 Bildbereichsersetzung 24

2.4.3.4 Gewichtete Bildaddition 25

2.4.4 Ausbreitungsprozesse 25

2.4.4.1 Segmentzuordnung 25

2.4.4.2 Prüfung und Bearbeitung fragmentierter Segmente 26

2.4.4.3 Vereinigung von Segmentmasken 26

2.4.4.4 Ermittlung von Randpunkten 26

2.4.4.5 Geodesische Distanz 27
2.4.4.6 Geodesisches Skelett 27
2.4.4.7 Wasserscheide-Verfahren 28
2.4.4.8 Levelings 28
2.4.5 Berechnung von Segmenteigenschaften 29
2.4.5.1 Segmentgröße und Umfang 29
2.4.5.2 Umschießendes Rechteck 29
2.4.5.3 Schwerpunkt 29
2.4.5.4 Mittelwert von Luminanz und Chrominanz 29
2.4.5.5 Histogramm 30
2.4.6 Weitere Operationen 30
2.5 Generische Adressierungsarten 30
2.5.1 Nachbarschaftssysteme 32
2.5.2 Randpunkte 33
2.5.3 Scan-Modus 33
2.5.4 Räumlicher Zusammenhang und Rekursivität 34
2.5.5 LIFO- und FIFO-Adressierung 34
2.5.6 Zeitpunkt der Verarbeitung 35
2.5.7 Bestimmung der Startpunkte 36
2.5.8 Hierarchische Queues 36
2.5.9 Änderung der Hierarchiestufe 37
2.5.10 Segmentindizierte Adressierung 38
2.6 Bildformat und Algorithmenkomplexität 38
3. Architekturen für Videoobjekt-Segmentierung 41
3.1 Algorithmische Anforderungen und Randbedingungen 41
3.2 Bekannte Architekturkonzepte 42
3.2.1 Operationsausführung 42
3.2.1.1 Datenabhängigkeit und Parallelität 42
3.2.1.2 Implementierung von Parallelität 43
3.2.1.3 Ahmdahlsches Gesetz 43
3.2.1.4 Parallelitätsebenen 44
3.2.2 Datenorganisation 45
3.2.2.1 Speicherelemente und Zugriffseigenschaften 45
3.2.2.2 Speicherhierarchie 47
3.2.2.3 Arten von Pufferspeichern 48
3.2.2.4 Paralleler Datenzugriff 49
3.2.3 Flexibilität 49
3.2.3.1 Programmierbarkeit und Konfigurierbarkeit 50
3.2.3.2 Zusatzaufwand für Flexibilität 51
3.2.3.3 Nutzungsmöglichkeiten 51
3.3 Bekannte Architekturen 52
3.3.1 Klassifikation 52
3.3.2 Universelle Architekturen 53
3.3.2.1 Prozessoren und Prozessorsysteme 53
3.3.2.1.1 Klassifikation von Prozessorarchitekturen 54
3.3.2.1.2 Prozessoren für Videosignalverarbeitung 57
3.3.2.2 Konfigurierbare Systeme 60
3.3.3 Problemangepaßte Architekturen 63

3.3.3.1 Datenparallele problemangepaßte Architekturen ... 63
3.3.3.1.1 Massiv parallele Architekturen ... 64
3.3.3.1.2 Architekturen mit reduzierter Array-Größe ... 66
3.3.3.2 Befehlsparallele problemangepaßte Architekturen ... 68
3.3.3.2.1 Architekturen für reguläre Bildoperationen ... 68
3.3.3.2.2 Architekturen für Ausbreitungsprozesse und Segmentoperationen ... 72
3.3.4 Spezialisierte Architekturen ... 73
3.3.5 Weitere Ansätze ... 74
3.4 Zusammenfassung ... 75
4. Konzeption einer flexiblen Architektur ... 77
4.1 Grundkonzept ... 77
4.1.1 Adressierung ... 78
4.1.2 Verarbeitung ... 79
4.1.3 Registermatrix ... 80
4.2 Parallelisierungsstrategie ... 81
4.2.1 Adressierungs- und Verarbeitungsebene ... 81
4.2.2 Bildpunktebene ... 82
4.2.3 Ebene der Bild- / Segmentoperation ... 83
4.3 Datenorganisation ... 83
4.3.1 Bilddaten ... 83
4.3.2 Segmentindizierte Daten ... 85
4.4 Steuerungskonzept ... 85
4.5 Systemkonzept ... 86
5. Entwurfsmethodik ... 89
5.1 Anforderungen beim Architekturentwurf für CONIAN ... 89
5.2 Entwurf digitaler HW-Architekturen und HW/SW-Systeme ... 90
5.2.1 Typischer Entwurfsablauf ... 90
5.2.2 Entwurfswerkzeuge ... 91
5.2.3 Entwurfsproblematik ... 92
5.3 Weiterentwickelte Entwurfstechniken ... 93
5.3.1 Syntheseverfahren ... 93
5.3.2 Hardware/Software-Co-Design ... 94
5.3.2.1 Hardware/Software-System-Spezifikation ... 94
5.3.2.2 Hardware/Software-Partitionierung ... 95
5.3.2.3 Hardware/Software-Co-Simulation ... 96
5.3.2.4 Hardware/Software-Co-Synthese ... 96
5.3.2.5 Hardware/Software-Co-Verifikation ... 96
5.3.3 Komponentenbasierte Entwurfsmethoden ... 96
5.3.3.1 Bibliotheksbasierte Entwurfsmethoden ... 96
5.3.3.2 IP-basierte Entwurfsmethoden ... 97
5.3.4 Simulationsverfahren ... 98
5.3.4.1 Beschleunigte Hardware-Simulation ... 98
5.3.4.2 Simulation auf Systemebene ... 99
5.4 Neue Simulationsmethodik ... 100
5.4.1 Architekturalternativen ... 101

5.4.2 Funktionale und zeitliche Aspekte der Simulation ... 101
5.4.3 Simulation architektureller Aspekte ... 102
5.4.4 Trennung der Simulationsmodelle ... 103
5.4.5 Kopplung Architekturmodell - Software-Algorithmenmodell ... 104
5.4.6 Operationskopplung und Granularität ... 105
5.4.7 Datenkopplung ... 107
5.4.8 Kopplung von Betriebsarten ... 108
5.4.9 Getrennte vs. integrierte Simulation ... 109
5.4.10 Architekturelemente und Verschaltung ... 110
5.5 Simulationsmodell für CONIAN ... 112
5.5.1 Algorithmenmodell und AddressLib ... 112
5.5.2 Kopplung von Algorithmus und Architektur ... 113
5.5.2.1 Kopplung von Betriebsmodi ... 113
5.5.2.2 Kopplung der Operationsausführung ... 114
5.5.2.3 Kopplung von Datenzugriffen ... 116
5.5.2.4 Referenzereignisse und Sektionen ... 116
5.5.2.5 Format der Aufzeichnung ... 116
5.5.3 Architekturmodelle ... 117
5.5.3.1 Eingesetzte Architekturelemente ... 117
5.5.3.2 Auswertung der Architektursimulation ... 118
6. Architekturentwurf ... 119
6.1 Ergebnisse der Architekturuntersuchung ... 119
6.1.1 Operationsablauf ... 119
6.1.1.1 Pipelining der Verarbeitung ... 121
6.1.1.2 Segment-Adressierung ... 121
6.1.2 Speicherschnittstelle ... 122
6.1.2.1 Bildspeicher ... 123
6.1.2.1.1 Konzepte für die Organisation der Bilddaten ... 123
6.1.2.1.2 Konzepte zur Pufferung von Bilddaten ... 127
6.1.2.2 Speicher für Segmentindizierte Daten / Histogramme ... 140
6.1.2.2.1 Konzepte für die Pufferung segmentindizierter Daten ... 140
6.1.2.3 Koordinatenspeicher ... 142
6.1.2.3.1 Charakterisierung der Zugriffe ... 142
6.1.2.3.2 Zeitliche Anforderungen ... 142
6.1.2.3.3 Größe des Koordinatenspeichers ... 143
6.1.2.4 Speicherarchitektur ... 144
6.1.3 Architekturvorschlag ... 147
6.2 Implementierung der vorgeschlagenen Architektur ... 147
6.2.1 Steuerungseinheit ... 147
6.2.2 Verarbeitungseinheit ... 148
6.2.2.1 Blockstruktur ... 148
6.2.2.1.1 Block für Intra-Operationen ... 150
6.2.2.1.2 Block für Inter-Operationen ... 150
6.2.2.1.3 Block für Segment-Operationen ... 151
6.2.2.1.4 Block für Segmentindizierte Operationen ... 151
6.2.2.2 Datenfluß ... 151
6.2.3 Adreßeinheit ... 151
6.2.3.1 Organisation des externen Speichers ... 151

6.2.3.2 Organisation des Pufferspeichers ... 153
6.2.3.3 Blockstruktur ... 154
6.2.3.3.1 Cache-Speicher, Cache-Controller und Tag-RAM ... 154
6.2.3.3.2 Transfer-Controller ... 156
6.2.3.3.3 Prefetch-Controller ... 157
6.2.3.3.4 SI-Controller ... 158
6.2.3.3.5 Arbiter ... 158

7. Ergebnisse und Bewertung ... 159

7.1 Ergebnisse für die CONIAN-Architektur ... 159
7.1.1 Flächenbedarf ... 159
7.1.2 Verarbeitungsdauer der Bildpunktoperationen ... 161
7.1.3 Systemgeschwindigkeit ... 162
7.1.3.1 Benchmark-Operationen ... 162
7.1.3.2 Farbsegmentierungsalgorithmus ... 163
7.2 Vergleich und Bewertung ... 165
7.2.1 Vergleich mit Software-Implementierung ... 165
7.2.2 Technologieskalierung ... 166
7.2.3 Vergleich mit anderen Architekturen ... 168
7.3 Bewertung der Simulationsmethodik ... 171
7.3.1 Simulationsgeschwindigkeit ... 171
7.3.2 Einsatzbereich ... 172

8. Ausblick ... 175

Anhang ... I

A. Organisation der segmentindizierten Daten ... I

B. Ablauf der Bildpunktoperationen ... II

C. Verarbeitungsdauer verschiedener Bildpunktoperationen ... IV

D. Zugriffshäufigkeit bei unterschiedlicher Speicherschnittstelle ... V

E. Vergleich von Zeilenspeicher und Prefetching ... VI

F. Platzierungsschemata ... VIII

Literaturverzeichnis ... IX

Abbildungsverzeichnis

1.1 Segmentiertes Bild mit drei hervorgehobenen anclickbaren Objekten 2
1.2 Anwendungsbeispiel virtuelle Modenschau 2
2.1 Filtermatrix und Bildausschnitt eines linearen 3x3 Filters 16
2.2 Gaußfilter und 121-Tiefpaß-Filter 17
2.3 Horizontales, vertikales und bidirektionales Interpolations-Filter 17
2.4 Kantenoperatoren erster Ableitung 18
2.5 Laplace-Filter 18
2.6 Bsp: Anwendung des Medianfilters 19
2.7 Bsp: Anwendung des Extremwertoperators 19
2.8 Schwellwertbildung mit einer/zwei Schwellen 20
2.9 Sättigung mit unterer und/oder oberer Grenze 20
2.10 Bsp: Binäre Dilatation und Erosion 21
2.11 Bsp: Grauwert Dilatation und Erosion 22
2.12 Bsp: Anwendung horizontaler morphologischer Kantenoperatoren 22
2.13 Günstiger und ungünstiger Verlauf der Segmentgrenze 23
2.14 Vereinigung zweier Segmentmasken 26
2.15 Geodesische Distanz 27
2.16 Bsp: Ablauf des Wasserscheide-Verfahren 28
2.17 Umschließendes Rechteck 29
2.18 Schematische Übersicht über generische Adressierungsarten 32
2.19 Ein- und zweidimensionale Nachbarschaftssysteme 32
2.20 Übersicht über wichtige Scan-Modi 33
2.21 Problematik der Prüfung auf Zugehörigkeit zur selben Fläche 34
2.22 Verarbeitungsreihenfolge bei (a) FIFO- und (b) LIFO-Adressierung. 35
2.23 Verarbeitungsreihenfolge in Abhängigkeit der Art der Startpunktsuche 36
2.24 Verarbeitungsreihenfolge beim Einsatz hierarchischer Queues 37
3.1 Das Prinzip der Speicherhierarchie 48
3.2 Mehrfache Nutzung von Bildpunkten mittels Schiebe-Operationen 48
3.3 Schematischer Aufbau eines FPGAs 50
3.4 Schematischer Aufbau eines Prozessors 51
3.5 Klassifikation von Architekturen für die Videoobjekt-Segmentierung 53
3.6 Blockdiagramm des TMS320C62x 59
3.7 Blockdiagramm des HiPAR 60
3.8 Blockdiagramm des Splash-2 Systems 61
3.9 Blockdiagramm des Sonic-1 Systems 62
3.10 Blockdiagramm des Reconfigurable Image Coprocessor 68
3.11 Blockdiagramm der PIMM-1 Architektur 69
3.12 Blockdiagramm der (a) MoM-PDA Architektur und des (b) Kress Arrays 70
3.13 Aufbau der Architektur [NoMe95] 72
4.1 Operation der Registermatrix 80
4.2 Parallelisierung auf Bildpunktebene 82
4.3 Blockbild des vorgeschlagenen Architekturkonzeptes 86
4.4 Blockbild des Systemkonzeptes 87
5.1 Typischer Ablauf beim Entwurf digitaler HW/SW-Systeme 91
5.2 Eingangsdaten (links) und Ergebnisse (rechts) der Architektursynthese 94
5.3 Ablauf der ereignisgesteuerten Simulation 98
5.4 Prinzip der zyklusbasierten Simulation 99
5.5 Funktionaler und zeitlicher Aspekt von Operationen 102

5.6 Architektureller Aspekt von Operationen 103
5.7 (a) Zuordnung part. Algorithmenteile (b) Kopplung logischer Abschnitte 106
5.8 Granularität und Operationsgruppierung 107
5.9 Kopplung von Array-Zugriffen in einem selektierten Bereich 108
5.10 Algorithmen- und Architekturmodell bei getrennter Simulation 109
5.11 Algorithmen- und Architekturmodell bei integrierter Simulation 110
5.12 Beispiele generischer Architekturelemente für Funktionseinheiten 111
5.13 Gegenüberstellung von (a) Architekturaufbau und (b) Architekturmodell 112
5.14 Aufbau des Algorithmenmodells und der AddressLib 113
5.15 Kopplung der Operationsausführung bei Segment-Adressierung 115
5.16 Aufbau eines der Architekturmodelle für CONIAN 117
6.1 Überlappender Ablauf der Teiloperationen einer Bildpunktoperation 120
6.2 Auslastung der Verarbeitungseinheit 121
6.3 Speicheranordnung für parallelen Zugriff auf mehrere Bildpunkte 124
6.4 Bankzugehörigkeit für parallelen Zugriff auf 125
6.5 Anordnung der Komponenten im Speicher 126
6.6 Puffer für reguläre Zugriffe: Zeilenspeicher mit Lese- und Schreibpuffer 128
6.7 Anzahl gleichzeitig zugegriffener Cache-Blöcke bei CON4 und CON8 130
6.8 (a) Direktabgebildeter und (b) 2-fach assoziativer Cache 131
6.9 Schreibstrategien „write around“ (a) „write through“ (b) „write back“ (c) 132
6.10 Zeitpunkte bei horizontaler zeilenweiser Verarbeitungsabfolge, ... 138
6.11 Bei horizontaler Verarbeitungsabfolge benötigte Anzahl an Blöcken 138
6.12 Verwendung des Eingangsregisters zur Pufferung segmentindizierter Daten 140
6.13 Häufigkeit der Koordinatenspeicherzugriffe bei CON4 (li.) bzw. CON8 (re.) 142
6.14 Verarbeitungsdauer der Speicherarchitekturvarianten 145
6.15 Entscheidungsbaum Speicherarchitektur 146
6.16 Zerlegung einer Intra-Operationen in Teiloperationen 149
6.17 Aufbau des Blocks für Intra-Operationen 150
6.18 Blockanordnung innerhalb einer Speicherseite 152
6.19 Zusammenhang zwischen Bildkoordinaten und Adressen 153
6.20 Aufbau der Adreßeinheit und Einbettung ins Gesamtsystem 154
6.21 Pipeline-Struktur des Cache-Controllers 155
6.22 Über- und Unterabtastung von Bilddaten 157
7.1 Abhängigkeit der Anzahl an Takten vom Optimierungsgrad 162
7.2 Unterschiedliche Verarbeitung der zu segmentierenden Bilder 163
7.3 Taktzyklen pro Bild aufgeschlüsselt nach verwendeter Adressierungsart 164
7.4 Taktzyklen pro Bild verglichen für verschiedene Sequenzen 164
7.5 Verarbeitungszeit verschiedener Architekturen 171
B.1 Ablauf einer Bildpunktoperation bei direktem Zugriff auf den Bildspeicher III
B.2 Ablauf einer Bildpunktoperation bei Intra-Adressierung III
B.3 Ablauf einer Bildpunktoperation bei Inter-Adressierung III
B.4 Ablauf einer Bildpunktoperation bei Segment-Adressierung III
F.1 Plazierungsschema bei Verwaltung von drei Bildern VIII
F.2 Plazierungsschema bei Verwaltung von zwei Bildern VIII
F.3 Plazierungsschema bei Verwaltung von einem Bild VIII

Tabellenverzeichnis

2.1 Anwendungsbereiche von Filteroperationen 15
2.2 Zugrundeliegende Adressierungsarten von Low-Level-Operationen 31
2.3 In der Videocodierung gebräuchliche Bildformate und Bildfrequenzen 39
2.4 Anzahl der Bildpunktoperationen bei verschiedenen Sequenzen 39
2.5 Anzahl der Bildpunktoperationen bei verschiedenen Bildformaten 39
2.6 Befehlshäufigkeit für einen iterativen Farbsegmentierungsalgorithmus 40
3.1 Klassifikation nach Flynn 43
3.2 Vergleich von Latch, Register und Speicherzellen 46
3.3 Parallelitätsarten und Prozessorarchitekturen 54
4.1 Übersicht über mögliche Verarbeitungszustände 84
4.2 Übersicht über die pro Bildpunkt gespeicherten Komponenten 84
6.1 Dauer der Operation „Segment markieren“ je nach durchlaufenem Zweig 122
6.2 Busbreite für bildpunkt- bzw. komponentenorientierten Zugriff 127
6.3 Relative Häufigkeit von Lese- und Schreibzugriffen 127
6.4 Zusätzlicher Zeitbedarf beim Einsatz eines Zeilenspeichers 129
6.5 Hit-Raten für einen direktabgebildeten Cache (implizite Verdrängung) 133
6.6 Hit-Raten für einen 4-fach assoziativen Cache (zufällige Verdrängung) 133
6.7 Vergleich der Hit-Raten von QCIF- und CIF-Bildsequenzen 134
6.8 Dauer tR eines Blockzugriffes auf SDRAM Hintergrundspeicher 135
6.9 Zugriffsdauer pro Bildpunktoperation für externen Cache, ohne Schreibpuffer 136
6.10 Zugriffsdauer pro Bildpunktoperation für internen Cache, ohne Schreibpuffer 136
6.11 Mittlere Anzahl an Takten für den Schreibzugriff auf den Hintergrundspeicher 136
6.12 Zugriffsdauer pro Bildpunktoperation für internen Cache, mit Schreibpuffer 136
6.13 Vergleich des Overheads von Zeilenspeicher und Cache mit Prefetching 139
6.14 Hit-Raten des Pufferregisters für segmentindizierte Daten 140
6.15 Hit-Rate des Cache für segmentindizierte Daten je nach Größe 141
6.16 Vergleich der Hit-Raten des Cache für segmentindizierte Daten 141
6.17 Mittlere Anzahl an Wartezyklen beim Zugriff auf den Koordinatenspeicher 143
6.18 Vergleich von Struktur und Schnittstellen der Verarbeitung 148
6.19 Speicherbedarf für Bilddaten und segmentindizierte Daten 152
6.20 Speicherorganisation in Abhängigkeit der SDRAM Größe 153
7.1 Logik und Verdrahtungsfläche der CONIAN-Architektur 159
7.2 Größe und Flächenbedarf der chipinternen Speicher 160
7.3 Größe und Flächenbedarf bei verschiedenen Bildformaten 160
7.4 Verarbeitungsdauer ausgewählter Operationen und korrespondierende Taktanzahl 161
7.5 Dauer ausgewählter Benchmark-Operationen auf der CONIAN-Architektur 163
7.6 Taktzyklen pro Bild unterteilt nach Art der Verarbeitung des Bildes 165
7.7 Vergleich der Verarbeitungsdauer des CONIAN mit dem Celeron / Pentium IV 166
7.8 Übersicht über die Implementierungsdaten 169
7.9 Verarbeitungsleistung verschiedener Architekturen 170
7.10 Simulierte Zeit und Simulationsdauer für Algorithmus und Architektur 172
A.1 Komponenten eines segmentindizierten Datensatzes I
A.2 Aufteilungen der segmentindizierten Daten in Bündel II
C.1 Verarbeitungsdauer verschiedener Bildpunktoperationen IV
D.1 Relative Anzahl der Lesezugriffe bei unterschiedlicher Speicherschnittstelle V
D.2 Relative Anzahl der Schreibzugriffe bei unterschiedlicher Speicherschnittstelle V
E.1 Initialisierungs- und Abschlußdauer bei Einsatz eines Cache mit Prefetching VI
E.2 Vergleich der Initialisierungs- und Abschlußdauer VII

Abkürzungsverzeichnis

ALU	*Arithmetic Logic Unit*; eine Verarbeitungseinheit, die arithmetische und logische Operationen unterstützt.
ASIC	*Application Specific Integrated Circuit*; eine Integrierte Schaltung, die eine Spezielle Anwendung implementiert.
CAM	*Content Addressable Memory*; ein Speicher, dessen Speicherzellen über den enthaltenen Inhalt adressiert werden.
CCIR 601	*Comité Consultatif International des radio Communications - Empfehlung 601*; Bildformat mit 720x576 Bildpunkten.
CIF	*Common Interchange Format*; Bildformat mit 352x288 Bildpunkten.
CMOS	*Complementary Metal Oxide Semiconductor*; dominierende Halbleitertechnologie für Speicher und Logikschaltungen.
CNN	*Cellular Neural Network*; ein massiv paralleles n-dimensionale Paradigma für Berechnungen. Dabei handelt es sich um ein reguläres Array aus Zellen mit mehreren Eingängen, Verbindungen zu lokalen Nachbarn, einem Ausgang und einer Zustandsgröße, dessen Dynamik kontinuierlich oder zeitdiskret sein kann.
CONIAN	*Configurable Image Analysis Coprocessor*; Name der in dieser Arbeit vorgeschlagenen Architektur.
CON0, etc.	Bezeichnung für Nachbarschaften von Bildpunkten, siehe Bild 2.19.
CPU	*Central Processing Unit*; Verarbeitungseinheit eines Prozessors.
DRAM	*Dynamic Random Access Memory*; wahlfrei zugreifbarer Halbleiterspeicher, dessen Dateninhalt regelmäßig aufgefrischt werden muß.
DRL	*Dynamic Reconfigurable Logic*; Konfigurierbare Einheit, deren Konfiguration während des Betriebs geändert werden kann.
DSP	*Digital Signal Processor*; auf Signalverarbeitung spezialisierter Prozessortyp.
EDA	*Electronic Design Automation*.
EEPROM	*Electrically Erasable Programable Read Only Memory*; elektrisch lösch- und programmierbarer Festwertspeicher.
EPROM	*Erasable Programable Read Only Memory*; lösch- und programmierbarer Festwertspeicher.
eRAM	*Embedded Random Access Memory*; wahlfrei zugreifbarer Speicher, der zusammen mit weiteren Einheiten eines Systems auf einem Chip integriert ist.
FIFO	*First In First Out*; das zuerst gespeicherte Datum wird auch zuerst wieder ausgelesen.
FIR	*Finite Impulse Response*; ein einzelner Impuls am Eingang erzeugt ein zeitlich begrenztes Ausgangssignal.
FPGA	*Field-Programmable Gate Array*; ein Chip, der aus konfigurierbaren Logikzellen aufgebaut ist.
FPU	*Floating Point Unit*; Verarbeitungseinheit für Gleitkomma-Operationen.
FSM	*Finite State Machine*; ein Automat, dessen Verhalten durch Zustände und Zustandsübergänge beschrieben wird.
HW	*Hardware*.

IEEE	*Institute of Electrical and Electronics Engineers*.
IIR	*Infinite Impulse Response*; ein einzelner Impuls am Eingang erzeugt ein Ausgangssignal unendlicher Dauer.
ILP	*Instruction Level Parallelism*; Möglichkeit zur parallelen Ausführung aufeinanderfolgender Befehle eines Programmes.
IP	*Intellectual Property.*
ISO	*International Organisation for Standardization.*
ISS	*Instruction Set Simulator*; ein Prozessorsimulator, der die Auswirkungen der durchgeführten Befehle auf Ebene des Programmiermodells simuliert.
ITU	*International Telecommunication Union.*
KPN	*Kahn Prozeß Netzwerk*; Eine datenflußorientierte Modellierungsform, die es gestattet Algorithmen durch nebenläufige kommunizierende Prozesse zu beschreiben.
LIFO	*Last In First Out*; das letzt gespeicherte Datum wird zuerst ausgelesen.
LRU	*Least Recently Used*; Strategie zur Ersetzung von Blöcken in Caches.
MAC	*Multiply Accumulate*; Multiplikation mit nachfolgender Summierung.
MSE	*Mean Square Error*; mittler quadratischer Fehler.
MIMD	*Multiple Instruction Multiple Data*; Konzept zur parallelen Operationsausführung, wobei unterschiedliche Operationen mit unterschiedlichen Daten parallel ausgeführt werden.
MISD	*Multiple Instruction Single Data*; Operationsausführung in Form einer Pipeline.
MMX	*Multimedia Extension*; Einheit die x86 Prozessoren um Sub-Wort-Parallelität erweitert.
MPEG	*Moving Pictures Expert Group*; Arbeitsgruppe innerhalb der ISO bzw. gebräuchlicher Name für die von ihr entwickelten Standards.
OMI	*Open Model Interface*; Standardisierte Schnittstelle zwischen Modellen und Simmulationswerkzeugen.
OTP	*One Time Programmable*.
PE	*Prozessorelement.*
PQI	*Process on Queue In*; die Verarbeitung eines Punktes erfolgt, wenn seine Position im Koordinatenspeicher abgelegt wird.
PQO	*Process on Queue Out*; die Verarbeitung eines Punktes erfolgt, wenn seine Position aus dem Koordinatenspeicher entnommen wird.
QCIF	*Quarter Common Interchange Format*; Bildformat mit 176x144 Bildpunkten.
RAM	*Random Access Memory*; wahlfrei zugreifbarer Speicher.
RGB	*Red Green Blue*; Bezeichnung eines Farbraumes der aus Rot-, Grün- und Blaukomponenten besteht.
RISC	*Reduced Instruction Set Computer.*
RTL	*Register-Transfer Level*; Beschreibungsform bei der Hardware-Module durch Register und dazwischenliegende logische Operationen dargestellt werden.
RSST	*Rekursive Shortest Spanning Tree*; Segmentierungsverfahren das auf wiederholter

	Zusammenfassung möglichst ähnlicher Segmente beruht.
SAD	*Sum of Absolute Distances*; Summe der Absolutbeträge, wird beispielsweise zur Bewegungsschätzung eingesetzt.
SDL	*Specification and Description Language*; eine Sprache die zur Beschreibung von Kommunikationsprotokollen eingesetzt wird.
SDRAM	Synchronous *Dynamic Random Access Memory*; DRAM mit synchroner Schnittstelle.
SI	*Segment Indiziert*; siehe Kapitel 2.5.10.
SIF	*Standard Interchange Format*; Bildformat mit 352x240 (USA, Japan) bzw. 352x288 (Europa) Bildpunkten.
SIMD	*Single Instruction Multiple Data*; Konzept zur parallelen Operationsausführung, wobei dieselbe Operation parallel mit unterschiedlichen Daten ausgeführt wird.
SISD	*Single Instruction Single Data*; sequentielle Operationsausführung.
SMT	*Simultaneous Multi Threading*; eine Prozessorarchitektur, bei der mehrere Thread gleichzeitig ausgeführt werden.
SQCIF	*Sub Quarter Common Interchange Format*; Bildformat mit 128x96 Bildpunkten.
SRAM	*Static Random Access Memory*; wahlfrei zugreifbarer Halbleiterspeicher, der Daten speichert ohne daß diese aufgefrischt werden müssen.
SRL	*Static Reconfigurable Logic*; Konfigurierbare Einheit, deren Konfiguration lediglich vor dem Betrieb geändert werden kann.
SW	*Software*.
TLP	*Task Level Parallelism*; Möglichkeit zur parallelen Ausführung unabhängiger Befehlsfolgen an unterschiedlichen Stellen eines Programmes.
VHDL	*Very High Speed Integrated Circuits Hardware Description Language*; Sprache zur Beschreibung von Hardware für Integrierte Schaltungen.
VIS	*Visual Instruction Set*; Befehlssatz bzw. Einheit die Sparc Prozessoren um Sub-Wort-Parallelität erweitert.
VLD	*Varable Length Decoding*; Dekodierung eines Stromes an Symbolen mit unterschiedlicher Länge.
VLIW	*Very Long Instruction Word*; Prozessorkonzept zur Steuerung parallel operierender Funktionseinheiten durch ein langes Befehlswort.
VRAM	*Video Random Access Memory*; Halbleiterspeicher, der einen zusätzlichen seriellen Ausgang für Videoausgabe beinhaltet.
VSIA	*Virtual Socket Interface Alliance*.
VSP	*Video Signal Processor*; auf Videosignalverarbeitung spezialisierter Prozessortyp.
WWW	*World Wide Web*; ein Dienst im Rahmen des Internets.
YUV	Bezeichnung eines in der Videocodierung gebräuchlichen Farbraumes der aus der Helligkeitskomponente Y und den Farbkomponenten U und V besteht.
1D, 2D, 3D	*ein-, zwei-, dreidimensional.*

1. Einführung

1.1 Trends bei Multimedia-Diensten

Multimedia-Dienste, die mehrere unterschiedliche Darstellungsformen von Information wie Sprache, Musik, Text, Graphik, Stand- und Bewegtbilder miteinander kombinieren, haben in den letzten Jahren eine stürmische Entwicklung durchlaufen. Durch die rasante Verbreitung des Internets hat sich im Rahmen des World Wide Web (WWW) die Integration von Text, Standbildern und (teilweise animierten) Graphiken bereits vollzogen, während sich die Integration dynamischer Darstellungsformen wie Sprache, Musik und Bewegtbilder wegen teilweise ungelöster technischer Probleme oder wegen zu hoher Kosten derzeit erst in der Anfangsphase befindet. Dabei sind dynamische Darstellungsformen und insbesondere graphische Elemente und Bewegtbilder sehr attraktiv, da sie mehr Aufmerksamkeit erregen und sich komplexe Zusammenhänge und Abläufe anschaulich und schnell durch Videosequenzen vermitteln lassen.

Videodienste sind erst durch die rasante Entwicklung der Mikroelektronik und Computertechnologie möglich geworden, da Videodaten im Vergleich zu anderen Darstellungsformen von Information und im Vergleich zu derzeit üblichen Speicherkapazitäten eine große Datenmenge darstellen, und da erst mit der gestiegenen Verfügbarkeit von Kompressionstechniken für Videodaten eine effiziente Speicherung und Übertragung möglich wurde. Im Bereich der Videokompression haben standardisierte Verfahren eine große Bedeutung, da diese die Interoperabilität zwischen Produkten verschiedener Hersteller gewährleisten, was insbesondere im Kommunikationsbereich wichtig ist. Hierbei sind zum einen die Standards MPEG-1, MPEG-2 und MPEG-4 der ISO/IEC zu nennen, die in den Bereichen Computer, digitales Fernsehen und Internet weit verbreitet sind, als auch die Standards H.261 und H.263 der ITU-T, die im Bereich Videokommunikation verbreitet sind. Ein wichtiger Trend, der über die Entwicklung der genannten Standards hinweg besteht, ist die Steigerung der Kodiereffizienz der eingesetzten Verfahren, so daß bei gleicher Bildqualität eine kleinere Datenmenge für die Codierung ausreicht. Ausgangspunkt für die Entwicklung von MPEG-4 [MPEG4] war die Identifizierung von Anforderungen, die durch vorige Standards nicht erfüllt werden. Neben anderen Anforderungen wurde objektorientierter Zugriff und Bearbeitung in Verbindung mit gleichzeitiger Codierung mehrerer Objekte als zentrale Anforderung identifiziert. Dies bedeutet, daß unterschiedliche Objekte getrennt voneinander kodiert bzw. verarbeitet werden und die Zusammensetzung (Komposition) zu einer Szene erst bei der Anzeige erfolgt.

Videocodierung

Zukünftige interaktive Anwendungen, die aus passiven Betrachtern aktive Teilnehmer machen werden, werden durch objektbasierte Funktionalität geprägt sein. Beispielsweise können virtuelle Touren durch eine Firma, einen Vergnügungspark, eine Hotelanlage, etc. auf diese Weise implementiert werden, bei der der Benutzer durch anklicken der für ihn interessanten Objekte im Videofilm verzweigen kann, so daß er sich eine Tour nach individuellen Interessen zusammenstellen kann. Die Funktionalität kann prinzipiell beliebig sein, beispielsweise können beim Klikken auf gezeigte Objekte zusätzliche Informationen angezeigt werden, oder ein Link in das Internet aktiviert werden. In einer virtuelle Modenschau kann dies eingesetzt werden, um für die gezeigten Kleidungsstücke Informationen wie Lieferbarkeit, lieferbare Farben und Preis anzuzeigen (siehe Bild 1.1 und 1.2). Da die Objektform in den kodierten Daten enthalten ist, lassen sich nach der Komposition zu einer Szene die enthaltenen Objekte gegeneinander abgrenzen. Objekte in einer Szene können so mit Funktionalität hinterlegt werden, die z.B. durch anklicken aktiviert werden kann.

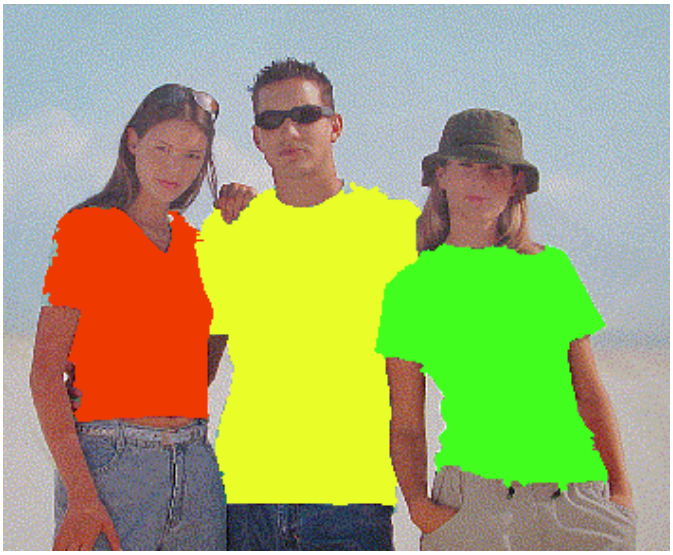


Bild 1.1: Segmentiertes Bild mit drei hervorgehobenen anclickbaren Objekten

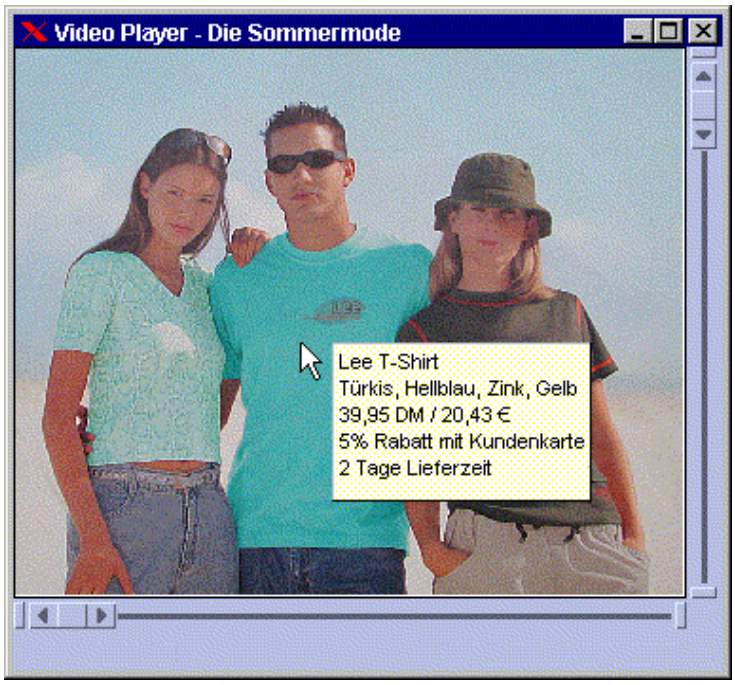


Bild 1.2: Anwendungsbeispiel virtuelle Modenschau

Auch in nicht interaktiven Anwendungen wird objektbasierte Funktionalität eine wichtige Rolle spielen. Durch die Trennung von Objekten und Hintergrund lassen sich diese bei der Kodierung entsprechend ihrer Relevanz behandeln. Bei Person-Hintergrund Szenen ist beispielsweise der Hintergrund i.d.R. für den Betrachter der Szene von geringerer Relevanz und wird meist nicht beachtet. Trotzdem kann der Hintergrund sehr komplex sein, z.B. wenn sich viele Gegenstände dort befinden, bei Bewegung durch andere Personen, bei Aufnahmen in einem Auto oder anderem Verkehrsmittel, oder bei Außenaufnahmen. Eine Verbesserung der Kodiereffizienz bzw. der subjektiven Qualität kann erreicht werden, indem die verfügbare Datenmenge in überproportionaler Weise für die relevanten Objekte verwendet wird. Ein Beispiel hierfür sind Überwachungs-

anwendungen, bei denen die Datenmenge klein gehalten werden muß, da die Daten zum einen aufgezeichnet werden und zum anderen meist über eine gemeinsame Leitung zu einer Zentrale übertragen werden müssen. Bei Anwendungen, bei denen Daten über mobile Kanäle übertragen werden, ist es möglich, Bits für den Fehlerschutz von Objekten mit geringer Relevanz einzusparen.

Objektbasierte Funktionalität kann auch dazu genutzt werden, um bei der Komposition Objekte aus verschiedenen Quellen zu mischen. Insbesondere in der Videokommunikation kann dies genutzt werden, um den Hintergrund auszublenden, so daß beispielsweise in einem Call-Center entsprechend der gewählten Nummer des jeweiligen Anrufer ein wechselnder virtueller Hintergrund angezeigt wird, oder um mehrere Teilnehmer einer virtuellen Videokonferenz in einem virtuellen Raum zusammenzubringen. Und nicht zuletzt erlaubt objektbasierte Funktionalität inhaltsbasierte Manipulierung z.B. für die Produktion von Multimedia-Material.

Mit einer Kamera aufgenommenes Videomaterial besteht aus einer Abfolge rechteckiger Bilder, und kann nicht direkt verwendet werden, um objektbasierte Funktionalität zur Verfügung zu stellen, da die im Videomaterial enthaltenen Objekte nicht voneinander getrennt sind. Daher müssen zunächst Objekte aus dem Videomaterial mittels Videosegmentierung erzeugt werden, d.h. die in den Videodaten enthaltenen Objekte werden gegeneinander und gegenüber dem Hintergrund abgegrenzt.

Multimedia-Datenbanken

Neben dem bei Multimedia-Diensten vorherrschenden Trend zu objektbasierter Funktionalität, spielt objektbasierte Verarbeitung auch bei der Suche von Multimedia-Daten eine Rolle. Automatische Suchwerkzeuge gewinnen zunehmend an Bedeutung, denn immer mehr Information wird in digitaler Form verfügbar, verstreut über verschiedenste Orte digitale Bibliotheken, Filmarchive, Firmendatenbanken, etc. Die wachsende Zahl an Multimedia-Daten und die weltweite Zugänglichkeit großer verteilter Datenmengen durch das Internet, vergrößert zwar einerseits das Informationsangebot, erschwert aber andererseits das Auffinden gesuchter Daten. Mit der wachsenden Datenflut ist es von grundlegender Bedeutung, eine gezielte und effiziente Suche nicht nur in Textdaten, sondern auch in Multimedia-Daten durchführen zu können.

Als die ISO im Jahre 1997 auf diese Problematik aufmerksam wurde, startete sie die Arbeit an dem MPEG-7 Standard [MPEG7], der diese Problematik adressiert. Als Ansatz wurde dabei gewählt, ein Format für Metadaten zu standardisieren, das es ermöglicht Inhalt, Struktur und Eigenschaften der zu durchsuchenden Mediadaten in kompakter Form beschreiben, und eine schnelle Suche in großen Datenmengen durchzuführen. Als Ergebnis der Aktivitäten soll im Oktober 2001 der MEPG-7 Standard als Internationaler Standard verabschiedet werden. Die Standardisierung beschränkt sich hierbei auf die Deskriptoren und deren Codierung, Methoden zur Generierung bzw. Extraktion der Metadaten und die Durchführung der Suche werden nicht festgelegt, da dies für Interoperabilität nicht erforderlich ist und da durch die Möglichkeit des Einsatzes verschiedenster Methoden Wettbewerb zur Etablierung der besten Methoden führt. Neben der Extraktion und Suche läßt sich MPEG-7 für weitere grundlegende Anwendungstypen, wie die Transkodierung von Mediadaten (z.B. die Generierung von Zusammenfassungen) und die Filterung von Metadaten einsetzen [HeNi01].

Bei der Erzeugung der Metadaten ist die Kenntnis der im Videomaterial enthaltenen Objekte Voraussetzung für die Formbeschreibung durch den Simple Shape Deskriptor, die es ermöglicht, gezielt nach Objekten mit einer bestimmten Form zu suchen. Darüberhinaus erlaubt die Kenntnis der Objektformen, die Zuordnung von Deskriptoren zu Objekten, so daß es beispielsweise möglich ist nach Objekten mit bestimmten Farben zu suchen, bzw. nach Objekten, die sich auf eine bestimmte Art und Weise bewegen. Nachdem ein Objekt gefunden wurde, ermöglicht die Formbeschreibung dessen Lokalisierung innerhalb eines Bildes bzw. die Bestimmung innerhalb wel-

ches Teiles der Sequenz das Objekt im Bild sichtbar ist. Um Metainformationen für diese Anwendungen zu extrahieren ist wiederum die Segmentierung unverzichtbar.

Als Folge der in den letzten Jahren erfolgten rasanten Entwicklung in beiden Bereichen gewinnt die Videosegmentierung zunehmend an Bedeutung, um Objekte in rechteckigem von Kameras erzeugtem Bildmaterial abzugrenzen und auszuschneiden. Bei Multimedia-Anwendungen ist in der Regel die Abgrenzung von Personen oder bewegten Objekten gegenüber einem Hintergrund gefordert. Neben der Objektabgrenzung in einem Bild ist auch die zeitliche Verfolgung von Objekten gefordert.

1.2 Implementierungsaspekte der Videoobjekt-Segmentierung

Segmentierungsverfahren für die Videoobjektgenerierung sind sehr rechenintensiv da sie aus einer Vielzahl durchzuführender Schritte bestehen. So nutzen Segmentierungsverfahren zur Ermittlung zusammengehöriger Bildbereiche mehrere Bildmerkmale. Neben der Objektabgrenzung umfassen sie auch Vorverarbeitungsschritte, die Verfolgung von Segmenten über der Zeit und die Identifizierung neuer Objekte. Eine große Anzahl an Operationen, die die Bilddaten direkt verarbeiten, bedingt die Verarbeitung großer Datenmengen und resultiert in hohen Anforderungen an die Speicherbandbreite. Iterative Verfahren, die eine Anpassung an unterschiedliche Strukturgrößen bzw. unterschiedliche Homogenitätsgrade ermöglichen, vervielfachen Rechenaufwand und Speicherbandbreite.

Bei Anwendungen im Kommunikations- und Überwachungsbereich müssen Videodaten einer Kamera mit der Geschwindigkeit segmentiert werden, mit der sie aufgenommen werden, um eine schritthaltende Verarbeitung zu gewährleisten. Da der Ablauf von Videosegmentierungsalgorithmen zwangsläufig von den verarbeiteten Videodaten abhängig ist, ergeben sich jedoch Schwankungen in der Verarbeitungsgeschwindigkeit. Überdimensionierte Implementierungen, die sicherstellen, daß in keinem Fall eine vorgegebene maximale Rechenzeit pro Bild überschritten wird, lassen sich im Falle der Echtzeit-Videoobjekt-Segmentierung durch Maßnahmen, wie z.B. Pufferung eines bzw. weniger Bilder vermeiden, wenn sich die Schwankung der Verarbeitungszeit und Kosten für den zusätzlichen Speicher tolerieren lassen. Eventuell kann die Segmentierung einzelner Bilder übersprungen werden, soweit die Genauigkeitsanforderungen dabei erfüllt werden. Trotzdem müssen Videosegmentierungsverfahren innerhalb derartiger Systeme eine statistische Echtzeitbedingung erfüllen, d.h. die Verarbeitungsdauer muß im Mittel ausreichend kurz sein.

Die Geschwindigkeit der Segmentierung spielt auch bei Anwendungen im Bereich der Produktion von objektbasiertem Videomaterial bzw. der Extraktion von Metadaten eine wesentliche Rolle, da sie entscheidend zu deren Dauer beiträgt. Die Geschwindigkeitsanforderungen können dabei die Anforderungen an Kommunikations und Überwachungsanwendungen um ein mehrfaches übersteigen. Für die Klasse von Produktions- und Extraktionsanwendungen bestehen weiche Echtzeitanforderungen.

Bei mobilen Anwendungen ist neben einer kompakten Bauweise der Leistungsverbrauch ein entscheidender Faktor. Bei CMOS-Schaltungen ist der Leistungsverbrauch proportional zur Schaltaktivität, zur Taktfrequenz und zum Quadrat der Versorgungsspannung, Leistungseffiziente Architekturen minimieren diese Größen. Hierfür ist zuererst die Vermeidung von Schaltaktivitäten durch Deaktivierung nicht benutzter Einheiten wichtig. Eine Architektur, die eine Berechnung mit einer geringeren Zahl von Schritten durchführen kann ist gegenüber einer anderen vorzuziehen. Mittels Parallelisierung läßt sich außerdem erreichen, daß eine längere Zykluszeit toleriert werden kann, so daß sich eine Leistungsersparnis durch Absenkung der Versorgungsspannung ergibt. Bei mobilen Anwendungen wird deshalb die Erzielung einer hohen Verarbeitungsleistung bei niedriger Taktfrequenz angestrebt.

Die rasante Entwicklung der Mikroelektronik ermöglicht erst die Implementierung von Echtzeit Videoobjekt-Segmentierung. Bedingt durch Herstellung Integrierter Schaltungen mit immer kleineren Strukturgrößen hat sich in den letzten Jahrzehnten die Menge pro Chip integrierbarer Transistoren entsprechend dem Gordon-Moore-Gesetz alle 18 Monaten verdoppelt. Zur Steigerung der Rechenleistung von Prozessoren tragen neben der Steigerung der Taktfrequenz auch architekturelle Verbesserungen bei, so daß das Joy-Gesetz für die Entwicklung der Rechenleistung eine Verdoppelung innerhalb eines Jahres vorhersagt [Rug99a]. Trotz wiederholt geäußerter Skepsis haben sich diese Steigerungsraten in den vergangenen Jahren immer wieder bestätigt.

Universalprozessoren, wie beispielsweise der Intel Pentium Prozessor, bieten trotzt hohem Flächenaufwand derzeit kein ausreichendes Maß an Rechenleistung für komplexe Echtzeit-Videosegmentierungsanwendungen, so daß hierfür Bedarf an leistungsfähigen Spezialarchitekturen besteht. Darüberhinaus haben Universalprozessoren im oberen Bereich des Leistungsspektrums einen sehr hohen Leistungsbedarf. Obwohl mobile Versionen dieser Prozessoren Techniken wie z.B. Speed Step zur Reduzierung des Leistungsverbrauches einsetzten, zielen diese auf die Leistungsreduzierung bei geringer Auslastung. Da jedoch bereits einfache Segmentierungsalgorithmen einen Prozessor voll auslasten, ist für die Betriebsdauer mobiler Geräte die Verlustleistung bei voller Auslastung entscheidend.

Da Segmentierungsalgorithmen viele unterschiedliche Operationen umfassen, führen Implementierungen mittels dedizierter Architekturen zu einer großen Anzahl von Hardware-Modulen, was im Hinblick auf eine kompakte flächeneffiziente Implementierung problematisch ist. Eine flexible Architektur, die einen weites Anwendungsspektrum abdeckt, ist auch von Vorteil, da sich ein großer Einsatzbereich durch die hohe Anzahl gefertigter Chips kostengünstig auswirkt. Innerhalb des Spektrums von Kommunikations-, Überwachungs-, Produktions- und Extraktions-Anwendungen bestehen unterschiedliche Anforderungen, beispielsweise aufgrund unterschiedlicher Aufnahmebedingungen. Bei Überwachungsanwendungen werden in der Regel festmontierte oder schwenkbare Kameras eingesetzt, so daß Verfahren für (zeitweise) stationären Hintergrund eingesetzt werden. Bei Kommunikationsanwendungen handelt es sich meist um sogenannte Kopf-Schulter-Szenen, bei denen eine Person sehr groß gegenüber dem Hintergrund aufgenommen wird. Um einen weiten Bereich abzudecken ist also Flexibilität im Hinblick auf den Einsatz unterschiedlicher Videosegmentierungsverfahren erforderlich.

1.3 Aufgabenstellung

Vor dem Hintergrund der zu erwartenden Bedeutung der Echtzeit-Videoobjekt-Segmentierung soll daher ein flexibles Architekturkonzept entwickelt werden, das ein breites Spektrum an Anwendungen der Videoobjekt-Segmentierung unterstützt, insbesondere stationäre und mobile Anwendungen. Aufgrund ihrer großen Bedeutung und des großen Rechenaufwandes sollen insbesondere pixelbasierte Verarbeitungsschritte mit hoher Geschwindigkeit unterstützt werden. Um hohe Flexibilität und den Einsatz in mobilen Anwendungen zu ermöglichen, ist die Integration in ein Hardware/Software-System anzustreben. Das zu entwickelnde Architekturkonzept soll eine hohe Verarbeitungsleistung und Flächeneffizienz erzielen, bei - mit Rücksicht auf die Einsetzbarkeit in mobilen Geräten und Terminals - einem hohen Maß an Parallelität und niedriger Taktfrequenz.

Bei Architekturen im Bereich der Videosignalverarbeitung ist insbesondere eine effiziente Speicherarchitektur wichtig zur Erzielung einer hohen Verarbeitungsleistung. Eine spezielle Schwierigkeit des Architekturentwurfes bei der Videoobjekt-Segmentierung ist der datenabhängige Ablauf der Operationen, insbesondere im Hinblick auf die Beurteilung der Leistungsfähigkeit einer Architektur. Ein bisher nicht ausreichend behandeltes Gebiet, auf das im Rahmen dieser Arbeit unter Berücksichtigung der besonderen Randbedingungen der Videoobjekt-Segmentie-

rung eingegangen werden soll, ist in diesem Zusammenhang die Simulation von Architekturen, die datenabhängige Algorithmen implementieren, mit möglichst geringem Modellierungsaufwand und der systematische Vergleich von Architekturalternativen.

Die vorliegende Arbeit stützt sich auf die Ergebnisse der am Institut von S. Herrmann durchgeführten Untersuchungen von Videosegmentierungsalgorithmen [HeMo99b] und deren strukturierte Softwareimplementierung [HeMo97b]. Schwerpunkt der vorliegenden Arbeit ist die Konzeption einer flexiblen Architektur und die Methodik für den Entwurf dieser Architektur.

1.4 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt: In Kapitel 2 wird auf Algorithmen zur Videoobjekt-Segmentierung eingegangen. Nach einem Überblick über die Aufgabenstellung der Videoobjekt-Segmentierung wird auf Bildmerkmale eingegangen, die sich zur Videoobjekt-Segmentierung nutzen lassen. Ausgehend von einer Klassifikation von Algorithmen zur Abgrenzung und Verfolgung von Videoobjekten werden bildpunktbasierte Operationen als häufig genutzt und rechenintensiv identifiziert und es wird ein Überblick über die von einer Hardware-Architektur zu unterstützenden Operationen und deren Funktionsweise gegeben. Die Analyse der Adressierungsarten, die diesen Operationen zugrundeliegen, bildet die Grundlage für die Architekturüberlegungen in den folgenden Kapiteln. Abschließend werden die Anforderungen hinsichtlich Bildformat und Rechenleistung dargelegt.

In Kapitel 3 wird auf Hardware-Architekturen für die Videoobjekt-Segmentierung eingegangen. Zunächst werden in Kapitel 3.1 algorithmische Anforderungen und Randbedingungen zusammengestellt. Die effiziente Unterstützung dieser Anforderungen dient im Folgenden als Kriterium für die Bewertung von Architekturkonzepten und -implementierungen. Mit Hinblick auf eine hohe Verarbeitungsgeschwindigkeit wird in Kapitel 3.2 ein Überblick über Arten möglicher Datenabhängigkeiten und daraus resultierende Konzepte zur Operationsausführung und Parallelisierung gegeben. Anschließend wird auf Konzepte zur Datenorganisation eingegangen, die kompakte Speicherung und schnelle Datenzufuhr gestatten. Vor dem Hintergrund der Forderung nach hoher Flexibilität werden ferner Konfigurierbarkeit und Programmierbarkeit miteinander verglichen. In Kapitel 3.3 werden bekannte Architekturen vorgestellt, die sich zur effizienten Implementierung von Algorithmen der Videoobjekt-Segmentierung eignen. Dabei werden sowohl Architekturen betrachtet, die komplette Segmentierungsalgorithmen abdecken, als auch solche, die sich zur Unterstützung von Segmentierungsalgorithmen einsetzen lassen. Eine bewertende Kategorisierung zeigt den Bedarf an einer problemangepaßten Architektur, die sowohl Bildoperationen als auch Segmentoperationen effizient unterstützt und führt zur Forderung nach einen neuen Architekturkonzept für die Videoobjekt-Segmentierung.

In Kapitel 4 wird ausgehend von den Ergebnissen der Algorithmenuntersuchung und dem Vergleich bekannter Architekturen das Grundkonzept der vorgeschlagenen Architektur entwickelt. Der Ansatzpunkt ist die Trennung von Adressierung und Verarbeitung, mit dedizierter Implementierung der generischen Adressierungsarten und konfigurierbarer Implementierung der Verarbeitungsoperationen, da hierdurch ein hohes Maß an Flexibilität und Effizienz erzielt werden kann. Ausgehend von diesem Grundansatz wird abgeleitet, mit welcher Parallelisierungsstrategie ein hohes Maß an Parallelität bei gleichzeitig hoher Flexibilität erreicht werden kann, wie die verschiedenen Arten zu verarbeitender Daten organisiert werden müssen, und wie die Steuerung erfolgen kann. Nach der Entwicklung des Grundkonzepts wird die Integration in ein Gesamtsystem für objektbasierte Kodierung bzw. objektbasierte Videoanalyse dargestellt.

Um von diesem Konzept zu einer implementierbaren Architektur zu gelangen, muß die Architektur der Adreßeinheit und der Verarbeitungseinheit entworfen werden. Zentrale Fragestellungen sind dabei die Zu- und Abfuhr großer Datenmengen und die Balancierung der Ressourcen, insbe-

sondere der Speicherschnittstelle und der Verarbeitungseinheit. Aufgrund datenabhängiger Operationen ist es unmöglich, eine effiziente Architektur ohne Simulation von Algorithmen mit realen Daten zu entwerfen. In Kapitel 5 wird auf die Probleme bei derzeit üblichen Entwicklungsabläufen für Hardware/Software-Systeme eingegangen, und es werden Defizite weiterentwickelter Entwurfsmethoden, im Hinblick auf einen schnellen Vergleich verschiedener Architekturen zur Implementierung eines komplexen datenabhängigen Algorithmus, aufgezeigt. Ausgehend von den Anforderungen im vorliegenden Fall wurde eine neue Simulationsmethodik entwickelt, die auf dem Prinzip der Trennung von Algorithmen- und Architekturmodell beruht und in Kapitel 5.4 dargestellt ist. Dort wird erläutert, wie durch gekoppelte Simulation komplementärer Modelle eine Architektursimulation für datenabhängige Algorithmen ermöglicht wird und wie durch einfach austauschbare Architekturmodelle der Vergleich von Architekturalternativen ermöglicht wird. In Kapitel 5.5 wird schließlich auf das Algorithmenmodell und die verschiedenen Architekturmodelle für das vorgeschlagene Architekturkonzept eingegangen.

Kapitel 6 gliedert sich in zwei Teile. In Kapitel 6.1 werden die Ergebnisse der Architekturuntersuchungen dargelegt, die mit der zuvor beschriebenen Simulationsmethodik erzeugt wurden. Zuerst wird auf den Operationsablauf, und auf die Balancierung von Verarbeitungsdauer zu Dauer der Speicherzugriffe eingegangen. Die Untersuchung der Speicheranbindung, die sich zuvor als essentiell für eine effiziente Architektur erwiesen hat, stellt den Kern der Untersuchungen dar. Für Bilddaten und für mit Segmenten assoziierte Daten wird neben der Speicherorganisation insbesondere der Einsatz von Pufferspeichern diskutiert, die durch den Aufbau einer Speicherhierarchie für eine effiziente Datenzufuhr sorgen. Weiterhin werden die Anforderungen hinsichtlich des Zeitverhaltens und der Größe des für Ausbreitungsprozesse erforderlichen Koordinatenspeichers dargestellt, und abschließend wird ein Überblick über die mit verschiedenen Speicherarchitekturen erzielbare Verarbeitungsleistung beim Einsatz statischer bzw. dynamischer Speichertechnologie gegeben. Aufgrund der Ergebnisse der angestellten Untersuchungen wurde das in Kapitel 4 erarbeitete Architekturkonzept in eine implementierbare Architektur umgesetzt. Der Aufbau dieser Architektur und deren Implementierung ist in Kapitel 6.2 beschrieben. Zuerst wird kurz auf die Steuereinheit eingegangen, dann wird die Verarbeitungseinheit und die Speicheranbindung dargestellt.

In Kapitel 7 werden die Ergebnisse der HW-Synthese und der Architektursimulation der CONIAN-Architektur vorgestellt. Zur Bewertung der Kosten für die Implementierung wird die Logik-, Verdrahtungs- und Speicher-Fläche bestimmt. Ausgehend von den Verarbeitungsdauern der Bildpunktoperationen wird die Systemgeschwindigkeit für verschiedene Benchmark-Operationen simuliert. Zum Beleg, daß sich mit der vorgeschlagenen Architektur Segmentierungsalgorithmen in Echtzeit ausführen lassen wird eine Simulation des in Kapitel 2.6 untersuchten Farbsegmentierungsalgorithmus durchgeführt. Im Anschluß erfolgt ein Vergleich mit der Geschwindigkeit und dem Flächenbedarf eines Universalprozessors bzw. anderer Hardware-Architekturen, und eine Bewertung der CONIAN-Architektur. Des weiteren wird in Kapitel 7 die Einsetzbarkeit der Simulationsmethodik untersucht. Um zu zeigen, daß mit der Simulationsmethodik sehr komplexe Systeme untersucht werden können, wird die Simulationsgeschwindigkeit für das Fallbeispiel CONIAN bestimmt und in Relation zur Architekturgröße gesetzt. Anschließend wird der Einsatzbereich und der Nutzen der Methodik diskutiert.

Kapitel 8 umfaßt, neben einer kurzen Zusammenfassung der vorgeschlagenen Architektur und ihrer Anwendungsmöglichkeiten, eine Diskussion von Möglichkeiten zu deren Weiterentwicklung. Darüberhinaus wird darauf eingegangen, daß sich die Simulationsmethodik nicht nur für die vorliegende Aufgabenstellung, sondern weitaus allgemeiner für den Vergleich von Hardware-Architekturen einsetzen läßt. Abschließend erfolgt ein Ausblick auf Möglichkeiten zur Weiterentwicklung der Simulationsmethodik.

2. Algorithmen für Videoobjekt-Segmentierung

In diesem Kapitel wird auf Algorithmen zur Videoobjekt-Segmentierung eingegangen. Nach einem Überblick über die Aufgabenstellung der Videoobjekt-Segmentierung in Kapitel 2.1 wird auf Bildmerkmale eingegangen, die sich zur Videoobjekt-Segmentierung nutzen lassen (Kapitel 2.2). Ausgehend von einer Klassifikation von Algorithmen zur Abgrenzung und Verfolgung von Videoobjekten (Kapitel 2.3) werden bildpunktbasierte Operationen als häufig genutzt und rechenintensiv identifiziert und es wird ein Überblick über die von einer Hardware-Architektur zu unterstützenden Operationen und deren Funktionsweise gegeben (Kapitel 2.4). Die Analyse der Adressierungsarten, die diesen Operationen zugrundeliegen, in Kapitel 2.5 bildet die Grundlage für die Architekturüberlegungen in den folgenden Kapiteln. Im Kapitel 2.6 werden schließlich die weiteren Anforderungen dargelegt, insbesondere Bildformat und erforderliche Rechenleistung.

2.1 Aufgabenstellung

Während die Zielsetzung anderer Videosegmentierungsverfahren z.B. die Abgrenzung von Fehlern im Bildmaterial zur Bildrestaurierung oder die Abgrenzung von Segmenten ist, die im Hinblick auf Form, Farbe und Helligkeit günstig zu kodieren sind, ist es Zielsetzung der Videoobjekt-Segmentierung, Objekte einer dreidimensionalen Szene, die mit einer Kamera aufgenommen wurde, gegeneinander und gegenüber dem Hintergrund abzugrenzen. Charakteristisches Merkmal von Objekten ist ein örtlicher Zusammenhang im dreidimensionalen Raum.

Durch die Reduktion der Anzahl der Dimensionen geht jedoch bei der Aufnahme einer dreidimensionalen Szene durch eine Kamera, die einer Abbildung des dreidimensionalen Raumes auf die zweidimensionale Bildebene der Kamera entspricht, Information über den räumlichen Zusammenhang verloren, so daß die dreidimensionale Szene nicht mehr eindeutig aus der Aufnahme rekonstruiert werden kann. Im aufgenommenem Bildmaterial kommen nicht nur solche Bildpunkte nebeneinander zu liegen, die in der dreidimensionalen Szene zu gleichen Objekten gehören, sondern an Objektgrenzen auch solche, die zu verschiedenen Objekten gehören, andererseits kann es durch Verdeckungen dazu kommen, daß zwischen Bildbereichen, die zu einem Objekt gehören, in den aufgenommenen Bildern kein örtlicher Zusammenhang besteht. Aus diesem Grund ist es nicht möglich alleine aufgrund des räumlichen Zusammenhangs in der Bildebene, Objekte abzugrenzen, jedoch sind im Bildmaterial andere Eigenschaften vorhanden, die für eine Abgrenzung genutzt werden können.

Eine weitere Zielsetzung ist die Verfolgung von Objekten über die Zeit, so daß in jedem Bild einer Videosequenz dieselben Objekte abgegrenzt werden, und eine Zuordnung der Objekte in zeitlich aufeinanderfolgenden Bildern möglich ist, sofern Objekte nicht aus dem Bild verschwinden bzw. neue Objekte im Bild erscheinen.

Je nach Anwendung unterscheiden sich die Anforderungen an die Genauigkeit und die Art der abzugrenzenden Objekte. Während es sich bei Kommunikationsanwendungen typischerweise um Personen bzw. Gesichter handelt, sind bei anderen Anwendungen Fahrzeuge, Gebäude und Gegenstände abzugrenzende Objekte. Bei Produktionsanwendungen hat die Segmentierung unter anderem zum Ziel, Objekte zu generieren, die sich mit Objekten aus anderen Quellen zu Videoszenen komponieren lassen. Da sich Fehler in der Segmentmaske nach der Komposition im Bild störend bemerkbar machen können, stellen Produktionsanwendungen sehr hohe Anforderungen an die Genauigkeit der Segmentierung. Bei interaktiven Anwendungen und bei Kommunikationsanwendungen, bei denen eine Szene auf dieselbe Weise komponiert wird, wie sie aufgenommen wurde, machen sich Ungenauigkeiten in der Segmentmaske nur gering bemerkbar, so daß Fehler in begrenztem Umfang tolerierbar sind. Bei Extraktionsanwendungen sind die Anforde-

rungen an die Genauigkeit typischerweise „weich“, d.h. Ungenauigkeiten können toleriert werden, jedoch verbessert sich mit zunehmender Genauigkeit der Segmentierung auch die Genauigkeit extrahierter Merkmale bzw. Deskriptoren. Neben der örtlichen Genauigkeit der Segmentmaske sind zeitliche Stabilität und Robustheit wichtige Anforderungen an Segmentierungsverfahren und werden daher auch bei der Evaluierung der Segmentierungsqualität betrachtet [MeMa01].

2.2 Bildmerkmale zur Objektabgrenzung

Aufgrund fehlender Information über den räumlichen Zusammenhang werden von Segmentierungsverfahren andere Merkmale zur Abgrenzung von Objekten benutzt. Die meisten Verfahren benutzen Bildmerkmale, die an den Stellen von Objektgrenzen Inhomogenitäten aufweisen.

Objekte bzw. Teile von Objekten haben oftmals eine einheitliche Farbe. Durch Beleuchtung ergibt sich an unterschiedlichen Punkten auf der Objektoberfläche eine unterschiedliche Helligkeit, je nach Art der Oberfläche und dem Winkel zu Lichtquellen. Abgesehen von Schatteneffekten und Objektkanten entstehen durch die Ausleuchtung gleichmäßige Übergänge, wogegen an der Grenze zwischen Objekten in der Regel Punkte mit unterschiedlicher Farbe oder Helligkeit zusammenstoßen, so daß sich an diesen Stellen i.a. starke Änderungen der Helligkeit und/oder Farbe im Bild ergeben. Um die zu einem Objekt gehörenden Bereiche zu bestimmen, ist es daher zweckmäßig, örtliche zusammenhängende Bereiche zu bestimmen, bei denen Helligkeit und Farbe eine große Homogenität aufweisen. Verfahren die auf diesem Prinzip beruhen werden als Farbsegmentierungsverfahren bezeichnet. Im Vergleich zu Verfahren, die andere Bildmerkmale zur Objektabgrenzung nutzen, weisen sie eine hohe Genauigkeit an Objektgrenzen auf, sie resultieren aber typischerweise in einer Übersegmentierung [Mul97], d.h. einer zu feinen Aufteilung des Bildes.

In manchen Fällen weisen Oberflächen von Objekten keine einheitliche Farbe auf, sondern eine einheitlich Textur. Eine Texturierung der Oberfläche resultiert aus einer farblichen Musterung, einer entsprechenden Mischung bzw. Strukturierung des zugrundeliegenden Materials oder durch Schatteneffekte bedingt durch eine unebene Oberfläche. Eine einheitliche Texturierung deutet darauf hin, daß eine Fläche zu einem Objekt gehört, und kann somit ebenfalls als Merkmal genutzt werden, um örtlich zusammenhängende Bereiche zu bestimmen. Im Vergleich zur Farbsegmentierung erzielen Textursegmentierungsverfahren eine geringe Genauigkeit [MaJa92].

Die meisten von Menschen hergestellten Objekte wie Gebäude, Fahrzeuge, Inneneinrichtung und Gegenstände sind aus Teilen zusammengesetzt, die sich auf einfache Formen zurückführen lassen. Nicht zusammenhängende, zu einem Objekt gehörige Bildbereiche, die durch Verdeckungen bei der Abbildung räumlich zusammenhängender Objekte in die Kameraebene entstehen, können als zusammengehörig identifiziert werden, wenn sie insgesamt eine einfache Form ergeben [HeMo99b]. Auch in Fällen der Übersegmentierung von Objekten kann durch Berücksichtigung der Objektform eine Zusammenfassung erfolgen.

Bei vielen Objekten handelt es sich um starre Objekte, d.h. sie sind nicht verformbar, und bewegen sich daher in einheitlicher Weise. Zeitlich aufeinanderfolgende Bilder einer Videosequenz gestatten es, aufgrund der Änderungen von Bild zu Bild die Bewegung im Bildmaterial zu ermitteln, die zur Bestimmung sich homogen bewegender Objekte benutzt werden kann. Da die Bewegung im Videomaterial der in die Kameraebene projizierten Bewegung und nicht der einheitlichen Objektbewegung im dreidimensionalen Raum entspricht, ist es erforderlich die Objektbewegung mit Bewegungsmodellen zu bestimmen, und die Parameter der Bewegunsmodelle zu vergleichen [MoHe99]. Verfahren zur Bewegungssegmentierung gestatten die Bestimmung von Objekten, die aus Teilen mit unterschiedlicher Farbe oder Textur bestehen. Da die Genauigkeit begrenzt ist, mit der ein kontinuierliches Bewegungsvektorfeld durch Verfahren zur

Bewegungsschätzung erzeugt werden kann, sind Verfahren zur Bewegungssegmentierung im Vergleich zu Farbsegmentierungsverfahren weniger akkurat.

Oftmals können auch anwendungsspezifische Merkmale genutzt werden, um die Unterteilung vorzunehmen. Beispielsweise kann beim Einsatz einer statischen, d.h. nicht bewegten Kamera, eine Trennung in Vorder- und Hintergrund aufgrund der Objektbewegung erfolgen. Auch bei Kopf-Schulter Szenen können charakteristische Merkmale des menschlichen Kopfes bzw. Gesichtes für die Segmentierung herangezogen werden.

Weitergehende Möglichkeiten zur Segmentierung sind theoretisch durch Objekterkennung realisierbar, wobei jedoch die Eingrenzung auf spezifische Objekte bzw. Merkmalen und die einem Echtzeiteinsatz gegenüberstehende hohe Komplexität dem entgegensteht. Andere Möglichkeiten ergeben sich auch durch Änderung des Hintergrundes und der Ausleuchtung, bzw. durch Verwendung zweier oder mehrerer gegeneinander versetzter Kameras, wobei dies jedoch die Änderung der Aufnahmebedingungen erfordert.

2.3 Überblick über Verfahrenstypen

2.3.1 Objektabgrenzung

Wie im vorherigen Kapitel dargelegt, beruht die Abgrenzung von Objekten in vielen Fällen auf der Bestimmung homogener Bereiche unterschiedlicher Bildmerkmale. In diesem Kapitel werden nun Verfahren zu deren Bestimmung näher betrachtet.

2.3.1.1 Punktorientierte Verfahren

Segmentierungsverfahren lassen sich in punktorientierte, regionenorientierte und konturorientierte Verfahren klassifizieren [Jäh93]. Punktorientierte Verfahren richten sich bei der Entscheidung, welchem Segment ein Bildpunkt angehört, nur nach den Eigenschaften des Punktes selbst. Entscheidend für punktorientierte Verfahren ist die Wahl geeigneter Merkmale, die hohe Diskriminanz bezüglich der zu segmentierenden Objekte aufweisen. Wenn die Zahl der im Bildmaterial auftretenden Merkmalsklassen und deren charakteristische Verteilung bekannt ist, spricht man von überwachter Segmentierung, andernfalls von nicht-überwachter Segmentierung [Kau95]. Im Falle überwachter Segmentierung, kann die Aufgabe mit den Schritten Merkmalsextraktion und Schwellwertbildung gelöst werden, im Falle nicht-überwachter Segmentierung werden im Anschluß an die Merkmalsextraktion Verfahren zur Clusterbildung eingesetzt.

Bei punktorientierten Verfahren, die die Umgebung eines zu segmentierenden Punktes lediglich zur Merkmalsberechnung berücksichtigen, jedoch nicht den Zusammenhang von Segmenten, ergibt sich typischerweise eine große Zahl kleiner, nicht zusammenhängender Segmente, so daß in der Regel Nachverarbeitungsschritte zur Vereinfachung der Segmentmaske und zur Eliminierung kleiner Segmente erforderlich sind.

Eine oftmals eingesetzte Methode ist die Berechnung der Merkmale aus dem Ergebnis der Subtraktion des statischen Hintergrundbildes. Derartige Verfahren werden als Änderungsdetektion bezeichnet. Wenn es keine Möglichkeit gibt, ein statisches Hintergrundbild aufzunehmen, verbleibt die Möglichkeit die Änderung zum jeweils vorherigen Bild zu benutzen. Da Objekte in diesem Fall nur bei Bewegung detektiert werden, und da die Detektion Objekte nur zum Teil erfaßt (es werden stark texturierte Teile erfaßt, und Teile, die aufgrund ihrer Bewegung erstmals Hintergrund verdecken) andererseits aber auch von Objekten aufgedeckte Bereiche des Hintergrundes erfaßt, müssen Techniken zur Eliminierung des aufgedeckten Hintergrundes und zur zeitlichen Akkumulation der detektierten Objektteile eingesetzt werden [MeWo97].

2.3.1.2 Regionenorientierte Verfahren

Regionenorientierte Verfahren gehen davon aus, daß Segmente zusammenhängend sein müssen, und berücksichtigen in Gegensatz zu punktorientierten Verfahren Nachbarschaftsbeziehungen zwischen Bildpunkten. Regionenorientierte Segmentierungsverfahren können in hierarchische Verfahren, wie Regionenwachstum und Regionenunterteilung, und in Repartitionierende Verfahren klassifiziert werden [Kau95].

Regionenwachstums-Verfahren, die eine sehr starke Verbreitung erlangt haben, gehen von einer sehr feinen Partitionierung - im Extremfall die einzelnen Bildpunkte - aus, und führen solange eine schrittweise Verschmelzung hinreichend ähnlicher Regionen durch, bis sich kein geeignetes Regionenpaar mehr findet. Das RSST-Verfahren (Rekursive Shortest Spanning Tree) [AlOn98] benutzt als Ähnlichkeitsmaß den quadratischen Abstand der Mittelwerte von Helligkeit und Farbe, gewichtet mit einem Faktor zur Bevorzugung kleiner Regionen und verschmilzt rekursiv Nachbarregionen mit dem jeweils kleinsten Distanzwert solange bis eine bestimmte Anzahl an Regionen erreicht wird oder bis ein bestimmer Distanzwert überschritten wird. Bei dem Wasserscheiden-Verfahren [ViSo91] handelt es sich um ein morphologisches Verfahren, das den Gradient eines Bildes als topographische Oberfläche betrachtet, die ausgehend von einem Satz an Startmarkierungen geflutet wird. Beim Fluten findet einem Ausbreitungsprozeß statt, bei dem die Einzugsbereiche der Täler der Startmarkierungen einem Segment zugeordnet werden.

Verfahren, die auf dem Prinzip der Regionenunterteilung beruhen, verfolgen den entgegengesetzten Weg. Sie gehen von einer sehr groben Partitionierung des Bildes - im Extremfall ist das ganze Bild eine Partition - aus, und verfeinern die Unterteilung schrittweise, so daß inhomogene Regionen in möglichst homogene Teile aufgespaltet werden. Derartige Verfahren haben jedoch die Schwierigkeit, daß es ein nichttriviales Problem ist, die optimale Zerteilung einer inhomogenen Region anzugeben, und genießen geringere Verbreitung als Regionenwachstums Verfahren [Kau95].

Während die bisher genannten hierarchischen Verfahren auf der Verfeinerung oder Vergröberung einer Partitionierung beruhen, versuchen repartitionierende Verfahren alleine durch Änderung einer Merkmalszuordnung eine Segmentierung zu erreichen. Neben der merkmalsbasierten Umordnung ist vor allem die bildpunktbasierte Umordnung von Bedeutung, die auch als Relaxation bezeichnet wird. Die Relaxation basiert auf der lokalen Energieberechung für alle möglichen Segmentzuordnungen eines Punktes, die auch als Kostenfunktion interpretiert werden kann [GeGe84]. Die Kostenfunktion kann dabei Terme enthalten, die den Beitrag für Kosten von Segmentgrenzen bzw. den Beitrag für Inhomogenitäten innerhalb von Segmenten darstellen. Bei der Relaxation wird die kleinstmögliche Kostenfunktion durch ein Minimierungsverfahren bestimmt, beispielsweise durch Simulated Annealing. Dabei erfolgt für jeden Punkt iterativ eine Überprüfung der Segmentnummer solange, bis für ein Bild keine Neuzuordnung mehr erfolgt ist.

2.3.1.3 Konturorientierte Verfahren

Konturorientierten Verfahren, die auch als Snake-Verfahren bzw. aktive Konturen bezeichnet werden, benutzen eine konturbasierte Betrachtungsweise zur Objektabgrenzung. Dem Segmentrand bzw. Snake wird eine lokale Energie zugeordnet, so daß die Bestimmung dessen Position durch Minimieren der Gesamtenergie erfolgt. Die Gesamtenergie eines Snakes ergibt sich als Integral über die lokale Energie, die sich wiederum aus interner und externer Energie zusammensetzt. Zur internen Energie gehören Terme, die nur von der Snake-Form abhängen, wie z.B. Terme für Krümmung der Kontur. Diese sorgen dafür, daß ein Snake nicht auf einen Punkt zusammengezogen wird, und daß eine glatte Form angestrebt wird. Die externe Energie sorgt dafür, daß es energetisch am günstigsten ist, wenn ein Snake über den Kanten im Bild zu liegen kommt. Zur Implementierung konturbasierter Verfahren erfolgt eine Diskretisierung der Snakes, d.h. eine Aufteilung in endlich viele Punkte. Der Abstand zwischen Snake-Punkten muß ausrei-

chend klein gewählt werden, damit Objektkonturen ausreichend genau angenähert werden, und ausreichend groß, damit kein sprunghafte Winkel- bzw. Abstandsänderungen auftreten und damit benachbarte Snake-Punkte nicht zusammenfallen.

Da globale Minimierungsverfahren wie z.B. [KaWi87] langsam sind, werden zur Minimierung der Energie Heuristiken zur lokalen Optimierung eingesetzt, sogenannte Greedy-Verfahren [WiSh92], [CoCo93]. Greedy-Verfahren suchen für jeden Snake-Punkt in dessen direkter Umgebung den energetisch günstigsten Punkt und verschieben den Snake-Punkt entsprechend. Dies wird solange wiederholt, bis kein Snake-Punkt mehr verschoben werden kann.

Konturbasierte Verfahren können zur Objektverfolgung eingesetzt werden, wenn die anfängliche Kontur durch Verfolgen der Segmenträndern einer Segmentmaske bestimmt wird. Die iterative Annäherung einer aktiven Kontur an eine Objektform ist hingegen schwierig, wenn nicht von einer ungefähren Startform ausgegangen werden kann. Vorteilhaft bei Snake-Verfahren ist die Flexibilität, die durch Modifizierung der Energiefunktion erreicht werden kann, und die Erzielung eines glatten Segmentrandes. Nachteilig ist, daß bei Verfolgung mehrerer Segmente Überlappungen und Löcher auftreten können. Probleme können bei Umrandungen mit starker Krümmung auftreten oder dadurch, daß sich Snakes von nicht ausreichend ausgeprägten Kanten entfernen [DWi97].

Trotz ihrer konturorientierten Arbeitsweise beinhalten Snake-Verfahren Bildpunktbasierte Vorverarbeitungsschritte z.B. zur Berechnung und Vereinfachung eines Gradientenbildes. Manche konturorientierte Verfahren benutzen auch Differenzbilder als Ausgangspunkt zur Berechnung der externen Energie. Neben rein konturorientierten Verfahren wurden auch kombiniert flächen- und konturorientierte Verfahren vorgeschlagen, um die Robustheit konturorientierter Verfahren zu steigern.

2.3.2 Einsatz mehrerer Bildmerkmale

Die im vorherigen Kapitel aufgelisteten Verfahren können direkt Helligkeit und Farbe als Merkmale für die Objektabgrenzung nutzen, sie können aber auch für weitere in Kapitel 2.2 dargestellte Bildmerkmale eingesetzt werden, z.B. durch Vorverarbeitungsschritte, in denen diese Merkmale extrahiert werden.

Während Helligkeit und Farbe eine sehr genaue Bestimmung der Lage von Segmentgrenzen ermöglichen, bieten andere Merkmale wie Textur, Form und Bewegung bessere Möglichkeiten zur Zusammenfassung zusammengehöriger Bereiche. Aus diesem Grund werden oftmals mehrere Bildmerkmale zur Videoobjekt-Segmentierung eingesetzt, insbesondere verbreitet sind Verfahren, die Farbe und Bewegung gemeinsam zur Segmentierung nutzen, wie beispielswiese [ErAl97] und [MoBh98].

Um einen weiten Einsatzbereich der Verfahren zu ermöglichen erfolgt oftmals eine hierarchische Segmentierung, d.h. es erfolgt eine Segmentunterteilung auf mehrere Ebenen, wobei die unterste Ebene der feinsten Unterteilung entspricht und die höheren Ebenen der Zusammenfassung von Segmenten der jeweils niedrigeren Ebene entsprechen.

Die meisten hierarchischen Algorithmen folgen dabei einer Bottom-Up Analysestrategie, d.h. sie beginnen mit der niedrigsten Ebene. In [Wan98] erfolgt zuerst eine Farbsegmentierung, anschließend werden die Segmente aufgrund der Farbe und Bewegung zusammengefaßt. Zusätzlich dazu wird in [GaMa97] die Tiefe und die semantische Information benutzt bzw. in [HeMo99b] die Segmentform. Manche Verfahren arbeiten mit einer modifizierten Bottom-Up Strategie: In [BPMo98] wird beispielsweise basierend auf den Prädiktionsfehlern einer globalen Bewegungskompensation gleichzeitig die Detektion des Vordergrundes und die Farbsegmentierung durchgeführt. In dem COST Analysis Model [SiSe97], [AlOn98] erfolgen Farb- und Bewegungssegmentierung voneinander getrennt, erst im Anschluß werden die Ergebnisse zusammengefaßt

2.3.3 Objektverfolgung

Im Zusammenhang mit Verfahren zur Objektabgrenzung müssen auch Verfahren zur Objektverfolgung betrachtet werden, um eine Zuordnung von Objekten in zeitlich aufeinanderfolgenden Bildern zu ermöglichen. Da in Videobildern neben Bereichen, die bereits im vorherigen Bild sichtbar waren, auch solche Bereiche enthalten sind, die durch Aufdeckung oder einen Kameraschwenk in den Bildbereich kommen, und da beide Arten von Bereichen im Bild verstreut sind, muß die Objektverfolgung mit der Objektabgrenzung interagieren.

Bei Verfahren wie z.B. [AlOn98] wird die Objektverfolgung nach der Objektabgrenzung durchgeführt, so daß die Bestimmung der Segmente im Videobild unabhängig vom Vorgängerbild erfolgt. Aufgabe der Objektverfolgung ist dabei die Zuordnung der Segmente im aktuellen Bild zu den Segmenten im vorherigen Bild. Da bei dieser Vorgehensweise zeitliche Fluktuationen der Segmentmaske durch die Entkopplung vom Vorgängerbild auftreten, wird jedoch bei anderen Verfahren wie z.B. [HeMo99b] zuerst die Objektverfolgung durchgeführt und dies als Basis für die Objektabgrenzung benutzt.

Einfache Verfahren beschränken sich darauf, die Segmentmaske des Vorgängerbildes als Startpunkt für die Objektabgrenzung zu benutzen. Das in [Lak96] beschriebene Verfahren verkleinert diese Segmentmaske zunächst durch Erosion und benutzt sie anschließend als Startmarkierung für das Wasserscheide-Verfahren. Obwohl dadurch bereits eine größere zeitliche Stabilität erreicht wird, ist die Stärke der Erosion ein schwierig zu wählender Parameter, da die Maske ausreichend stark verkleinert werden muß, um innerhalb durch Bewegung verschobener Objekte zu liegen, andererseits aber groß genug sein muß, um alle zum Objekt gehörigen Einzugsbereiche des Wasserscheide-Verfahrens abzudecken.

Ein anderer Weg ist es, die Videosequenz als dreidimensionalen Datensatz aufzufassen und die Objektverfolgung mit dem dreidimensionalen Wasserscheide-Verfahren durchzuführen. Vorteilhaft hierbei ist, daß die Wahl der Erosionsstärke entfällt, jedoch muß auch hier die Objektbewegung ausreichend klein sein, so daß sich die Einzugsbereiche des Wasserscheide-Verfahrens überdecken.

Um die Objektverfolgung bei großen Bewegungen zu ermöglichen, wird bei anderen Verfahren eine Bewegungskompensation der Segmentmaske durchgeführt. Um Bereiche zu finden, in denen neue Objekte z.B. durch Aufdeckung von Hintergrund erscheinen, wird in [HeMo99b] geprüft, wie gut das bewegungskompensierte Bild mit dem tatsächlichen Bild übereinstimmt, und in Bereichen mit starker Abweichung wird die Segmentmaske gelöscht, so daß für diese Bereiche eine von Vorgängerbild unabhängige Abgrenzung erfolgt.

2.4 Grundlegende Operationen

2.4.1 Klassifikation

Algorithmen der Videosignalverarbeitung werden anhand ihrer Eigenschaften in High-Level-Operationen und Low-Level-Operationen unterteilt [Pir98]. Gelegentlich wird auch die Klasse der Mid-Level- Operationen eingeführt, deren Eigenschaften zwischen Low- und High-Level-Operationen liegen.

Low-Level-Operationen sind charakterisiert durch die Verarbeitung großer Datenmengen bestehend aus einfachen Datenelementen mit einfachen Operationen großer Regularität. Die durchzuführenden Sequenzen von Operationen und Datenzugriffe stehen unabhängig von den verarbeiteten Daten im Voraus fest. Trotzt der einfachen Operationen ist aufgrund der großen Datenmenge die Komplexität von Low-Level-Operationen hoch. Wegen der einfachen Implementierbarkeit und der hohen inhärenten Parallelität lassen sich Low-Level-Operationen typischerweise gut in Hardware implementieren.

Im Gegensatz dazu stehen High-Level-Operationen bei denen komplexe Datenstrukturen verarbeitet werden, die jedoch insgesamt kleine Datenmengen darstellen. Die verarbeiteten Datenstrukturen sind in vielen Fällen von variabler Größe und die Grundoperationen komplexer als Low-Level-Operationen. Meist handelt es sich um irreguläre Operationen und meist beinhaltet der Ablauf datenabhängige Verzweigungen. Aufgrund der geringen Datenmenge ist die erforderliche Rechenleistung kleiner als die von Low-Level-Operationen. Daher und aufgrund der meist geforderten Flexibilität, lassen sich High-Level-Operationen am besten als Software auf einem Prozessor implementieren.

Eine weitere mögliche Klassifikation unterscheidet zwischen lokalen und nichtlokalen Operationen. Während nichtlokale Operationen in beliebiger Weise Bilddaten für Berechnungen nutzen, verwenden lokale Operationen lediglich die Daten eines begrenzten Bildausschnitts als Eingangsdaten für Berechnungen und erzeugen auch nur Ergebnisse für Bildpunkte, die innerhalb dieses Bildausschnittes liegen. Lokalität von Operationen ist vorteilhaft für die Datenorganisation, da nur auf eine definierte Menge an Daten bei Durchführung einer Operation zugegriffen wird. In dieser Hinsicht ist auch von Nutzen, daß viele nichtlokale Operationen durch mehrfache Ausführung lokaler Operatoren implementiert werden können.

Die meisten Videoobjekt-Segmentierungsverfahren umfassen sowohl Low-Level-Operationen, als auch High-Level-Operationen. Im Folgenden wird auf lokale Low-Level-Operationen detailliert eingegangen, da diese am besten für eine Hardware-Implementierung geeignet sind und den größten Teil des Rechenaufwandes bedingen.

2.4.2 Nachbarschaftsbasierte Operationen

2.4.2.1 Filter-Operationen

Durch Bearbeitung des Ausgangsbildes mit einem Filter können verschiedene Eigenschaften des ursprünglichen Bildes erkannt, hervorgehoben oder abgeschwächt werden. Tabelle 2.1 zeigt verschiedene Anwendungsbereiche und dafür geeignete Filteroperationen [Lan94].

Ziel	Filteroperation
Konturverschärfung	Hochpaßfilterung
Rauschunterdrückung	Tiefpaßfilterung Medianfilterung
Kantendetektion	Gradient, Sobel-, Laplace-Operator
Hervorheben von Kanten in Gegenwart von Rauschen	Bandpaßfilterung

Tabelle 2.1: Anwendungsbereiche von Filteroperationen

Für die Segmentierung spielen Filter zur Kantendetektion eine wichtige Rolle, da sich Segmentgrenzen typischerweise an Orten von Kanten befinden. Filter zur Rauschunterdrückung werden eingesetzt, um Eingangsbildern im Normalfall überlagertes Rauschen abzuschwächen. Bei Verarbeitung von Farbbildern werden Filteroperationen sowohl auf die Luminanzkomponente als auch auf die Chrominanzkomponenten angewandt.

2.4.2.1.1 Lineare Faltungsfilter

Filter, die das Ergebnisbild durch Faltung einer Filtermatrix mit dem ursprünglichen Bild berech-

nen, sind linear und werden als Faltungsfilter bezeichnet. Das Resultat der Faltung eines Bildes f(x,y) der mit einer Filtermatrix g(x,y) der Größe (2M+1)*(2N+1) ist ein Bild, das sich nach folgender Formel berechnet [Jäh93]:

$$(f*g)(x,y) = \frac{1}{(2M+1)\cdot(2N+1)} \cdot \sum_{i=-M}^{M} \sum_{j=-N}^{N} f(x-i, y-j) \cdot g(i,j) \qquad (2.1)$$

Für die Bildpunkte außerhalb des Bildbereiches wird angenommen, daß sie denselben Wert, wie der nächste Randpunkt des Bildbereiches besitzen. Die Filtermatrix wird oftmals graphisch dargestellt. Mit den Koeffizienten a bis i der Filtermatrix g(x,y) und den Werten p_1 bis p_9 des ursprüngliche Bild f(x,y) ergibt sich die in Bild 2.1 gezeigte Darstellung.

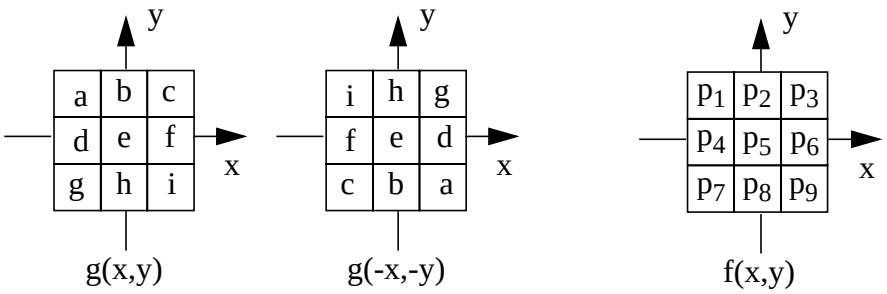


Bild 2.1: Filtermatrix und Bildausschnitt eines linearen 3x3 Filters

Mit diesen Bezeichnungen ergibt sich der Wert eines gefilterten Punktes zu:

$$(f*g)(0,0) = \frac{1}{9} \cdot (i \cdot p_1 + h \cdot p_2 + g \cdot p_3 + \ldots + c \cdot p_7 + b \cdot p_8 + a \cdot p_9) \qquad (2.2)$$

Durch die Koeffizienten des Filters werden die einzelnen Bildpunkte unterschiedlich gewichtet. Soll der Wertebereich des Resultats gleich dem der Eingangswerte sein, muß in Gleichung (2.1) ein zusätzlicher Faktor auf der rechten Seite eingeführt werden, der sich aus dem Kehrwert des Mittelwerts aller Koeffizienten ergibt. Somit ergibt sich:

$$(f*g)(x,y) = \frac{1}{k} \cdot \sum_{i=-M}^{M} \sum_{j=-N}^{N} f(x-i, y-j) \cdot g(i,j) \qquad \text{mit } k = \sum_{i=-M}^{M} \sum_{j=-N}^{N} g(i,j) \qquad (2.3)$$

In Bild 2.1 ist sowohl die normale als auch die gespiegelte Form der Filtermatrix dargestellt, die direkt mit dem korrespondierenden Bildausschnitt multipliziert wird. Die im folgenden beschriebenen Filter weisen Punktsymmetrie bezüglich des Zentralpunktes auf, so daß normale und gespiegelte Form identisch sind.

2.4.2.1.1.1 Gaußfilter

Bei Gaußfilter handelt es sich um ein Tiefpaßfilter, mit dem Rauschen in Bilder verringert werden kann. Das bedeutet, daß eine mit dem Bildinhalt unkorrellierte Komponente von Farb- oder Helligkeitswerten herabgesetzt wird [KlZa92]. Der Name resultiert daher, daß die Filterkoeffizienten innerhalb des Fensters näherungsweise einer auf Null abklingenden Gauß-Funktion entsprechen. Soll das Filter nur auf die horizontale oder vertikale Nachbarschaft angewandt werden, gewichtet man alle anderen Nachbarn mit 0. Diese Filter werden als horizontales bzw. vertikales 121-Filter bezeichnet.

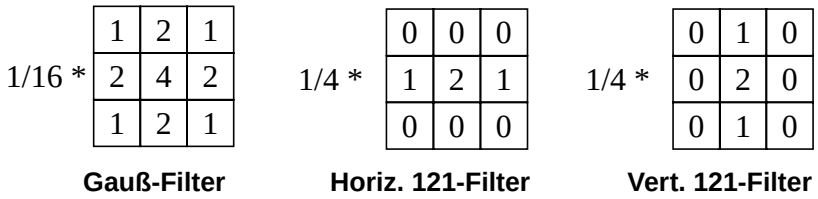


Bild 2.2: Gaußfilter und 121-Tiefpaß-Filter

2.4.2.1.1.2 Interpolationsfilter

Bei Verdoppelung der Auflösung eines Bildes ist es erforderlich die Zwischenpositionen an denen keine der Bildpunkte des ursprünglichen Bildes zu liegen kommen, zu interpolieren. Die Interpolation erfolgt dadurch, daß die Zwischenpositionen zunächst mit dem Wert Null belegt werden und anschließend ein Interpolationsfilter angewandt wird. Die in Bild 2.3 gezeigten Interpolationsfilter führen eine lineare bzw. bilineare Interpolation durch [Ohm95]. Sie sind bis auf den Vorfaktor identisch mit dem Gauß-Filter bzw. den 121-Tiefpaß-Filtern.

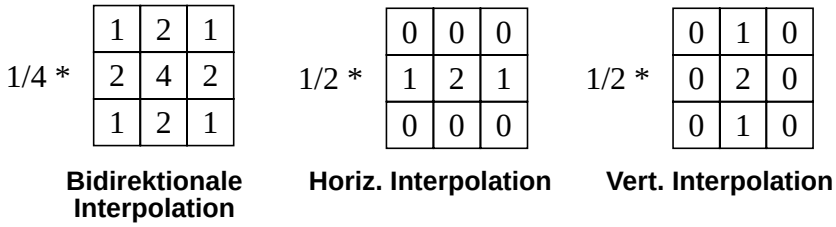


Bild 2.3: Horizontales, vertikales und bidirektionales Interpolations-Filter

2.4.2.1.1.3 Kantenoperatoren erster Ableitung

Oftmals ist der erste Schritt zur Segmentierung das Auffinden von Kanten. Kanten bedeuten Änderungen der Helligkeit oder Farbe zwischen aneinandergrenzenden Gebieten im Bild. Kantenoperatoren geben Informationen darüber, ob und wie stark eine Kante ausgeprägt ist. Lineare Filter, die zur Kantendetektion eingesetzt werden, besitzen Hochpaßcharakter und gehören zur Gruppe der Gradientenfilter.

Gradientenfilter, die der ersten Ableitung nach den Ortskoordinaten entsprechen, ergeben an den Stellen von Kanten lokale Extremwerte im Ergebnisbild und eignen sich besonders zur Bestimmung scharfer Kanten. Diese Gruppe von Gradientenfiltern gewichtet die Bildpunkte auf einer Seite einer möglichen Kante mit negativen Koeffizienten, die Bildpunkte auf der anderen Seite mit positiven Koeffizienten [Rad93]. Je höher die Differenz nach der Summation über alle Punkte im Fenster ist, desto größer ist der resultierende Ergebniswert für den Zentralpunkt und damit die Kantenstärke. Mit dieser Methode kann außerdem eine Aussage über die Kantenrichtung gemacht werden, weshalb diese Gradientenfilter zur Gruppe der gerichteten Kantenoperatoren gehören.

Der einfache Gradient und der Sobel-Operator sind Gradientenfilter, die der ersten Ableitung entsprechen. Beim einfachen Gradient wird lediglich die Differenz zwischen zwei gegenüberliegenden direkten Nachbarn des Bezugspunktes gebildet, beim Sobel-Operator werden auch die diagonal benachbarten Bildpunkte berücksichtigt. Beide Operatoren verwenden zwei Faltungsmatrizen, die jeweils die senkrechten und die waagerechten Kanten erkennen. Wie in Bild 2.4 gezeigt, entsteht durch Kombination gerichteter Gradientenfilter, die für unterschiedliche Kantenrichtungen ausgelegt sind, ein ungerichteter Kantenoperator. Dieser wird benutzt, wenn die

Stärke einer Kante von Interesse ist, jedoch nicht ihre Richtung. Dabei ist zu beachten, daß nach jeder Faltung mit einer Matrize der Betrag des Zwischenergebnisses gebildet wird, da sich sonst die Werte für die senkrechte und die waagerechte Komponente einer Kante auslöschen können.

$$1/2 * \left(\left| \begin{matrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix} \right| + \left| \begin{matrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{matrix} \right| \right)$$

Gradient

$$1/2 * \left(1/4 * \left| \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix} \right| + 1/4 * \left| \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \right| \right)$$

Sobel-Operator

Bild 2.4: Kantenoperatoren erster Ableitung

2.4.2.1.1.4 Kantenoperatoren zweiter Ableitung

Gradientenfilter, die der zweiten Ableitung nach den Ortskoordinaten entsprechen, führen an den Stellen von Kanten zu lokalen Nulldurchgängen im Ergebnisbild, da die zweite Ableitung der Bildfunktion an den Punkten Nullstellen aufweist, an denen sich in der ursprünglichen Funktion Wendepunkte befinden. Dadurch liefert diese Art von Filtern besonders bei sanften Übergängen gute Ergebnisse [Lan94]. Das Resultat läßt jedoch keinen Aufschluß über die Richtung der Kante zu, es handelt sich bei diesen Filtern um ungerichtete Kantenoperatoren. Das in Bild 2.5 gezeigte Laplace-Filter gehört zu dieser Gruppe. Es ist ein Hochpaßfilter, das tiefe Ortsfrequenzen abschwächt und hohe Ortsfrequenzen verstärkt.

$$1/4 * \begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

Bild 2.5: Laplace-Filter

Zur Gruppe ungerichteter Kantenoperatoren zählen auch morphologische Kantenoperatoren auf die in Kapitel 2.4.2.2 eingegangen wird. Ein Grundproblem der Kantenoperatoren ist ihre hohe Empfindlichkeit gegen Rauschen im Bild. Überdies ist oft auch eine Aussage über die Gesamtstärke von Kanten erforderlich, während die Kantenoperatoren getrennt auf die Helligkeit und Farbkomponenten angewandt werden. In diesem Fall werden die Ergebnisse der drei Komponenten addiert.

2.4.2.1.2 Nichtlineare Filter

Neben Rangordnungsfiltern sind Schwellwertoperationen, Sättigung und morphologische Operationen wichtige nicht-linearen Filter, wobei morphologische Operationen separat in Kapitel 2.4.2.2 behandelt werden. Rangordnungsfilter basieren auf der Sortierung der Eingangswerte in einem Bildfenster nach ihrer Größe, das Ergebnis ist eine Funktion der sortierten Werte.

Rangordnungsfilter haben eine gemeinsame algorithmische Struktur, aber unterschiedliche Auswirkungen auf die zu verarbeitenden Bilder [KlZa92].

2.4.2.1.2.1 Medianfilter

Durch Tiefpaßfilterung wird das Rauschen im Bild herabgesetzt, jedoch werden auch kleine Objekte und Konturen verändert. Das Median-Filter ist ein bekanntes Filter zur Rauschunterdrückung, das die Kantensteigung im Bild nahezu unverändert läßt [Jäh93]. Die Eingangswerte im aktuellen Bildfenster werden sortiert und dem Bezugspunkt wird der mittlere Wert aus der sortierten Liste zugewiesen, wie in Bild 2.6 dargestellt.

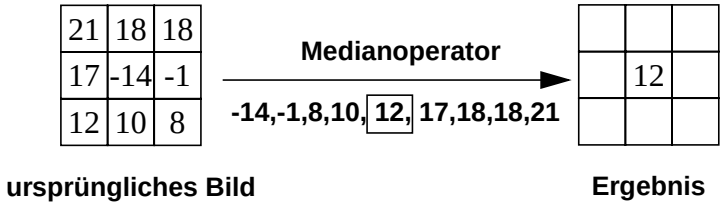


Bild 2.6: Bsp: Anwendung des Medianfilters

2.4.2.1.2.2 Extremwertoperator

Ausgehend von einer geordneten Liste von Einganswerten des aktuellen Fensters, bilden der maximale und der minimale Wert die Extremwerte. Diese werden benutzt, um Kanten zu schärfen, die beispielsweise zuvor als Nebeneffekt einer anderen Operation geglättet wurden. Detailreiche Strukturen mit geringem Kontrast werden im Ergebnisbild betont [KlZa92]. Dazu wird dem Bezugspunkt das Maximum oder das Minimum aus der momentanen Nachbarschaft zugeordnet, je nachdem welchem Extremwert der bisherige Wert des Bezugspunktes näher kommt. Die Funktion ist in Bild 2.7 verdeutlicht.

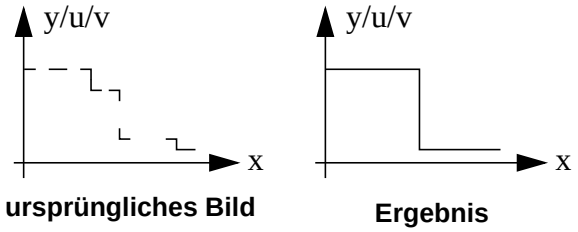


Bild 2.7: Bsp: Anwendung des Extremwertoperators

2.4.2.1.2.3 Schwellwertoperation

Bei der Schwellwertoperation wird der Eingangswert mit einem Schwellwert verglichen. Je nachdem, ob sich der Eingangswert unterhalb oder oberhalb des Schwellwerts befindet, wird ein entsprechendes Resultat ausgegeben [Dav97]. Die Schwellwertoperation wird eingesetzt, um Bildbereiche zu klassifizieren, um Bilder mit mehreren Graustufen in Schwarz-Weiß-Bilder umzuwandeln bzw. um aus einem Gradientenbild ein Kantenbild zu berechnen.

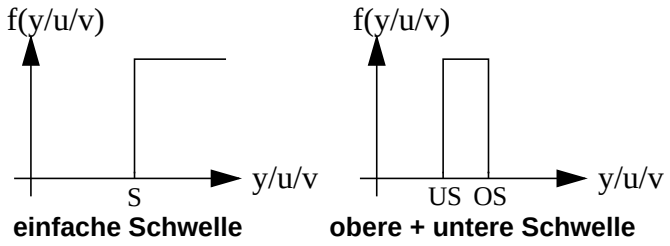


Bild 2.8: Schwellwertbildung mit einer/zwei Schwellen

2.4.2.1.2.4 Sättigung

Als Ergebnis der Sättigung bleiben alle Eingangswerte unterhalb des oberen Grenzwertes unverändert, während alle Werte oberhalb durch eine Konstante ersetzt werden. Analog wird bei einem unteren Grenzwert verfahren, der den Wertebereich unterhalb des Grenzwertes auf einen konstanten Wert abbildet. Werden obere und untere Grenze zusammen verwendet, so bleiben die Eingangswerte, die zwischen den Grenzen liegen unverändert, wohingegen die Werte außerhalb dieses Bereichs durch feste Werte ersetzt werden. Mit dieser Methode können Farb- und Helligkeitswerte gefiltert werden, die sich ausschließlich in einem bestimmten Wertebereich befinden. Bei einem Gradientenbild werden z.B. niedrige, durch Rauschen verursachte Gradientenwerte abgeschnitten.

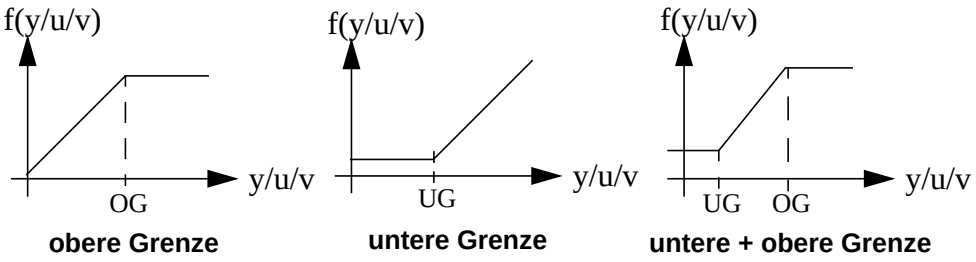


Bild 2.9: Sättigung mit unterer und/oder oberer Grenze

2.4.2.2 Morphologische Operatoren

Morphologische Operationen werden zur Segmentierung benutzt, weil sie sich in vielfältiger Weise für die Bearbeitung von Objektformen eignen. Die Grundoperationen, aus denen sich viele morphologische Operationen ableiten bzw. zusammensetzen lassen bilden Dilatation und Erosion, zwei nichtlinearen lokalen Operationen. Darüberhinaus werden z.B. morphologische Kantenoperatoren zur Berechnung von Kantenbilder eingesetzt. Morphologische Operationen, die auf Ausbreitungsprozessen beruhen, werden in Kapitel 2.4.4 behandelt.

2.4.2.2.1 Dilatation und Erosion

Für binäre Bilder lassen sich Erosion und Dilatation [Ser82] als Vereinigung bzw. Schnittmenge der in einem durch ein Strukturelement definierten Bereich liegenden Punkte darstellen. Wenn ein Segment im Ausgangsbild durch die Menge an schwarzen Punkten definiert ist, dann wird

- bei der Dilatation jeder Bildpunkt des Ergebnisbildes genau dann schwarz, wenn ein an seine Stelle verschobenes Strukturelement einen schwarzen Punkt des ursprünglichen Bildes überdeckt. Ein Beispiel für eine Dilatation ist in Bild 2.10 gezeigt. Wenn das Strukturelement nur die Zentralposition (0,0) enthält, wird die Form des ursprünglichen Segmentes erhalten, wenn

es zusätzliche Positionen enthält, wird die Form des ursprünglichen Segmentes in die jeweilige Richtung vergrößert.

- bei der Erosion jeder Bildpunkt des Ergebnisbildes genau dann schwarz, wenn ein an seine Stelle verschobenes Strukturelement ausschließlich schwarze Punkte des ursprünglichen Bildes überdeckt. Dadurch tritt der umgekehrte Effekt ein, wie das in Bild 2.10 dargestellte Beispiel für eine Erosion zeigt. Wenn das Strukturelement nur die Zentralposition (0,0) enthält, wird die Form des ursprünglichen Segmentes erhalten, wenn es zusätzliche Positionen enthält, wird die Form des ursprünglichen Segmentes in der jeweiligen Richtung verkleinert.

Mathematisch formuliert ergibt sich für die binäre Dilatation die Gleichung (2.4) und für die binäre Erosion die Gleichung (2.5), wobei F die Menge der zu einem Segment gehörenden Punkte, F(v) die Verschiebung von F um v und B das Strukturelement bezeichnet.

$$\delta_B(F) = \bigcup_{v \in B} F(v) \tag{2.4}$$

$$\varepsilon_B(F) = \bigcap_{v \in B} F(v) \tag{2.5}$$

Wird ein Grauwertbild als übereinanderliegende Schichtung von Binärbildern betrachtet, und jeder möglichen Graustufe ein Binärbild zugeordnet, das genau an denjenigen Stellen Punkte enthält, deren Wert größer oder gleich der jeweiligen Graustufe ist, lassen sich binäre Erosion und Dilatation für die Anwendung auf Grauwertbilder verallgemeinern. Erosion und Dilatation können für Grauwertbilder ohne Bestimmung der korrespondierenden Binärbilder effizient berechnet werden. Aus Gleichung (2.4) für die Dilatation ergibt sich eine Maximumbildung

$$(\delta_B f)(x) = \max_{v \in B} (f(x+v)) \tag{2.6}$$

und aus Gleichung (2.5) für die Erosion ergibt sich eine Minimumbildung

$$(\varepsilon_B f)(x) = \min_{v \in B} (f(x+v)) \tag{2.7}$$

Die Richtung, in der Erosion und Dilatation ausgeführt werden, wird wiederum durch das Strukturelement bestimmt. Dabei handelt es sich um den Satz von Bildpunkten in der Nachbarschaft, dessen Werte für die Minimum- bzw. Maximumbildung herangezogen werden, während Punkte, die nicht zum Strukturelement gehören, keinen Einfluß auf das Ergebnis haben. Bild 2.11 zeigt Erosion und Dilatation für ein Grauwertbild für einen Schnitt entlang der x-Achse, bei dem das zugrundegelegte Strukturelement den linken und rechten Nachbarn, sowie den Zentralpunkt enthält. Die Bezeichnung Erosion und Dilatation entspricht dem Einfluß der Operationen auf das Bild: Vergleichbar mit einem Höhenprofil werden die Werte der Bildpunkte angehoben und das Profil verbreitert (Dilatation) bzw. abgetragen und das Profil verschmälert (Erosion).

Die morphologischen Operationen Öffnen und Schließen können durch aufeinanderfolgende Anwendung von Erosion und Dilatation in definierter Reihenfolge durchgeführt werden [Ser82].

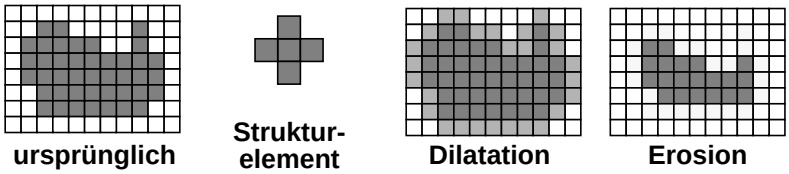


Bild 2.10: Bsp: Binäre Dilatation und Erosion

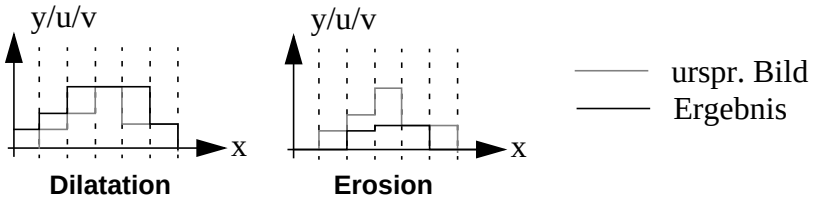


Bild 2.11: Bsp: Grauwert Dilatation und Erosion

2.4.2.2.2 Morphologische Kantenoperatoren

Der morphologische Gradient [Ohm95] wird durch Subtraktion des erodierten Bildes vom dilatierten Bild und Division durch zwei berechnet, man erhält also die Hälfte der maximalen Differenz zweier Werte innerhalb des vom Strukturelements bestimmten Ausschnitts. Auf diese Weise gewinnt man ein Maß, wie stark eine Kante ausgeprägt ist. Wenn man ein 3x3 Strukturelement zugrundelegt ergibt sich:

$$\text{ngrad} = \frac{1}{2} \cdot (\max(p1, p2, p3, p4, p5, p6, p7, p8, p9) - \min(p1, p2, p3, p4, p5, p6, p7, p8, p9)) \quad (2.8)$$

In diesem Fall des 3x3 Strukturelements handelt es sich um einen ungerichteten Kantenoperator, durch Verwendung eines gerichteten Strukturelements wird ein gerichteter Kantenoperator erzeugt.

Während beim morphologischen Gradient die Differenz von Maximum und Minimum der Nachbarschaft benutzt wird, bilden der Dilatations-Gradient und der Erosions-Grad die Differenz zwischen Maximum und Bezugspunkt bzw. zwischen Bezugspunkt und Minimum, entsprechend den Gleichungen (2.9) und (2.10) für ein 3x3 Strukturelement.

$$\text{dgrad} = \max(p1, p2, p3, p4, p5, p6, p7, p8, p9) - p_5 \quad (2.9)$$

$$\text{egrad} = p_5 - \min(p1, p2, p3, p4, p5, p6, p7, p8, p9) \quad (2.10)$$

Erosions- und Dilatations-Gradient besitzen eine Eigenschaft, die sie von den anderen bisher beschrieben Kantenoperatoren unterscheidet. Eine ideale Kante, d.h. ein abrupter Übergang von homogenen Bereichen erzeugt bei den bisher beschriebenen Kantenoperatoren ein Ergebnis von zwei Punkten Breite.

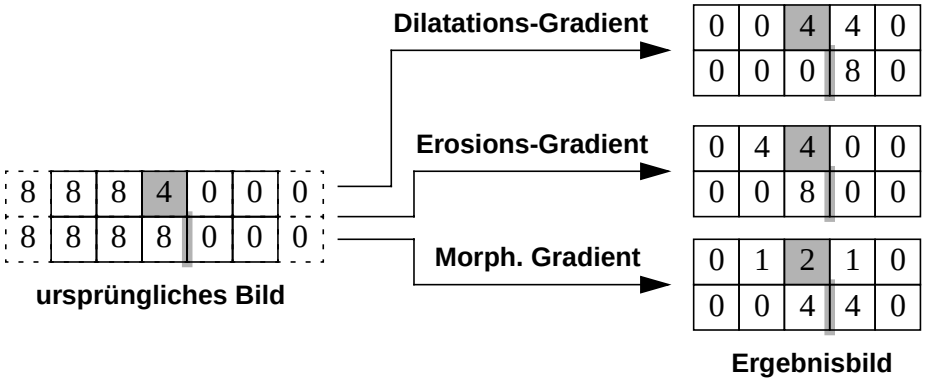


Bild 2.12: Bsp: Anwendung horizontaler morphologischer Kantenoperatoren

Der Erosions- und Dilatations-Gradient stellen die Kante mit einer Breite von einem Punkt dar, im Falle des Dilatations-Gradients befindet sich die Kante auf der Seite der Bildpunkte mit niedrigeren Werten, im Falle des Erosions-Gradients auf der Seite der Bildpunkte mit höheren Werten. Bild 2.12 zeigt einen Vergleich der verschiedenen morphologischen Kantenoperatoren für den Fall, daß ein horizontales 1x3 Strukturelement verwendet wird.

2.4.2.3 Relaxation

Die Relaxation [GeGe84] ist ein iteratives Verfahren, bei dem eine Kostenfunktion zur iterativen Korrektur einer Segmentzuordnung verwendet wird. Für jeden Bildpunkt werden die Kosten berechnet, die seine Zuordnung zu einem Nachbarsegment bzw. zu einem eigenen Segment bewirken würde. Der Bildpunkt wird danach dem Segment zugeordnet, bei dem die geringsten Kosten auftreten. Das ursprüngliche Bild muß vor der Anwendung der Relaxation bereits in Segmente aufgeteilt sein, wobei im Extremfall auch ein einzelner Bildpunkt ein Segment bilden kann. Durch iterative Anwendung werden solange Punkte neu zugeordnet, bis die Aufteilung optimal ist.

Die Kostenfunktion, die über die Segmentzuordnung entscheidet, berücksichtigt die Ähnlichkeit des jeweils betrachteten Punktes zu den Nachbarn, die bei einer Zuordnung dem selben Segment angehören. Das Potential wird in diesem Fall entsprechend Gleichung (2.11) aus der Distanz der Luminanz- und Chrominanzwerte zwischen Nachbar N und Zentralpunkt C berechnet. In dem Fall, daß Zentralpunkt und Nachbarpunkt unterschiedlichen Segmenten zugeordnet werden, werden feste Kosten für die Segmentgrenze veranschlagt.

$$dist_N = |y_C - y_N| + |u_C - u_N| + |v_C - v_N| \tag{2.11}$$

Für jede angenommene Zuordnung werden die Kosten ermittelt, indem die Kosten für die Grenze zu jedem Nachbarn berechnet und aufsummiert werden. Von den für jede mögliche Segmentzuordnung gebildeten Kosten wird der geringste Wert gewählt, und der Zentralpunkt erhält die Nummer des Segments, dem der minimale Kostenwert entspricht. Auf diese Weise erfolgt eine Zuordnung zu dem Segment, zu dem der Bildpunkt die größte Ähnlichkeit besitzt.

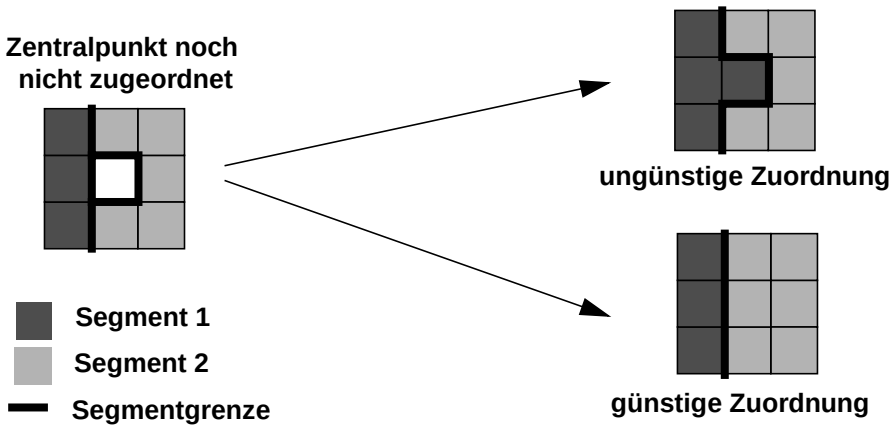


Bild 2.13: Günstiger und ungünstiger Verlauf der Segmentgrenze

Neben der Distanz von Helligkeits- und Farbwerten kann auch die Form der Segmentaufteilung bei der Kostenberechnung berücksichtigt werden. Da glatte Segmentkanten bevorzugt werden, wird der Kostenwert um einen konstanten Betrag erhöht, wenn die Segmentgrenzen keinen glatten Verlauf haben. Um festzustellen, ob der Verlauf der Segmentgrenze vorteilhaft ist oder nicht, muß die Segmentzugehörigkeit der direkten Nachbarn, also oben, unten, links und rechts, mit der Zuordnung des Bezugspunkten verglichen werden. Wenn, wie in Bild 2.13 gezeigt, drei Nach-

barn zu einem Segment gehören, ergibt die eine Zuordnung zum Segment des vierten Nachbarn einen ungünstigen Kantenverlauf und somit eine Erhöhung der Segmentkosten.

2.4.3 Bildverknüpfende Operationen

2.4.3.1 Bilddifferenz

Bei der Änderungsdetektion wird die Differenz von Bildern zueinander bzw. gegenüber einem Referenzbild bestimmt. Die Differenzbildung ist auch erforderlich, wenn man vom aktuellen Bild einen Störeinfluß beseitigen möchte und deshalb ein Korrekturbild subtrahiert. Neben der vorzeichenbehafteten Bilddifferenz entsprechend Gleichung (2.12), wird die Stärke der Differenz benutzt, die ein Maß der Ähnlichkeit zweier Bildpunkte darstellt, und sich als Betrag der Differenz bzw. als Differenzquadrat entsprechend Gleichung (2.13) und (2.14) berechnen läßt. Die Indizes 1 und 2 bezeichnen im Folgenden die Nummern des Eingangsbildes aus welchem die verarbeiteten Daten stammen. Bei Operationen, die Luminanz und Chrominanz auf identische Weise verarbeiten, sind nachfolgend nur die Formeln für die Luminanzkomponente y angegeben.

$$y = (y_1 - y_2) \quad \text{bzw.} \quad (y_2 - y_1) \tag{2.12}$$

$$y = |y_1 - y_2| \tag{2.13}$$

$$y = (y_1 - y_2)^2 \tag{2.14}$$

2.4.3.2 Akkumulation der Differenzbeträge

Die Summe der Differenzbeträge ist ein Maß für die Ähnlichkeit von Bildpunkten. Sie setzt sich aus den Helligkeits- und Farbwerten zweier Bildpunkte gemäß folgender Formel zusammen:

$$y = |y_1 - y_2| + |u_1 - u_2| + |v_1 - v_2| \tag{2.15}$$

Beim Tracking wird die Position von Segmenten bzw. Ausschnitten aus einem Bild im Folgebild gesucht. Dazu kann die Summe der Differenzbeträge von Luminanz und Chrominanz in einem Segment bzw. Ausschnitt des aktuellen Bildes und des Folgebildes über alle n zugehörigen Punkte im Ausschnitt akkumuliert werden. Dies wird als Sum of Absolute Distances (SAD) bezeichnet und entsprechend Gleichung (2.16) berechnet. Das Ergebnis wird minimal, wenn der am besten passende Ausschnitt im Folgebild gefunden ist. Wenn größere Abweichungen stärker gewichtet werden sollen, wird entsprechend Gleichung (2.17) das Quadrat der Abweichungen aufsummiert. Bei der Akkumulation der Differenzquadrate spricht man vom Mean Square Error (MSE).

$$\mathrm{SAD}(v) = \sum_{i=1}^{n} \left(\left| y_1(\underline{x}_i + \underline{v}) - y_2(\underline{x}_i) \right| + \ldots + \left| v_1(\underline{x}_i + \underline{v}) - v_2(\underline{x}_i) \right| \right) \tag{2.16}$$

$$MSE(v) = \frac{1}{n} \cdot \sum_{i=1}^{n} \left((y_1(\underline{x}_i + \underline{v}) - y_2(\underline{x}_i))^2 + \ldots + (v_1(\underline{x}_i + \underline{v}) - v_2(\underline{x}_i))^2 \right) \tag{2.17}$$

2.4.3.3 Bildbereichsersetzung

Die Bildbereichersetzung dient beispielsweise zur Kombination mehrerer Gradientenbilder oder von Distanz und Gradientenbild. In Abhängigkeit davon, ob ein Bildpunkt festgelegte Grenzen über- oder unterschreitet, wird der Wert an dieser Stelle im ersten oder zweiten Bild oder ein konstanter Wert als Ergebnis erzeugt. Die Bildbereichersetzung läßt sich allgemein mit Gleichung (2.18) bis (2.20) beschreiben, die auch Elemente der Schwellwertbildung bzw. Sättigung umfassen.

$$y = \begin{cases} val_1 & falls \quad y_1 > SW_1 \\ val_2 & falls \quad y_1 \leq SW_2 \\ val_3 & sonst \end{cases} \tag{2.18}$$

$$val_1, val_2, val_3 \in \{y_1, y_2, const_1, const_2\} \tag{2.19}$$

$$SW_1, SW_2 \in \{y_2, const_1, const_2\} \tag{2.20}$$

2.4.3.4 Gewichtete Bildaddition

Beim Einsatz im Zusammenhang mit objektbasierter Videokodierung spielt die Bildkomposition eine Rolle, die einer gewichteten Bildaddition entspricht. Ziel ist dabei, ein Bild aus verschiedenen sich überlappenden Objekten zusammenzusetzen. Die Objekte sind getrennt gespeichert und besitzen die Information über ihre Transparenz in Form eines Transparenzwertes für jeden Bildpunkt. Bei der gewichteten Bildaddition werden zwei Bildpunkte mit Hilfe dieses Transparenzwert α überlagert, der Werte von 0 bis 1 annehmen kann. Dabei bedeutet der Wert 0 vollständige Transparenz und der Wert 1 steht für einen völlig undurchsichtigen Bildpunkt.

Beim Zusammensetzen zu einem Bild wird beim untersten Objekt, das den Hintergrund bildet, begonnen. Nacheinander werden die darüberliegenden Objekte überlagert. Dabei wird der jeweils überlagerte Bildpunkt mit seinem Transparenzwert α multipliziert, während das Zwischenergebnis aus der gewichteten Addition der darunterliegenden Bildpunkte mit (1 - α) multipliziert wird. Die entsprechende Formel lautet

$$y_1(n+1) = (1-\alpha) \cdot y_1(n) + \alpha \cdot y_2 \tag{2.21}$$

2.4.4 Ausbreitungsprozesse

2.4.4.1 Segmentzuordnung

Werden neue Segmente gebildet oder wird von bereits bestehenden Segmenten die Segmentnummer geändert, z.B. bei Vereinigung von Segmenten, muß den betroffenen Bildpunkten eine Segmentnummer zuwiesen werden. Da Segmente zusammenhängende Bereiche darstellen müssen, muß bei der Segmentzuordnung sichergestellt werden, daß sie nur für einen solchen Bereich erfolgt. Da Ausbreitungsprozesse definitionsgemäß Nachbarschaftsbeziehungen berücksichtigen, bieten sie sich zum Einsatz für die Segmentzuordnung an.

Im Falle der Neubildung von Segmenten erfolgen Ausbreitungsprozesse innerhalb von Bildbereichen, deren Bildpunkte noch keine Segmentnummer besitzen und gleichzeitig das Homogenitätskriterium (2.22) erfüllen. Wenn die Größe der neu zu bildenden Segmente eine Rolle spielt, werden die Bildpunkte während des Ausbreitungsprozesses gezählt. Die vom Ausbreitungsprozeß erfaßten Bildpunkte erhalten eine bisher unbenutzte Segmentnummer. Bei der Vereinigung von Segmenten erfolgen Ausbreitungsprozesse innerhalb der Bildbereiche, deren Punkte zu dem zu vereinigenden Segment gehören. Die vom Ausbreitungsprozeß erfaßten Bildpunkte erhalten in diesem Fall die Nummer des Segments, mit dem sie fusioniert werden. Eine dritte Art der Zuordnung ist die Auflösung bestehender Segmente, indem die Segmentnummer der von Ausbreitungsprozessen erfaßten Bildpunkte gelöscht wird. Diese Bildpunkte stehen bei einer nachfolgenden Zuordnung zu Segmenten wieder frei zur Verfügung und können anderen Segmenten hinzugefügt werden.

$$g_{Lm} \cdot |y_C - y_N| + g_{Cr} \cdot (|u_C - u_N| + |v_C - v_N|) \quad \leq \quad d_{max} \tag{2.22}$$

g_{Lm}, g_{Cr} *Gewichtungsfaktoren für Luminanz und Chrominanz*
y_C, u_C, v_C Luminanz- und Chrominanzwerte des Zentralpunktes C

y_N, u_N, v_N	Luminanz- und Chrominanzwerte des Nachbarpunktes N
d_{max}	obere Grenze für Erfüllung der Homogenität

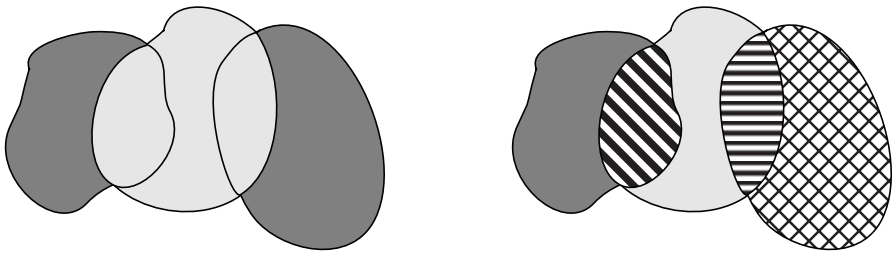
2.4.4.2 Prüfung und Bearbeitung fragmentierter Segmente

Durch Anwendungen von Operationen, die Punkte aus einem Segment entfernen (z.B. aufgrund eines Histogramms oder durch Erosion), kann es zur Fragmentierung des Segments kommen. Ein fragmentiertes Segment ist in mehrere Teile aufgespalten, die trotzdem dieselbe Segmentnummer besitzen. Um sicherzustellen, daß nur zusammenhängende Segmente existieren, muß nach Entstehung fragmentierter Segmente allen Teilen eine eigene Segmentnummer zugewiesen werden. Dazu wird nach kritischen Operationen zunächst geprüft, ob die selbe Segmentnummer mehrmals im Bild vorkommt, d.h. ob Fragmente entstanden sind. Bei Existenz von Fragmenten wird deren Größe bestimmt, und für alle Fragmente - abgesehen vom größten - wird eine Neuzuordnung der Segmentnummer durchgeführt.

2.4.4.3 Vereinigung von Segmentmasken

Die Vereinigung von Segmentmasken gestattet es mehrere Segmentmasken aus unterschiedlichen Quellen zu verschmelzen, die keine hierarchische Aufteilung darstellen. Eine solche Situation kommt z.B. dadurch zustande, daß eine Maske vom Benutzer erstellt und eine andere mittels Farbanalyse erzeugt wurde und die Segmente der einen Maske denen der anderen zugeordnet werden sollen, oder daß Masken durch getrennte Verarbeitung von Helligkeits- und Farbkomponenten erzeugt wurden, und zu einer Maske vereinigt werden sollen.

Bei der Vereinigung kommen in der Regel Segmente einer Maske über mehreren Segmenten der anderen Maske zu liegen. Um eine vereinigte Maske zu erzeugen, bei der jeder Bildpunkt nur einem Segment angehört, wird ausgehend von einem Punkt ein Ausbreitungsprozeß durchgeführt, der sich auf den Bereich erstreckt, der sich in beiden ursprünglichen Segmentmasken jeweils innerhalb derselben Segmente befindet. Während des Ausbreitungsprozeß wird dieser Bereich markiert, so daß anschließend nach einem noch nicht markierten Punkt gesucht werden kann, für den ein weiterer Ausbreitungsprozeß gestartet werden kann, solange bis eine komplette Vereinigung erreicht ist.



Überlagert dargestellte Segmentmasken **Segmentmaske nach der Vereinigung**

Bild 2.14: Vereinigung zweier Segmentmasken

2.4.4.4 Ermittlung von Randpunkten

Die Ränder von Segmenten spielen eine entscheidende Rolle, da sich an diesen Stellen die Eigenschaften von Bildpunkten stark ändern, außerdem werden sie z.B. zur Beschreibung der Kontur von Segmenten bzw. Objekten benötigt. Die inneren Randpunkte eines Segments werden durch die Bildpunkte gebildet, die selbst noch zum Segment gehören, die aber mindestens einen Nachbarpunkt aufweisen, der nicht mehr zum Segment gehört. Die äußeren Randpunkte sind diejenigen Bildpunkte, die selbst nicht zum Segment gehören, aber mindestens einen Nachbarpunkt

besitzen, der Teil des Segments ist. Die Ermittlung der Randpunkte entspricht einem Ausbreitungsprozeß, der nur innerhalb eines Segments stattfindet, und die Speicherung von Punkten veranlaßt, die die Bedingungen für einen äußeren bzw. inneren Randpunkt erfüllen.

2.4.4.5 Geodesische Distanz

Distanzberechnungen sind erforderlich um den Abstand von Punkten zu Segmenten bzw. von Segmenten zueinander zu bestimmen. Durch Bestimmung der Distanz der Punkte innerhalb eines Segments zum Segmentrand, können Segmente nicht nur hinsichtlich ihrer Fläche sondern auch hinsichtlich ihrer Form charakterisiert werden, beispielsweise um schmale Segmente zu eliminieren. Darüberhinaus ist die Distanzberechnung ist auch zur Bestimmung Geodesischer Skelette erforderlich.

Während zur Distanzberechnung normalerweise die direkte Verbindung herangezogen wird, können Hindernisse, wie in Bild 2.15 gezeigt, bei der Berechnung der Geodesischen Distanz berücksichtigt werden. Die Berechnung der Geodesischen Distanz erfolgt durch einen Ausbreitungsprozeß beginnend von den Punkten mit der Distanz Null. Die Distanz eines Punktes ergibt sich durch Erhöhung des kleinsten Distanzwertes in der Umgebung um eins, wobei die Ausbreitung in Reihenfolge ansteigender geodesischer Distanz erfolgen muß.

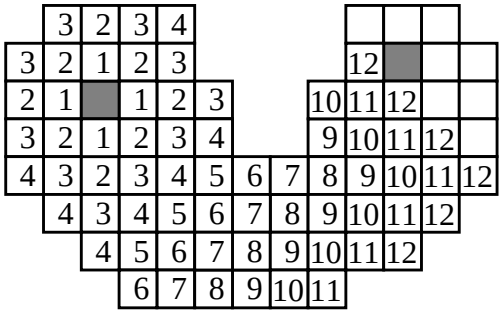


Bild 2.15: Geodesische Distanz

2.4.4.6 Geodesisches Skelett

Das geodesische Skelett wird durch eine kleine Menge von Punkten gebildet, auf die eine Segmentform reduziert werden kann, und aus der sie später wiedergewonnen werden kann. Neben dem Einsatz zur Kodierung von Segmentformen finden geodesische Skelette für die Formanalyse Anwendung, um durch Skelett Rekonstruktion zu Segmenten korrespondierende Formen zu konstruieren, die auf einfachen Grundformen beruhen [HeMo99b].

Jedem Skelettpunkt ist ein Abstandswert zum Rand des korrespondierenden Segments zugeordnet, so daß sich die Form als Vereinigung der von den Skelettpunkten überdeckten Bereiche ergibt. Zur Bestimmung der Skelettpunkte wird die Eigenschaft genutzt, daß Skelettpunkte keinen benachbarten Punkt besitzen, der einen größeren Abstand zum Segmentrand aufweist. Im ersten Schritt wird für alle Punkte des Segments der Abstand zum Rand bestimmt, und im zweiten Schritt wird für jeden Bildpunkt der Abstand zum Segmentrand mit dem der direkten Nachbarn innerhalb des Segments verglichen. Dabei werden lokalen Maxima bzw. Punkte innerhalb eines Plateaus als Skelettpunkte zusammen mit ihrem Distanzwert gespeichert.

Bei der Rekonstruktion der Segmentform aus den Skelettpunkten erfolgt ein Ausbreitungsprozeß, bei dem beginnend mit dem Skelettpunkt mit der größten Entfernung zum Rand die zum Segment gehörigen Punkte markiert werden. Nachdem die Ausbreitung um eine Distanzstufe vorangeschritten ist, werden auch die Skelettpunkte mit einem um einen Schritt kleineren Abstand zum Rand berücksichtigt. Dieser Prozeß wird solange fortgesetzt, bis der Abstand Null

erreicht ist.

2.4.4.7 Wasserscheide-Verfahren

Bei dem Wasserscheide-Verfahren handelt es sich um ein morphologisches Verfahren, mit dem Segmentgrenzen so bestimmt werden können, daß sie an den Stellen mit größtem Gradientenwert zwischen den jeweiligen Markierungen lokalisiert sind. Der Name des Verfahrens rührt daher, daß der Gradient eines Bildes als topographische Oberfläche betrachtet wird, die ausgehend von einem Satz an Startmarkierungen geflutet wird. Dabei findet einem Ausbreitungsprozeß statt, bei dem die Einzugsbereiche der Täler der Startmarkierungen einem Segment zugeordnet werden.

In Bild 2.16 ist der Ablauf des Wasserscheide-Verfahrens beispielhaft für einen eindimensionalen Schnitt durch ein Bild gezeigt. Die Startmarkierungen m_1, m_2 und m_3 breiten sich bei Flutung bis zur Höhe h_1, h_2 und h_3 wie in dem Bild gezeigt aus. Bei der Flutung von h_1 nach h_2 wird das lokale Maximum n_3 erreicht und das rechts danebenliegende Tal überflutet. Bei der weiteren Flutung von h_2 nach h_3 werden die Maxima n_1 und n_2 überflutet und Maximum g_2 erreicht. Da bei g_2 die Einzugsgebiete von m_2 und m_3 zusammenstoßen entsteht dort eine Segmentgrenze.

Da vor dem Einsatz des Wasserscheide-Verfahrens der Gradient an den Stellen der Bildpunkte berechnet wird, lassen sich sehr feine Bildstrukturen, bei denen nicht ausreichend Platz für die Lokalisation des Gradients an beiden Seiten eines Bereiches besteht, nicht voneinander abgrenzen. Dieser Nachteil kann durch eine modifizierte Form des Wasserscheide-Verfahrens vermieden werden, die ohne Gradientenberechnung direkt die Differenzen von Bildpunkte nutzt.

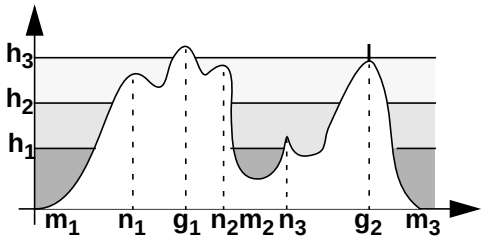


Bild 2.16: Bsp: Ablauf des Wasserscheide-Verfahren

2.4.4.8 Levelings

Die Klasse von Levelings umfaßt Operationen, mit denen sich unwichtige Details in Bildern auf ein konstantes Niveau verflachen lassen, während relevante Bildbereiche unverändert bleiben. Gemäß [Mey98b] sind Levelings durch definiert, daß bei der Abbildung des Orginalbildes f auf das Bild g für beliebige Nachbarpunkte p und q gilt:

$$g_p > g_q \Rightarrow (f_p \geq g_p) \wedge (g_q \geq f_q) \tag{2.23}$$

Zur Berechnung von Levelings werden zwei Eingangsbilder benutzt, ein Originalbild und ein sogenanntes Markerbild, dessen Relation zum Orginalbild bestimmt, welche Bereiche auf welche Weise vereinfacht werden: In Bereichen, in denen der Markerwert eines Punktes kleiner als dessen Originalwert ist, erfolgt die Ausbreitung ausgehend von Punkten, die in ihrer Nachbarschaft keinen Punkt mit einem größeren Markerwert und mindestens einen Punkt mit einem kleineren Markerwert aufweisen. Während des Ausbreitungsprozesses wird aus allen Markerwerten der noch nicht verarbeiteten Nachbarpunkte und aus allen Ergebniswerten der bereits verarbeiteten Nachbarpunkte das Maximum berechnet. Wenn das Maximum kleiner als der Originalwert des jeweiligen Bildpunktes ist, dann wird dessen Wert gleich dem Maximum gesetzt (d.h. erniedrigt). Auf diese Weise werden lokale Maxima in diesen Bereichen entfernt und durch flache Bereiche ersetzt. In Bereichen in denen der Markerwert größer als der Originalwert ist, wird entsprechend

mit umgekehrtem Vorzeichen verfahren, wobei die Ausbreitung in beiden Fällen die Grenzen dieser Bereiche nicht überschreitet.

2.4.5 Berechnung von Segmenteigenschaften

2.4.5.1 Segmentgröße und Umfang

Die Segmentgröße wird benötigt, um kleine Segmente zu verwerfen bzw. um Schwerpunkt und Mittelwert von Helligkeit und Farbe zu bestimmen. Die Größe bzw. Fläche eines Segments wird durch Abzählen der Anzahl der Bildpunkte bestimmt, zur Bestimmung des Umfangs werden die Randpunkte des Segments gezählt (siehe Kapitel 2.4.4.4). Aus dem Verhältnis von Größe zu Umfang ergibt sich eine Information über die Form des Segments. Eine relativ große Fläche bei kleinem Umfang ist ein Hinweis auf eine kompakte Form, während im umgekehrten Fall viele Randpunkte bei relativ geringer Fläche für eine verzweigte Form sprechen.

2.4.5.2 Umschießendes Rechteck

Die Bestimmung des kleinsten umschließenden Rechtecks eines Segments dient zur groben Lokalisation eines Segments und läßt sich zur Lagebeschreibung bzw. zur Eingrenzung des Verarbeitungsbereiches benutzten. Wie in Bild 2.17 gezeigt, müssen für die Bestimmung des umschießenden Rechtecks vier Koordinatenwerte festgehalten werden: Der minimale und maximale Wert der x-Koordinate, sowie der minimale und maximale Wert der y-Koordinate.

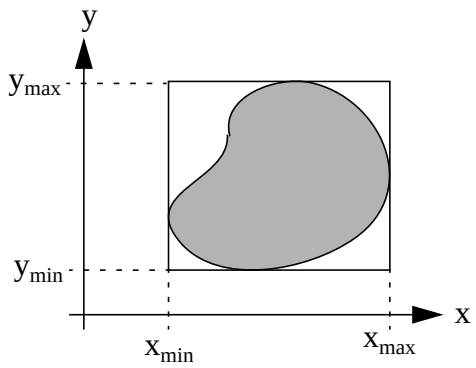


Bild 2.17: Umschließendes Rechteck

2.4.5.3 Schwerpunkt

Wie das umschießende Rechteck, so dient auch der Schwerpunkt zur Lokalisation bzw. Lagebeschreibung von Segmenten. Mit der Kenntnis der Segmentgröße N berechnet sich der Schwerpunkt entsprechend Gleichung (2.24) als Quotient aus der Summe der x- bzw. y-Koordinaten und der Anzahl der Punkte im Segment.

$$x_s = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad y_s = \frac{1}{N}\sum_{i=1}^{N} y_i \tag{2.24}$$

2.4.5.4 Mittelwert von Luminanz und Chrominanz

Der Mittelwert von Luminanz und Chrominanz dient zur Charakterisierung der Helligkeit bzw. Farbe des typischen Repräsentanten eines Segments, so daß sich aus der Differenz eines Bildpunkts zum Mittelwert des Segments, eine Aussage über die Homogenität treffen läßt, bzw. im Fall weicher Übergänge die Homogenität von Bereichen auch global betrachtet werden kann. Der Mittelwert läßt sich analog zur Bestimmung des Schwerpunktes entsprechend

Gleichung (2.25) berechnen.

$$y_s = \frac{1}{N}\sum_{i=1}^{N} y_i \qquad u_s = \frac{1}{N}\sum_{i=1}^{N} u_i \qquad v_s = \frac{1}{N}\sum_{i=1}^{N} v_i \tag{2.25}$$

2.4.5.5 Histogramm

Für jedes Merkmal wie z.B. Luminanz und Chrominanz eines Bildpunktes kann in einem Histogramm die Häufigkeit festgehalten werden. Histogramme liefern im Vergleich zum Mittelwert der Luminanz und Chrominanz eine detailliertere Analyse globaler Objekteigenschaften. Mit ihrer Hilfe kann beispielsweise ermittelt werden, ob die Häufigkeitsverteilung verschiedener Merkmale innerhalb eines Segmentes voneinander getrennte Maxima aufweist, so daß als Konsequenz beispielsweise eine Unterteilung des Segments vorgenommen werden kann. Die Berechnung der Histogramme geschieht durch Zählung der jeweiligen Häufigkeit.

2.4.6 Weitere Operationen

Neben den genannten Operationen umfassen Segmentierungsverfahren weitere Hilfsfunktionen, die ebenfalls zu den häufig benutzten Low-Level-Operationen zählen. Dies sind beispielsweise:

- Kopieren der Helligkeit, Farbe bzw. Segmentnummer von Punkten eines Bildes bzw. Segments
- Zuweisung eines konstanten Wertes für Helligkeit, Farbe bzw. Segmentnummer von Punkten eines Bildes bzw. Segments
- Auflösung der Segmentzugehörigkeit von Bildpunkten, die zu einem Segment gehören bzw. andere Eigenschaften erfüllen
- Bestimmung des Wertebereichs innerhalb eines Bildes
- Bestimmung des Wertebereichs und des Hauptmaximums in einem Histogramm

Außerdem werden Operationen benutzt, die auf den zuvor vorgestellten Operationen basieren, jedoch Helligkeit und Farbkomponenten nicht voneinander unabhängig verarbeiten. Hierzu zählt beispielsweise die Gradientenberechnung, bei denen im Anschluß die Helligkeits- und Farbkanten überlagert werden.

2.5 Generische Adressierungsarten

Die zuvor beschriebenen Operationen können in Adressierung und Verarbeitung unterteilt werden [HeMo97b]. Zur Adressierung zählt die Bestimmung der zu verarbeitenden Daten, und deren Laden und Speichern, die Verarbeitung umfaßt die Durchführung arithmetischer bzw. logischer Operationen. Eine wichtige Beobachtung ist, daß die in Kapitel 2.4 beschriebenen Operationen Daten auf ähnliche Weise adressieren, so daß sich Operationen anhand der Adressierung der Bilddaten in drei Klassen unterteilen lassen [HeMo97c],[HeMo97b]:

- Operationen die zwei Bildpunkte aus verschiedenen Bildern oder Bildausschnitten als Eingangsdaten benutzen, wie z.B. die Bilddifferenz oder die Sum of Absolute Difference (SAD). In Anlehnung an die Bezeichnungen bei Videokodierung wird diese Art der Adressierung als Inter-Adressierung bezeichnet.
- Operationen, die räumlich benachbarte Bildpunkte innerhalb eines Bildes als Eingangsdaten benutzen. Diese Art der Adressierung wird von Filter-Operationen benutzt (wie z.B. lineare Filter mit endlicher / unendlicher Impulsantwort und nichtlineare Filter wie Median und Relaxation), und in Anlehnung an die Begrifflichkeit bei Videokodierung als Intra-Adressierung bezeichnet.
- Operationen die im Gegensatz dazu nicht ganze Bilder oder Bildausschnitte, sondern beliebig

geformte Bereiche entsprechend dem Bildinhalt bearbeiten. Beispiele für derartige Operationen sind die Markierung zusammenhängender Bereiche, das Wasserscheide-Verfahren, die Konstruktion geodesischer Skelette oder morphologische Filter [Mey98]. Dies Art der Adressierung wird im Folgenden als Segment-Adressierung bezeichnet.

Während die ersten beiden Adressierunsarten in der Videosignalverarbeitung bekannt und verbreitet sind [Sch94], [PIMM1], wurde die dritte Adressierungsart - von [NoMe95] abgesehen - bisher kaum als universell einsetzbare Adressierungsart wahrgenommen. Operationen, die diese Adressierungsart nutzen, spielen jedoch insbesondere in der Videosegmentierung und Videoanalyse eine große Rolle.

Bei der Bearbeitung von Segmenten werden auch Daten benutzt, die nicht einem einzelnen Bildpunkt zugeordnet sind, sondern einem gesamten Segment, wie z.B. Segmenteigenschaften und Histogramme. Da der Zugriff auf mit Segmenten assoziierten Daten einem Tabellenzugriff entspricht, bei dem die Segmentnummer die Funktion eines Tabellenindex besitzt, wird er als Segmentindizierte Adressierung bezeichnet.

Die Adressierung definiert lediglich, welche Daten als Eingangs- bzw. Ausgangsdaten für eine Operation dienen, nicht jedoch welche Verarbeitungsschritte für eine Operation durchzuführen sind. Daher lassen sich die Adressierungsarten für eine Vielzahl von Operationen einsetzen. Bild 2.18 zeigt eine schematische Übersicht über die generischen Adressierungsarten, Tabelle 2.2 zeigt die den zuvor beschriebenen Operationen zugrundeliegenden Adressierungsarten. Aufgrund der datenabhängigen nicht vorherbestimmten Reihenfolge wird die Segment-Adressierung auch als *irreguläre* Adressierungsart bezeichnet, im Gegensatz zur Inter- und Intra-Adressierung, die auch als *reguläre* Adressierungsarten bezeichnet werden.

Adressierungsart	Operationen
Intra Adressierung	Lineare und nichtlineare Filter, Gaußfilter, Interpolationsfilter, Kantenoperator, Median, Extremwertoperator, Schwellwert, Sättigung, Morphologische Operationen, Erosion, Dilatation, Öffnen, Schließen, Relaxation
Inter Adressierung	Bilddifferenz, Akkumulation der Differenzbeträge (SAD / MSE), Bildbereichsersetzung, Gewichtete Bildaddition
Segment Adressierung	Segmentzuordnung, Test auf Fragmentierung, Vereinigung von Segmentmasken, Ermittlung von Randpunkten, Geodesische Distanz, Geodesisches Skelett, Wasserscheide-Verfahren, Levelings
Segmentindizierte Adressierung	Extraktion von Segmenteigenschaften, Histogramme

Tabelle 2.2: Zugrundeliegende Adressierungsarten von Low-Level-Operationen

Bei Intra-Adressierung wird unterschieden, ob die Daten innerhalb der Nachbarschaft in jedem Fall Daten des Eingangsbildes an der entsprechenden Stelle darstellen, oder ob die Verarbeitungsergebnisse von Bildpunkten im folgenden Schritt an die korrespondierenden Stellen der Nachbarschaft kopiert und wieder als Eingangsdaten verwendet werden. In Anlehnung an die Bezeichnung bei linearen Filtern spricht man von nicht-rekursiver bzw. rekursiver Verarbeitung. Während die nicht-rekursive Verarbeitung der Regelfall ist, wird die rekursive Verarbeitung beispielsweise bei der Relaxation eingesetzt, um instabiles Verhalten zu verhindern oder bei Opera-

tionen, die räumlich zusammenhängende Bereiche bestimmen, wie in Kapitel 2.5.4 beschrieben.

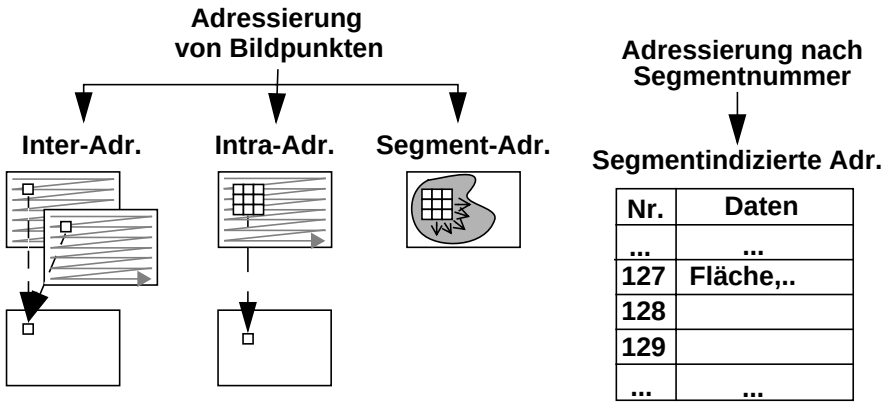


Bild 2.18: Schematische Übersicht über generische Adressierungsarten

2.5.1 Nachbarschaftssysteme

Operationen, die Intra-Adressierung benutzen können nach Form und Größe der Nachbarschaft unterschieden werden, die als Eingangsdaten benutzt werden. Typischerweise werden zweidimensionale Nachbarschaften eingesetzt, wie z.B. bei Gradientenfiltern und morphologischen Operationen. Die kleinste zweidimensionale Nachbarschaft besteht aus dem Zentralpunkt, und den vier links, rechts, oben und unten liegenden Nachbarpunkten, und wird als CON4 bezeichnet. Werden zur CON4-Nachbarschaft die vier benachbarten Diagonalelemente hinzugenommen ergibt sich die als CON8 bezeichnete und in Bild 2.19 gezeigte Nachbarschaft. CON12 und CON24 sind weitere symmetrische Nachbarschaftssysteme.

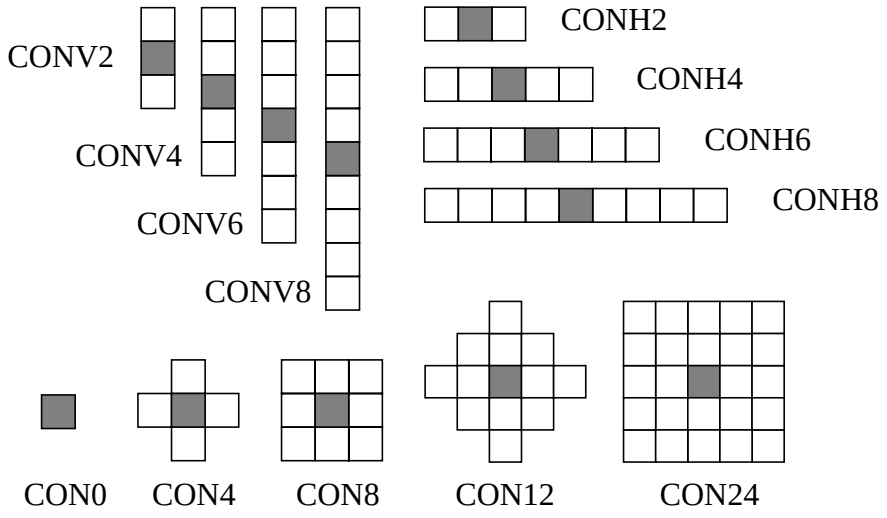


Bild 2.19: Ein- und zweidimensionale Nachbarschaftssysteme

Manche Operationen, wie. z.B. lineare Filter oder morphologische Operationen mit geeigneten Strukturelementen sind separierbar, d.h. sie können in zwei nacheinander ausgeführte Operationen mit einer horizontalen und einer vertikalen Nachbarschaft an Eingangspunkten aufgeteilt werden. In Bild 2.19 sind typische, mit CONH2 bis CONH8 bezeichnete, horizontale und, mit CONV2 bis CONV8 bezeichnete, vertikale Nachbarschaftssysteme gezeigt.

2.5.2 Randpunkte

Bei Operationen, die Bildpunkte innerhalb einer Nachbarschaft als Eingangsdaten verwenden, muß definiert werden, was geschieht, wenn die Nachbarschaft nicht vollständig innerhalb des Bildbereiches liegt. Bei einer Erosion sollen z.B. Randpunkte nicht abgetragen werden, d.h. für die außerhalb liegenden Punkte wird der Maximalwert des Wertebereiches angenommen, während bei einer Dilatation keine Ausbreitung des Randes in das innere des Bildes erwünscht ist, d.h. für die außerhalb des Bildbereiches liegenden Punkte wird der Minimalwert des Wertebereiches angenommen. Allgemein lassen sich bei dieser Art von Operationen Punkte außerhalb des Bildbereiches durch konstante Werte darstellen.

Bei Operationen wie z.B. Gradienten- oder Medianfilter ist hingegen erwünscht, daß die außerhalb liegenden Punkte sich nahtlos an den Bildrand anfügen, so daß die Filter auch am Bildrand entsprechend ihrer Aufgabe operieren. In diesem Falle muß der Wert jedes außenliegenden Punktes durch den Wert des nächstliegenden Randpunktes ersetzt werden.

2.5.3 Scan-Modus

Der Scan-Modus definiert die Reihenfolge, in der Bildpunkte bei regulärer Adressierung verarbeitet werden. Bild 2.20 zeigt den typischerweise verwendeten horizontalen und vertikalen Scan-Modus, die korrenspondierenden reversen Scan-Modi und die korrespondierenden mäanderförmiger Scan-Modi.

Bei nicht-rekursiver Verarbeitung ist aus algorithmischer Sicht die Wahl des Scan-Modus gleichgültig, da in jedem Fall dasselbe Verarbeitungsergebnis berechnet wird. Aus Sicht der Implementierung ist der horizontale Scan-Modus günstiger, wenn die Eingangsdaten in einer Nachbarschaft enthalten sind, die eine größere Breite als Höhe besitzt (CONH2 bis CONH8), bzw. der vertikale Scan-Modus im umgekehrten Fall (CONV2 bis CONV8).

.

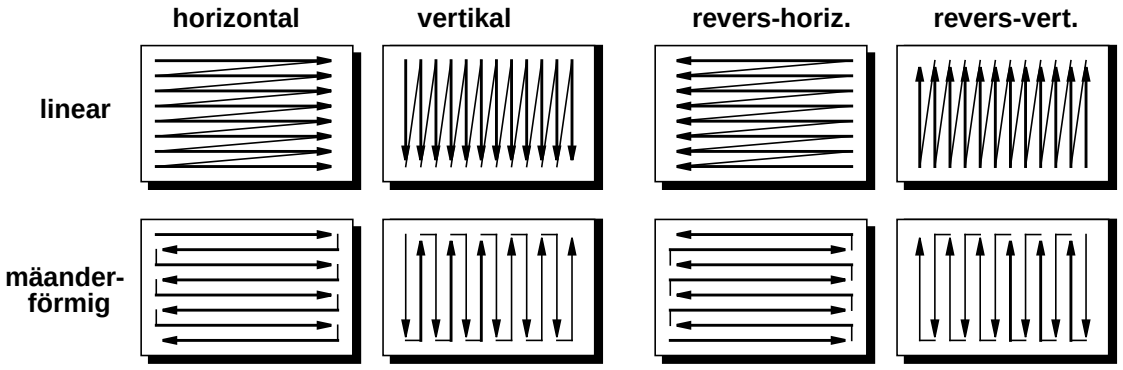


Bild 2.20: Übersicht über wichtige Scan-Modi

Wenn rekursive Verarbeitung zur Bestimmung räumlich zusammenhängender Bereiche genutzt wird, ist eine alternierende Verarbeitungsabfolge erforderlich, wie in Kapitel 2.5.4 dargelegt. Für die Intra-Adressierung bedeutet dies, daß zusätzlich zum horizontalen und zum vertikalen Scan Modus die korrespondierenden Scan-Modi mit reverser Verarbeitungsabfolge benützt werden.

Um Zeilensprünge zu vermeiden können mäanderförmige Scan-Modi benutzt werden. Dies ist jedoch nur bei geringer Zeilenbreite sinnvoll, d.h. wenn kleine Bildformate bzw. Bildausschnitte verarbeitet werden, und wenn dem komplizierteren Ablauf Einsparungen durch Vermeidung eines erhöhten Aufwandes für den Zeilenwechsel gegenüberstehen. Aus algorithmischen Gründen kommt der Einsatz mäanderförmiger Scan-Modi nur bei der Relaxation in Betracht um ggf. eine durch den Scan-Modus vorgegebene Kausalitätsrichtung zu vermeiden.

2.5.4 Räumlicher Zusammenhang und Rekursivität

Während die Bearbeitung bekannter, beliebig geformter, zusammenhängender Flächen mit Inter- bzw. Intra-Adressierung möglich ist, können unbekannte, beliebig geformte, zusammenhängende Flächen grundsätzlich nicht mittels nicht-rekursiver Inter- bzw. Intra-Adressierung bestimmt werden, da, wie in Bild 2.21 gezeigt, bei regulärer Verarbeitungsabfolge (z.B. zeilenweise oder spaltenweise Abfolge) nicht entschieden werden kann ob zwei Punkte zur gleichen Fläche gehören.

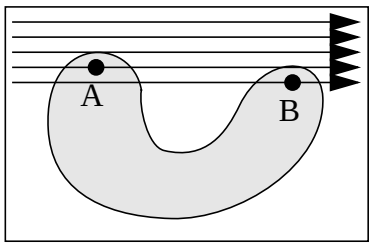


Bild 2.21: Problematik der Prüfung auf Zugehörigkeit zur selben Fläche

Um den räumlichen Zusammenhang von A und B zu ermitteln, muß entweder in der Verarbeitung oder in der Adressierung ein rekursives Element enthalten sein:

- Wenn die Verarbeitung rekursiv erfolgt, wird die Information in Richtung der Verarbeitungsabfolge von einem Bildpunkt zum nächsten dadurch weitergegeben, daß das Verarbeitungsergebnis eines Punktes als Eingangswert des folgenden Punktes benutzt wird, was dem Prinzip eines rekursiven Filters entspricht. Durch die Verarbeitungsabfolge ist eine Kausalitätsrichtung gegeben, daher muß eine Operation mit rekursiver Intra-Adressierung mehrfach auf ein Bild mit alternierender Verarbeitungsabfolge angewandt werden, wobei die Anzahl der Iterationen von den Bilddaten abhängig und nicht vorherbestimmbar ist.
- Wenn die Adressierung rekursiv erfolgt, beginnt die Verarbeitung mit Bildpunkt A. Anschließend werden dessen Nachbarn, dann deren Nachbarn, usw. verarbeitet bis Punkt B erreicht wird. Dies entspricht Segment-Adressierung. Gegenüber rekursiver Verarbeitung ist die Anzahl der zu bearbeitenden Bildpunkte durch die Größe der zu verarbeitenden Segmente definiert. Wenn ein Bild vollständig in Segmente unterteilt ist und alle Segmente bearbeitet werden, ist die Gesamtdauer unabhängig von den Bilddaten.

Die Ausbreitung findet bei rekursiver Adressierung nur innerhalb des Bereiches statt, in dem ein definiertes, von der Art der Low-Level-Operation abhängiges Nachbarschaftskriterium erfüllt ist. Im Folgenden werden unterschiedliche Implementierungen der Segment-Adressierung im Bezug auf die Anforderungen unterschiedlicher Low-Level-Operationen genauer betrachtet.

2.5.5 LIFO- und FIFO-Adressierung

Eine einfache Anwendung für rekursive Adressierung ist die Markierung zusammenhängender Bereiche. Die rekursive Adressierung kann auf zwei verschiedene Arten implementiert werden:

Bei der ersten Art kommt ein Stapel für die Speicherung der Koordinaten von Bildpunkten zum Einsatz, der nach dem LIFO-Prinzip arbeitet, d.h. zuletzt abgespeicherte Koordinaten werden zuerst entnommen. Ausgehend von einem Startpunkt wird getestet, welche Nachbarpunkte ein festgelegtes Nachbarschaftskriterium erfüllen, und so zum selben Objekt gehören. Wenn ein solcher Nachbarpunkt gefunden wurde, wird die Koordinate des aktuellen Punktes sofort auf dem Stapel abgelegt, und im Anschluß mit diesem Nachbarpunkt ebenso verfahren. Für jeden Punkt muß der Verarbeitungszustand gespeichert werden, d.h. ob er noch nicht betrachtet wurde, ob er auf dem Stapel abgelegt ist, oder ob er nach der Verarbeitung wieder vom Stapel entfernt wurde.

Als mögliche, zum selben Objekt gehörige Nachbarpunkte kommen nur solche in Frage, die zuvor noch nicht betrachtet wurden. Wenn für einen Punkt kein zum selben Objekt gehöriger Nachbarpunkt existiert, werden die obersten auf dem Stapel befindlichen Koordinaten entnommen, und es wird für diesen Punkt die Prüfung der weiteren Nachbarn fortgesetzt. Mit dieser als LIFO-Adressierung bezeichneten Variante der Segment-Adressierung wird jeder zu dem zusammenhängenden Bereich gehörige Bildpunkt mindestens einmal betrachtet, und dabei für ihn ein Verarbeitungsergebnis berechnet.

Bei der zweiten Art kommt eine Queue für die Speicherung der Koordinaten von Bildpunkten zum Einsatz, die nach dem FIFO-Prinzip arbeitet, d.h. zuerst abgespeicherte Koordinaten werden zuerst entnommen. Beginnend mit dem Startpunkt wird für alle Nachbarpunkte geprüft, ob sie ein festgelegtes Nachbarschaftskriterium erfüllen. Die Koordinaten solcher Nachbarpunkte werden in der Queue abgelegt, wobei - um eine Endlosschleife zu vermeiden - nur solche Punkte in Frage kommen, die entsprechend ihrem Verarbeitungsstatus bisher noch nicht betrachtet wurden. Im Anschluß werden aus der Queue die Koordinaten eines Punktes entnommen, und es wird mit ihnen auf dieselbe Weise verfahren. Diese Variante der Segment-Adressierung wird als FIFO-Adressierung bezeichnet und ist beispielsweise in [ViSo91] zum Einsatz für das Wasserscheide-Verfahren beschrieben.

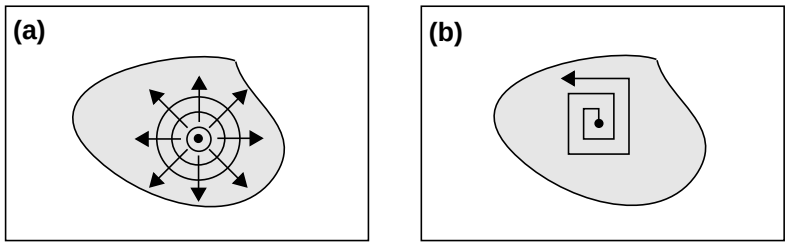


Bild 2.22: Verarbeitungsreihenfolge bei (a) FIFO- und (b) LIFO-Adressierung.

Ein wichtiger Unterschied zwischen FIFO- und LIFO-Variante der Segmentadressierung ist, daß im Falle der LIFO-Adressierung ein Bildpunkt ggf. mehrmals betrachtet wird, so daß die zugehörigen Daten mehrmals geladen werden müssen, wobei sich jedoch von einem Schritt zum nächsten die betrachtete Position nur um einen Bildpunkt verschiebt, so daß nicht alle Bildpunkte innerhalb der Eingangs-Nachbarschaft neu geladen werden müssen. Der zweite wichtige Unterschied betrifft die Verarbeitungsreihenfolge, wie in Bild 2.22 gezeigt. Während bei LIFO-Adressierung die Ausbreitung spiralförmig um den Startpunkt stattfindet, erfolgt bei FIFO-Adressierung die Verarbeitung streng in Reihenfolge ansteigender geodesischer Distanz vom Startpunkt. Bei einige Operationen wie z.B. der Markierung zusammenhängender Bereiche spielt die Verarbeitungsreihenfolge keine Rolle. Da bei den meisten Operationen eine Verarbeitung in Reihenfolge ansteigender geodesischer Distanz erforderlich ist, hat die FIFO-Variante einen weiteren Einsatzbereich als die LIFO-Variante.

2.5.6 Zeitpunkt der Verarbeitung

Die Berechnung des Verarbeitungsergebnisses für einen Bildpunkt kann bei FIFO-Adressierung zu zwei verschiedenen Zeitpunkten erfolgen: Entweder bevor seine Koordinate in der Queue gespeichert wird (*Process on Queue In* bzw. *PQI*) oder nachdem seine Koordinate aus der Queue entnommen wurde (*Process on Queue Out* bzw. *PQO*) [HeMo99a].

PQO bedeutet, daß nach der Entnahme einer Koordinate aus der Queue und dem Laden der zugehörigen Nachbarschaft jeweils nur ein Verarbeitungsergebnis für das Zentralpixel berechnet werden muß, so daß die komplette Nachbarschaft hierfür zur Verfügung steht. PQI bedeutet hingegen, daß nach der Entnahme einer Koordinate aus der Queue und dem Laden der zugehöri-

gen Nachbarschaft, je nach Anzahl der gefundenen Nachbarn mehrere Verarbeitungsergebnisse berechnet werden müssen. Gegenüber PQO ist nachteilig, daß für die jeweiligen Nachbarpunkte keine komplette Nachbarschaft zur Berechnung des Verarbeitungsergebnisses zur Verfügung steht, andererseits ist von Vorteil, daß zum Verarbeitungszeitpunkt feststeht, von welchem Punkt aus die Ausbreitung erfolgte. Dies ist beispielsweise bei Distanzberechnungen von Vorteil, da keine aufwendige Suche nach dem vorher verarbeiteten Nachbarn erfolgen muß, und da für alle Nachbarpunkte dasselbe Ergebnis berechnet wird. PQI kann als „Voranschieben“ des Verarbeitungsergebnisses vor der Wellenfront aufgefaßt werden, während PQO als „Nachziehen“ verstanden werden kann.

2.5.7 Bestimmung der Startpunkte

Bei Segment-Adressierung sind für den Start des Ausbreitungsprozesses ein oder mehrere Startpunkte erforderlich, die vor Beginn in der Queue abgelegt werden. Die Startpunkte stammen entweder aus einer in einem vorherigen Schritt ermittelten Liste oder aus regulärem Durchsuchen des Eingangsbildes.

Im Falle der Markierung zusammenhängender Bereiche eignet sich jeder unmarkierte Bildpunkt als Startpunkt. Sobald ein Startpunkt gefunden wurde, wird ein Ausbreitungsprozeß gestartet, der solange dauert, bis der, zum Startpunkt gehörende zusammenhängende Bereich, komplett markiert wurde. Anschließend wird mit der Suche nach Startpunkten fortgefahren, bis ein neuer Startpunkt gefunden wird, und für diesen wiederum ein Ausbreitungsprozeß gestartet wird. Dies wird als *Non-Collected-Mode* bezeichnet.

Bei anderen Operationen ist es notwendig den Ausbreitungsprozeß mit einem Satz an Startpunkten zu beginnen. Bei der Berechnung eines Distanzfeldes muß z.B. die Distanz vom Segmentrand berechnet werden, so daß die Verarbeitungsabfolge dieser Distanz entsprechen muß. Daher muß die Menge alle Randpunkte des Segments als Startpunkte verwendet werden, d.h. auch wenn bereits ein Startpunkt gefunden wurde, muß die Suche fortgesetzt werden, bis alle Punkte geprüft wurden, erst dann kann der Ausbreitungsprozeß erfolgen. Dies wird als *Collected-Mode* bezeichnet.

Der Collected-Mode kann auch eingesetzt werden, um mehrere Segmente gleichzeitig zu bearbeiten. Bild 2.23 zeigt den Unterschied der Verarbeitungsabfolge zwischen Non-Collected-Mode und Collected-Mode. Beim Start von einem Punkt aus (a) ergibt sich eine kreisförmige Charakteristik bei euklidischer Distanz, eine rhombische Charakteristik bei Manhattan Distanz bzw. eine rechteckige Charakteristik bei Schachbrett-Distanz. Ein Satz an Startpunkten erzeugt eine (b) bzw. mehrere (c) sich parallel ausbreitende Wellenfront(en).

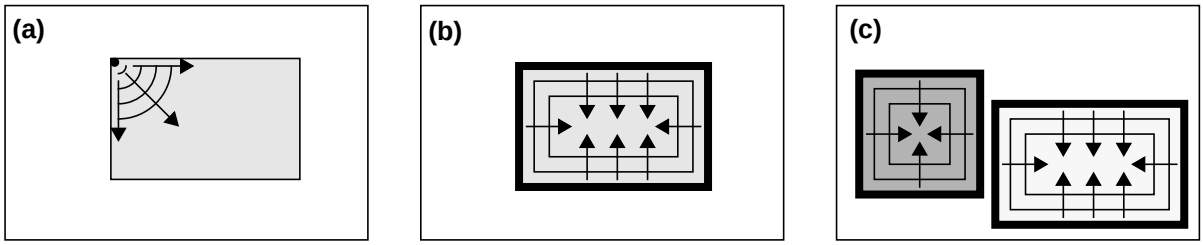


Bild 2.23: Verarbeitungsreihenfolge in Abhängigkeit der Art der Startpunktsuche (a) Non-Collected-Mode bzw. Collected-Mode mit (b) einem zusammenhängenden bzw. (c) nicht-zusammenhängenden Satz an Startpunkten

2.5.8 Hierarchische Queues

In Fällen, in denen die Verarbeitungsabfolge von der Reihenfolge ansteigender geodesischer

Distanz abweicht, da die Ausbreitungsgeschwindigkeit durch die verarbeiteten Daten bestimmt wird, ist eine gewöhnliche FIFO nicht mehr ausreichend. Ein typisches Beispiel für adaptive Ausbreitungsgeschwindigkeit ist das Wasserscheide-Verfahren [Mey91], bei dem die Ausbreitung in Reihenfolge ansteigenden Gradients erfolgt, d.h. sie erfolgt um so schneller je geringer der Anstieg der Gradientenwerte ist, lediglich innerhalb von Bereichen mit gleichem Gradientenwert erfolgt die Ausbreitung in Reihenfolge steigender geodesischer Distanz. Um dieses Ausbreitungsverhalten zu erreichen, müssen hierarchisch geordnete FIFOs eingesetzt werden, die gewöhnlich als hierarchische Queues bezeichnet werden. Für jeden Punkt, der das Nachbarschaftskriterium erfüllt, wird ein Hierarchiewert errechnet, der bestimmt, in welcher der Queues seine Koordinaten abgelegt werden. Bei der Entnahme von Koordinaten aus dem Satz hierarchisch geordneter Queues, entspricht die Bedeutung der Hierarchiestufe einer Priorität, d.h. die Entnahme erfolgt aus derjenigen nicht-leeren Queue, die die höchste Hierarchiestufe besitzt. (Im Falle des Wasserscheide-Verfahrens entspricht dies dem kleinsten Wert des Gradients). Bild 2.24 zeigt ein Beispiel für einen derartigen Ausbreitungsprozeß. Ein weiteres Beispiel für Ausbreitungsprozesse mit adaptiver Ausbreitungsgeschwindigkeit ist die Rekonstruktion geodesischer Skelette. Die Ausbreitungsreihenfolge wird in diesem Fall durch die zu den Skelettpunkten gehörenden Distanzwerte festgelegt.

Anstelle hierarchischer Queues, kann auch eine hierarchische LIFO eingesetzt werden, da die Verarbeitungsreihenfolge der Punkte keine Rolle spielt, wenn diese nicht zur höchsten Punkte enthaltenden Hierarchiestufe gehören. Um die Punkte der höchsten Hierarchiestufe in Reihenfolge ansteigender geodesischer Distanz zu verarbeiten, müssen zwei LIFOs zu dieser Hierarchiestufe zugeordnet werden, die alternierend operieren. Dies bedeutet, daß Punkte aus der ersten LIFO entnommen werden, während zu speichernde Punkte in der zweiten LIFO abgelegt werden. Wenn keine Punkte mehr entnommen werden können, da die erste LIFO den Leerzustand erreicht hat, werden beide LIFOs vertauscht. Jeder Vertauschungsvorgang entspricht dem Beginn einer neuen Distanzstufe. Erst wenn beide LIFOs leer sind, gilt auch die zugehörige Hierarchiestufe als leer.

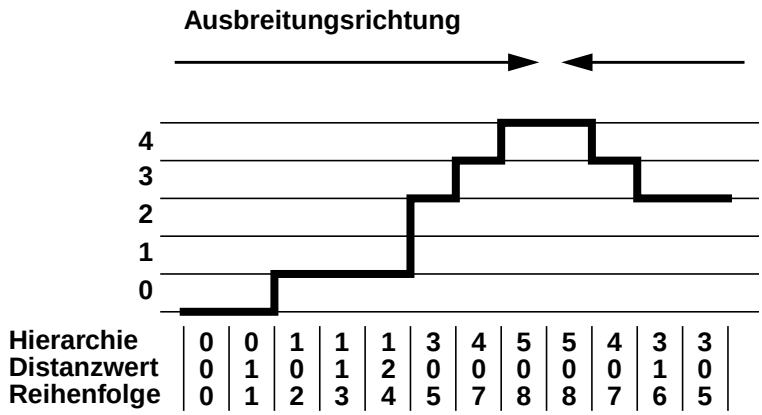


Bild 2.24: Verarbeitungsreihenfolge beim Einsatz hierarchischer Queues am Beispiel einer eindimensionalen Gradientenoperation

2.5.9 Änderung der Hierarchiestufe

Bei Operationen wie z.B. den Levelings, erfolgt die Propagation von mit Bildpunkten assoziierten Werten. Zu Beginn werden alle Punkte, die als Kandidaten für die Propagation in Frage kommen als Startpunkte identifiziert, und in denjenigen Queues abgelegt, deren Hierarchiestufen den Funktionswerten der Startpunkte entsprechen [Rob97]. Anschließend erfolgt der Ausbreitungsprozeß unter Beachtung der Hierarchiestufen. Bei der Propagation ist es möglich, daß Start-

punkte erreicht werden, deren Koordinaten bereits in einer Queue mit niedrigerer Hierarchiestufe abgelegt sind, so daß es erforderlich ist sie aus dieser Queue zu entfernen und ein einer anderen Queue abzulegen, ohne daß eine Verarbeitung erfolgt. Dies kann entweder zu dem Zeitpunkt erfolgen, zu dem diese Punkte in einer höheren Hierarchiestufe abgelegt werden, oder es kann, da es sich um eine begrenzte Anzahl an Punkten handelt, zum Zeitpunkt, an dem ihre reguläre Verarbeitung erneut erfolgen würde, die wiederholte Verarbeitung übersprungen werden.

2.5.10 Segmentindizierte Adressierung

Bei der Bearbeitung von Segmenten spielen nicht nur Bilddaten eine Rolle, sondern es werden auch Daten benutzt, die nicht einem einzelnen Bildpunkt zugeordnet sind, sondern einem gesamten Segment. Beim Regionenwachstum wird z.B. getestet, wie ähnlich ein einzelner Punkt der mittleren Helligkeit bzw. Farbe eines Segments ist. Ein anderes Beispiel ist die Segmentgröße, die benutzt wird, um zu entscheiden, ob es sich um relevante Segmente oder kleine Splitter handelt. Wichtige Segmenteigenschaften sind z.B:

- Segmentnummer im vorherigen Bild
- Segmentgröße und Segmentumfang
- Koordinaten des umschließenden Rechtecks
- Schwerpunkt
- Mittelwert von Luminanz / Chrominanz

Da die Eigenschaften eines Segmentes eine relativ kleine Datenmenge darstellen, können die Eigenschaften eines zu bearbeitenden Segments innerhalb der Verarbeitungseinheit lokal gespeichert werden und vor Verarbeitungsbeginn geladen bzw. nach Verarbeitungsende im Speicher abgelegt werden. Bei vielen Operationen werden jedoch mehrere Segmente gleichzeitig bearbeitet, d.h. die bearbeiteten Bildpunkte gehören zu unterschiedlichen Segmenten, so daß je nach Segmentzugehörigkeit eines Punktes die Eigenschaften unterschiedlicher Segmente als Eingangs- bzw. Ausgangsdaten dienen.

Da der Zugriff auf mit Segmenten assoziierte Daten einem Tabellenzugriff entspricht, bei dem die Segmentnummer die Funktion eines Tabellenindex besitzt, wird er als Segmentindizierte Adressierung bezeichnet. Auch Histogrammdaten stellen Daten dar, die nicht mit Bildpunkten sondern mit Segmenten assoziiert sind. Da sowohl Bilddaten als auch mit Segmenten assoziierte Daten gleichzeitig verarbeitet werden müssen, erfolgt segmentindizierte Adressierung parallel zu einer der Adressierungsmethoden für Bilddaten.

2.6 Bildformat und Algorithmenkomplexität

Im Bereich der Videokommunikation werden die in Tabelle 2.3 gezeigten Bildformate eingesetzt. Die Bildformate CIF und QCIF werden am häufigsten eingesetzt, so daß deren Unterstützung für den Einsatz bei Kommunikationsanwendungen in jedem Fall erforderlich ist. Für andere Anwendungen mit größeren Bildformaten ist die Unterstützung bis hin zum Bildformat 4CIF wünschenswert.

Bei Videokommunikationsanwendungen wird typischerweise der YUV Farbraum verwendet, da sich in dieser Form die Datenmenge leicht durch Unterabtastung der Farbkomponenten ohne starke Beeinträchtigung der visuellen Qualität reduzieren läßt. Ein Unterabtastung nur in x-Richtung wird als 4:2:2 Farbformat, eine Unterabtastung in x- und y-Richtung als 4:2:0 Farbformat und der Einsatz nicht unterabgetasteter Bilddaten als 4:4:4 Farbformat bezeichnet. Während Unterabtastung zur Datenreduktion bei Videocodierung sinnvoll ist, ist für Videosegmentierungsverfahren der Einsatz nicht unterabgetasteter Videodaten für die Erzielung einer hohen Genauigkeit sinnvoller. In Abhängigkeit der Randbedingungen ist auch der Einsatz anderer Farbräume wie beispielsweise RGB sinnvoll. Auch in diesen Fällen ist eine Unterabtastung nicht

zweckmäßig.

Bezeichnung	Breite	Höhe	Bildfrequenz
SQCIF	128	96	10
QCIF	176	144	10/30
CIF	352	288	30
4CIF	704	576	30
SIF	352	240[a] / 288[b]	30[a] / 25[b]

a. USA und Japan
b. Europa

Tabelle 2.3: In der Videocodierung gebräuchliche Bildformate und Bildfrequenzen

Da Segmentierungsverfahren große Datenmengen bearbeiten, und viele Verarbeitungsschritte umfassen, sind sie sehr rechenintensiv. Zur Quantifizierung der Komplexität wird der in [HeMo99b] beschriebene Farbsegmentierungsalgorithmus näher untersucht.

Sequenz Bildpunktopr.	Mother & Daughter (QCIF)	Foreman (QCIF)	Mittelwert (QCIF)	Anteil (Mittelwert)
Inter	$0{,}46 \cdot 10^6$	$0{,}46 \cdot 10^6$	$0{,}46 \cdot 10^6$	1,7%
Intra	$13{,}44 \cdot 10^6$	$27{,}34 \cdot 10^6$	$19{,}11 \cdot 10^6$	71,8%
Segment	$5{,}23 \cdot 10^6$	$8{,}84 \cdot 10^6$	$7{,}04 \cdot 10^6$	26,5%
Gesamt	$19{,}13 \cdot 10^6$	$36{,}64 \cdot 10^6$	$26{,}61 \cdot 10^6$	100%

Tabelle 2.4: Anzahl der Bildpunktoperationen bei verschiedenen Sequenzen

Sequenz Bildpunktopr.	Foreman (QCIF)	Foreman (CIF)	Foreman (QCIF 1:3)
Inter	$0{,}46 \cdot 10^6$	$1{,}82 \cdot 10^6$	$0{,}46 \cdot 10^6$
Intra	$27{,}34 \cdot 10^6$	$65{,}57 \cdot 10^6$	$11{,}23 \cdot 10^6$
Segment	$8{,}84 \cdot 10^6$	$26{,}28 \cdot 10^6$	$3{,}84 \cdot 10^6$
Gesamt	$36{,}64 \cdot 10^6$	$93{,}67 \cdot 10^6$	$15{,}54 \cdot 10^6$

Tabelle 2.5: Anzahl der Bildpunktoperationen bei verschiedenen Bildformaten bzw. beim Überspringen von Bildern

Tabelle 2.4 zeigt die Anzahl an Bildpunktoperationen für 10 Bilder des Formates QCIF. Insgesamt wurden acht MEPG-4 Testsequenzen verglichen („Akiyo“, „Bream“, „Children“, „Coastguard“, „Container Ship“, „Foreman“, „Mother&Daughter“, „Stefan“) und in Tabelle 2.4 die Sequenzen mit kleinster und größter Anzahl an Operationen und der Mittelwert insgesamt angegeben. Den mit 71,8% größten Anteil an der Gesamtzahl an Operationen hat die Intra-Adressierung, einen ebenfalls nennenswerten Anteil von 26,5% hat die Segment-Adressierung. Tabelle 2.5 zeigt am Beispiel der Foreman Sequenz, daß sich beim Bildformat CIF eine Steigerung der durchzuführenden Operationen um 156% gegenüber dem Bildformat QCIF ergibt, bzw. daß durch abwechselnde Segmentierung eines Bildes und Bewegungskompensation dreier Bilder eine um 57% geringere Anzahl an Operationen zur Durchführung ausreicht.

Neben der Anzahl der Bildpunktoperationen beeinflußt auch deren Komplexität den insgesamt erforderlichen Rechenaufwand. Zur Quantifizierung der Komplexität wird ein Instruktions-Profiling des Farbsegmentierungsalgorithmus [HeMo99b] erstellt. Von Interesse sind die Anzahl und Art von Low-Level-Operationen im Algorithmus und deren Verhältnis zum High-Level-Algorithmus. Ein weiterer Punkt ist das Verhältnis zwischen Adressierung und Verarbeitung von Bildpunkten. Als Maß für die erforderliche Speicherbandbreite ist die Art und Anzahl von Speicherzugriffen von Interesse. Für die Untersuchung wurde das Werkzeug *iprof* [M0921], [iprof] eingesetzt, das auch im Rahmen der Standardisierung von MPEG-4 verwendet wurde. Die Befehlshäufigkeiten bei Verarbeitung, Adressierung, High-Level-Algorithmus und Daten Ein-/ Ausgabe wurden getrennt analysiert, wie in Tabelle 2.6 aufgeführt ist.

Befehle	Daten Ein- / Ausgabe	High-Level Algorithmus	Adressierung	Verarbeitung	Gesamt
Laden / Speichern	7,1G / 37%	0,07G / 44%	17,0G / 51%	2,7G / 45%	26,9G / 45%
Arithm. Befehle	2,9G / 15%	0,03G / 19%	6,8G / 20%	1,6G / 27%	11,3G / 19%
Sprünge / Vergl.	5,4G / 28%	0,03G / 19%	5,3G / 16%	1,1G / 18%	11,8G / 20%
Typkonvertierung	3,5G / 18%	0,02G / 13%	3,9G / 12%	0,6G / 10%	8,0G / 14%
Sonstige	0,5G / 3%	0,01G / 6%	0,5G / 1%	0,0G / 0%	1,0G / 2%
Gesamt	19,4G / 33%	0,16G / 0,3%	33,5G / 57%	6,0G / 10%	59,1G / 100%

Tabelle 2.6: Befehlshäufigkeit für einen iterativen Farbsegmentierungsalgorithmus (MPEG-4 Sequenz Akiyo, Bild 0-4, Bildformat QCIF, Farbformat 4:4:4)

Wie aus der Tabelle 2.6 ersichtlich, ist die Komplexität für Adressierung mit $33{,}5 \cdot 10^9$ Operationen wesentlich höher als die Komplexität für Verarbeitung, die $6 \cdot 10^9$ Operationen beträgt. Es zeigt sich, daß das Verhältnis von Adressierung zu Verarbeitung bei den Low-Level-Operationen generell zwischen 3:1 und 9:1 liegt (d.h. 75% bis 90% Anteil der Adressierung). Der Anteil des High-Level-Algorithmus an der Gesamtzahl der Instruktionen ist mit 0,3% sehr klein und läßt sich aufgrund der niedrigen Werte auf einem Universalprozessor implementieren, während Adressierung und Verarbeitung mit einer Summe von $39{,}5 \cdot 10^9$ Operationen die Leistungsfähigkeit eines Universalprozessors übersteigt.

3. Architekturen für Videoobjekt-Segmentierung

In diesem Kapitel werden bekannte Architekturkonzepte und Architekturen vorgestellt, die sich zur effizienten Implementierung von Algorithmen der Videoobjekt-Segmentierung eignen. Dabei werden sowohl Architekturen untersucht, die komplette Segmentierungsalgorithmen abdecken, als auch solche, die sich zur Unterstützung von Segmentierungsalgorithmen einsetzen lassen. Ausgehend von den Anforderungen und Randbedingungen, werden zunächst bekannte Architekturkonzepte für Operationsausführung, Datenorganisation und Flexibilität diskutiert. Im Anschluß daran werden bekannte Architekturen betrachtet, und die Notwendigkeit einer angepaßten aber flexiblen Architektur herausgearbeitet, die reguläre und irreguläre Adressierung unterstützt.

3.1 Algorithmische Anforderungen und Randbedingungen

Segmentierungsverfahren zur Videoobjektgenerierung sind sehr komplex und erfordern daher eine hohe Verarbeitungsleistung. Demgegenüber stehen die Echtzeitanforderungen von Anwendungen im Kommunikations- und Überwachungsbereich und der Bedarf an schneller Verarbeitung bei anderen Anwendungen. Somit ergibt sich ein Bedarf an Architekturen zur Implementierung von Videoobjekt-Segmentierung mit hoher Verarbeitungsleistung. Aus algorithmischer Sicht ergeben sich folgende Anforderungen:

1. Um eine hohe Verarbeitungsleistung zu erzielen, müssen bildpunktbasierte Low-Level-Operationen unterstützt werden, die den größten Teil des erforderlichen Rechenaufwandes darstellen.
2. Videoobjekt-Segmentierungsverfahren enthalten sowohl Operationen, die ganze Bilder bearbeiten, als auch solche, die der Bestimmung zusammenhängender Bereiche dienen. Zur Objektverfolgung müssen Bilder verknüpft werden. Außerdem spielen Segmenteigenschaften eine wichtige Rolle. Somit müssen alle in Kapitel 2.5 aufgelisteten Adressierungsarten unterstützt werden.
3. Da Videoobjekt-Segmentierungsverfahren viele unterschiedliche Operationen beinhalten, und um ein breites Einsatzspektrum abzudecken, ist ein hohes Maß an Flexibilität erforderlich.
4. Eine weitere Anforderung ist die Unterstützung der in Kapitel 2.4 aufgelisteten Operationen. Um eine hohe Flexibilität zu ermöglichen, ist darüberhinaus anzustreben, daß auch weitere Operationen unterstützt werden, die sich mit den dafür erforderlichen Ressourcen implementieren lassen.
5. Um Einsatz im Bereich der Videokommunikation zu ermöglichen, müssen dort gebräuchliche Bildformate wie QCIF, CIF und 4CIF unterstützt werden, soweit dies die Verarbeitungsleistung zuläßt.

Neben den algorithmischen Anforderungen müssen weitere Randbedingungen berücksichtigt werden:

6. Um eine hohe Verarbeitungsleistung zu erzielen, ist eine hohe Parallelität der Operationsausführung ein entscheidender Faktor.
7. Da große Datenmengen verarbeitet werden ist eine effiziente Datenorganisation und eine schnelle Zufuhr großer Datenmengen erforderlich.
8. Da die Herstellungskosten eines Chips stark vom Flächenbedarf abhängen (Ausbeute, Gehäusekosten), ist eine möglichst geringe Chipfläche anzustreben. Auch für mobile Anwendungen ist eine kompakte Implementierung anzustreben.
9. Für mobilen Einsatz ist ein möglichst geringer Leistungsverbrauch wichtig. Der wesentliche Anteil der Verlustleistung in CMOS Schaltungen ist proportional zur Häufigkeit der Schaltvorgänge und zum Quadrat der Versorgungsspannung. Hohe Parallelität ist anzustreben, da sie

(bei unveränderter Anzahl an Schaltvorgängen) eine niedrigere Taktfrequenz und damit die Absenkung der Versorgungsspannung ermöglicht.

10. Weitere Faktoren, die den Leistungsverbrauch beeinflussen, müssen berücksichtigt werden. Insbesondere müssen unnötige Schaltvorgänge bei Speicherzugriffen und bei der Operationsausführung vermieden werden.

3.2 Bekannte Architekturkonzepte

Bevor auf bekannte Architekturen eingegangen wird, werden die zugrundeliegenden Architekturkonzepte für die Operationsausführung, Datenorganisation und Flexibilität betrachtet. Zunächst wird auf die Arten möglicher Datenabhängigkeiten und daraus resultierende Konzepte zur Operationsausführung und Parallelisierung eingegangen. Anschließend werden Datenorganisation, Speicherhierarchie und paralleler Zugriff betrachtet, die wichtige Faktoren für eine kompakte Speicherung und eine schnelle Datenzufuhr darstellen. Vor dem Hintergrund der Forderung nach hoher Flexibilität werden ferner die Architekturkonzepte Konfigurierbarkeit und Programmierbarkeit erläutert und miteinander verglichen.

3.2.1 Operationsausführung

Um eine hohe Verarbeitungsleistung zu erreichen, ist es erforderlich, Berechnungen zeitlich parallel durchzuführen. Dazu müssen mehrere Verarbeitungseinheiten eingesetzt werden, die gleichzeitig operieren können. Um eine effiziente Ressourcennutzung und eine schnelle Operationsausführung zu erreichen, ist die effiziente Nutzung der im Algorithmus vorhandenen Parallelität notwendig. Neben der Erreichung einer hohen Beschleunigung ist auch die Skalierbarkeit hinsichtlich der bearbeiteten Datenmenge bzw. Bildgröße ein weiterer Grund für Parallelisierung.

3.2.1.1 Datenabhängigkeit und Parallelität

Der zur Lösung einer Aufgabenstellung verwendete Algorithmus definiert die Art und Anzahl der auszuführenden Operationen. Ist der Algorithmus festgelegt und werden bei der Implementierung keine Algorithmentransformationen angewandt, dann ist auch die Hardware-Architektur hinsichtlich der auszuführenden Operationen festgelegt. Für deren Abfolge bestehen jedoch Freiheitsgrade.

Welche Freiheitsgrade dies sind, wird durch die Anzahl und Struktur der Datenabhängigkeiten bestimmt. Wenn zwischen zwei Operationen eine Datenabhängigkeit besteht, müssen diese in einer festen Reihenfolge ausgeführt werden, da sich sonst das Endergebnis beider Operationen ändert. Besteht keine Datenabhängigkeit, läßt sich die Reihenfolge willkürlich wählen bzw. beide Operationen können gleichzeitig ausgeführt werden, wenn hierfür ausreichend Verarbeitungsressourcen vorhanden sind. Datenabhängigkeiten können in drei Klassen unterteilt werden [Brä93]:

- Flow-Dependence: Eine Operation benutzt ein Ergebnis der vorherigen Operation.
- Anti-Dependence: Eine Operation schreibt ein Ergebnis in eine Speicherzelle, die einen Wert enthält, der von einer vorherigen Operation benutzt wurde.
- Output-Dependence: Eine Operation überschreibt das Ergebnis einer vorherigen Operation.

Anti-Dependence und Flow-Dependence kommen durch die Zuordnung der Daten zweier Operationen zur gleichen Speicherzelle zustande. Sie sind somit implementierungsbedingt und können durch Einsatz einer zweiten Speicherzelle eliminiert werden. Die Flow-Dependence resultiert hingegen aus der Weiterverwendung eines Operationsergebnisses. Sie ist durch den Algorithmus bedingt und kann nicht eliminiert werden.

3.2.1.2 Implementierung von Parallelität

Bei der Implementierung von Parallelität wird gewöhnlich zwischen Datenparallelität und Befehlsparallelität unterschieden. Während mit Datenparallelität die parallele Anwendung von Operationen des gleichen Typs auf mehrere Daten bezeichnet wird, erfolgt bei Befehlsparallelität die parallele Ausführung unterschiedlicher Operationen. Bei Prozessoren wird Befehlsparallelität in der Regel in *Instruction-Level-Parallelität* (ILP) und *Task-Level-Parallelität* (TLP) unterteilt (siehe Kapitel 3.3.2.1).

Zur Charakterisierung von Systemen - insbesondere von Prozessorsystemen - ist die in Tabelle 3.1 gezeigte Klassifikation nach Flynn verbreitet [Fly66]. Sequentielle Ausführung wird als SISD (Single Instruction Single Data) bezeichnet. Wenn ein System so aufgebaut ist, daß dieselben Daten der Reihe nach eine Pipeline durchlaufen spricht man von Pipelining bzw. MISD (Multiple Instruction Single Data). Auf der rechten Seite des Diagramms finden sich SIMD (Single Instruction Multiple Data) und MIMD (Multiple Instruction Multiple Data), zwei Arten der Parallelisierung für mehrere Daten.

	Single Data	**Multiple Data**
Single Instruction	SISD Sequentiell	SIMD Parallelisierung
Multiple Instruction	MISD Pipelining	MIMD Parallelisierung

Tabelle 3.1: Klassifikation nach Flynn

Pipelining ist besonders vorteilhaft, wenn ein Algorithmus viele sequentiell abhängige Operationsabfolgen ohne Verzweigung enthält, während Parallelisierung sich vorteilhaft einsetzen läßt, wenn im Algorithmus viele aus Sicht des Datenflusses unabhängige Operationen durchgeführt werden. Wenn es sich um viele gleichartige Operationen handelt, eignen sich SIMD Ansätze am besten, bei unterschiedlichen Operationen hingegen MIMD Ansätze.

Für eine effiziente Beschleunigung eines Verfahrens ist es notwendig, die enthaltene Parallelität vollständig zu nutzen. Dazu muß die enthaltene Parallelität nach Art, Grad und Parallelitätsebenen analysiert werden, um einen effizienten Ansatz zur Parallelisierung bzw. Pipelining zu bestimmen.

3.2.1.3 Ahmdahlsches Gesetz

Aufgrund unterschiedlicher Eigenschaften von Algorithmenteilen läßt sich durch Parallelisierung und/oder Pipelining in der Regel keine gleichmäßige Beschleunigung erzielen. Besonders problematisch sind Algorithmenteile, die Parallelisierung und Pipelining gänzlich unzugänglich sind, da diese in einer Obergrenze für die Beschleunigung des Gesamtsystems resultieren.

Das Ahmdahlsche Gesetz beschreibt den Zusammenhang zwischen dem Grad p der Parallelisierung bzw. Pipelining, dem Anteil f der parallelisierbaren bzw. pipelinebaren Operationen am Gesamtsystem und der insgesamt erreichbaren Beschleunigung S [HePa90]. Es lautet:

$$S = \frac{1}{(1-f) + f/p} \tag{3.1}$$

Wie aus Gleichung (3.1) ersichtlich, wächst die insgesamt erreichbare Beschleunigung mit zunehmendem Parallelisierungs- bzw. Pipelinegrad immer langsamer. Während p prinzipiell unendlich groß werden kann, beträgt S maximal 1/(1-f).

3.2.1.4 Parallelitätsebenen

Bei Algorithmen der Videosignalverarbeitung ist aufgrund der Struktur der verarbeiteten Daten bereits eine Struktur der Datenabhängigkeiten gegeben, die aus mehreren verschachtelten Ebenen besteht, innerhalb deren Potential für Parallelität gegeben ist:

- Bitoperation
- Datenwortoperation
- Komponentenoperation
- Bildpunktoperation
- Bildoperation
- Verarbeitung eines Bildes
- Verarbeitung einer Sequenz

Datenwortoperationen setzen sich durch mehrfache Ausführung von Bitoperationen zusammen, jeweils eine pro Bitposition der verarbeiteten Datenworte. Komponentenoperationen entstehen durch Verkettung einer oder mehrerer Datenwortoperationen, so daß insgesamt die Verarbeitung einer Helligkeits- oder Farbkomponente erfolgt. Werden mehrere Komponentenoperationen kombiniert, ergibt sich eine Bildpunktoperation und deren wiederholte Ausführung für ein gesamtes Bild ergibt eine Bildoperation. Zur Verarbeitung eines Bildes werden i.d.R. mehrere Bildoperationen verkettet und für die Verarbeitung einer Sequenz wird i.d.R. die Verarbeitung für jedes Bild der Sequenz wiederholt.

Die Ebenen Bitoperation und Datenwortoperation sind elementarer Natur und treten in fast allen Algorithmen auf. Die Ebenen Komponentenoperation und Bildpunktoperation korrespondieren zu Low-Level-Operationen im Algorithmus, die Ebene Bildoperation zu Mid- bzw. High-Level-Operationen und die Verarbeitung eines Bildes bzw. einer Sequenz zu High-Level-Operationen.

Parallelität besteht auf diesen Ebenen je nach Algorithmus in unterschiedlichem Maße. Bei Algorithmen der Videosignalverarbeitung ist eine Parallelisierung auf Bitebene praktisch immer möglich, außer wenn binäre Daten verarbeitet werden. Sie erfolgt meist implizit, da Datenworte in der Regel als Einheit verarbeitet werden.

Ob eine gleichzeitige Verarbeitung auf Datenwortebene durch überlappende Ausführung möglich ist, hängt vom Low-Level-Algorithmus ab. Sie ist unter verschiedenen Voraussetzungen möglich, z.B. wenn keine Datenabhängigkeit zu Nachbarpunkten durch rekursive Verarbeitung besteht, wenn Datenabhängigkeiten durch die Verarbeitungsabfolge aufgebrochen werden, oder wenn Operationen bei Hardware-Architekturen innerhalb eines Taktes verkettet werden können.

Komponentenoperationen können je nach Low-Level-Algorithmus aus unabhängigen Operationen für Helligkeits- und Farbkomponenten bestehen, oder eine Kombination der Ergebnisse unabhängiger Teiloperationen beinhalten. Auf dieser Ebene besteht typischerweise nur eingeschränkte Regularität, die von der Art des Low-Level-Algorithmus abhängt.

Auf der Bildebene ist das Parallelisierungspotential aufgrund der großen Anzahl an Bildpunkten extrem hoch und in erster Linie davon abhängig, ob der Algorithmus rekursiv oder nicht rekursiv arbeitet. Bei nichtrekursiven Algorithmen besteht das Parallelisierungspotential unabhängig von der Verarbeitungsabfolge, bei rekursiven Algorithmen besteht i.d.R. Potential für eine eingeschränkte Parallelisierung, je nach Form der Nachbarschaft der verwendeten Eingangsdaten.

Parallelisierung auf den zuvor genannten Ebenen kann gegenüber der Algorithmenentwicklung verborgen bleiben, wenn die Bildoperationen jeweils eine Einheit darstellen, deren genaue Implementierung keine Auswirkung auf den Algorithmus hat. Bei der Verarbeitung eines Bildes besteht jedoch ebenfalls großes Parallelisierungspotential, beispielsweise können verkettete Bildoperationen zeitlich überlappend ausgeführt werden bzw. unabhängige Bildoperationen zeitlich parallel. Die Größe dieses Potentials ist algorithmenabhängig, die Nutzung dieser Parallelität erschwert jedoch die Entwicklung bzw. die Änderung eingesetzter Verfahren.

Die Parallelisierung auf Sequenzebene ist ebenfalls von der Art des eingesetzten Algorithmus abhängig. Sie ist möglich, solange keine sich fortpflanzenden Datenabhängigkeiten zwischen den Bildern bestehen. Eine Parallelisierung bzw. Pipelining auf dieser Ebene erhöht jedoch i.d.R. die Verarbeitungslatenz in einem Maß, das keine Echtzeitverarbeitung von Daten mehr erlaubt, obwohl die Verarbeitungsgeschwindigkeit auf diese Weise deutlich gesteigert werden kann.

Neben der streng hierarchischen Schachtelung der Ebenen kann auch eine Vertauschung der Operationsabfolge über die Ebenen hinweg erfolgen. Beispiele hierfür sind:

- Separate Operationsausführung für die Bitebenen
- Separate Operationsausführung für Helligkeits- bzw. Farbkomponenten
- Verkettung mehrerer Bildpunktoperationen

Die Vertauschung kann günstige Auswirkungen auf die Parallelisierbarkeit haben, und z.B. die Anpassung an massiv parallele Architektur gestatten. Im Fall der Verkettung von Bildpunktoperationen werden High-Level- und Low-Level-Operationen im Algorithmus miteinander vermischt, was hinsichtlich der Flexibilität der Algorithmenentwicklung und der Einsatzmöglichkeiten des Algorithmus ungünstig ist, jedoch aus Sicht der Reduzierung von Speichertransfers vorteilhaft sein kann.

Bei Algorithmen der Videosegmentierung und -analyse kommt zu obigen Aspekten eine zusätzliche Ebene hinzu. Es handelt sich um die Segmentebene, die (vergleichbar mit der Blockebene bei Videocodierung) zwischen Bildpunktebene und Bildebene lokalisiert ist. Da die Form von Segmenten und damit die Anzahl und Abfolge zu verarbeitender Bildpunkte von der datenabhängigen Segmentgröße bestimmt wird, besteht auf dieser Ebene keine Regularität. Während eines Ausbreitungsprozesses ist aufgrund dieser Parallelitätsform die gleichzeitige Verarbeitung aller Bildpunkte innerhalb der Ausbreitungsfront möglich.

3.2.2 Datenorganisation

Große, schnell zu verarbeitende Datenmengen bei Videosegmentierungsverfahren erfordern Konzepte zur Datenorganisation, die schnellen Zugriff mit großer Bandbreite ermöglichen. Außerdem müssen die Datenorganisation relevante Zugriffsmuster und -charakteristika berücksichtigen und die Speicherung großer Datenmengen gestatten. Von technologischer und architektureller Seite her sind diese Anforderungen jedoch teilweise gegenläufig, insbesondere besteht zwischen Zugriffsgeschwindigkeit und Speichergröße und zwischen Zugriffsgeschwindigkeit und wahlfreiem Zugriff ein Gegensatz.

3.2.2.1 Speicherelemente und Zugriffseigenschaften

Zur Datenspeicherung in Hardware-Architekturen stehen unterschiedliche Möglichkeiten zur Verfügung. Bei Latches und Registern handelt es sich um einzelne Speicherzellen, die aus Standardzellen aufgebaut sind, während es sich bei Speicherblöcken um Module zur Speicherung größerer Datenmengen handelt, deren Grundelemente mittels Full-Custom-Entwurf erstellt werden, und die in der Regel entsprechend vorgegebener Parameter aus den Grundelementen automatisch generiert werden.

Register werden typischerweise mit D-Flip-Flops implementiert, die auf Taktflanken sensitiv sind. Latches benötigen eine geringere Fläche als Register, ihr Einsatz ist jedoch mit sehr hohem Entwurfsaufwand verbunden, da derzeitige Entwurfswerkzeuge den Einsatz zweier nicht überlappender Takte in einem Modul nicht unterstützen. Wenn die Taktfrequenz der Schaltung einen Minimalwert nicht unterschreitet und ausreichend häufige Schreibzugriffe sichergestellt sind, können dynamische Register eingesetzt werden, die ebenfalls platzsparender als statische Register sind. Die zweite Voraussetzung ist jedoch i.d.R schwierig zu verifizieren bzw. bei vielen Anwendungsfällen wie z.B. Konfigurationsdaten und Betriebsmodi nicht gegeben.

Speicherblöcke werden in statische Speicher (SRAM) und dynamische Speicher (DRAM) unterteilt. Eine statische Speicherzelle besteht aus zwei Adreßtransistoren, über welche auf die Speicherzelle zugegriffen wird, und einem bistabilem Flip-Flop, das mit vier p-Kanal-Transistoren oder platzsparender mit zwei Transistoren vom Verarmungstyp ausgeführt sein kann [Sha97]. Eine dynamische Speicherzelle benutzt einen Kondensator zur Datenspeicherung anstelle des Flip-Flops, was zu einer deutlich höheren Speicherdichte führt, typischerweise in der Größenordnung von Faktor drei [Wid96]. Während bei statischen Speicherzellen der Dateninhalt solange erhalten beleibt, wie die Versorgungsspannung anliegt, entladen sich sie Kondensatoren in dynamischen Speichern mit der Zeit, so daß ein regelmäßiges Auffrischen (Refresh) des Speicherinhaltes notwendig ist, typischerweise mit einer Periode von einigen hundert Mikrosekunden bis einigen hundert Millisekunden.

Zur Nettofläche der Speicherzellen muß die anteilige Fläche für Adressierung und Datenzugriff hinzugerechnet werden, um die Bruttofläche zu erhalten. Der Mehraufwand hierfür fällt jedoch um so schwächer ins Gewicht, je größer die Speicherkapazität ist. Für Register ist ein höherer Flächenaufwand erforderlich als für eine Speicherzelle gleicher Kapazität, da Register - wie in Tabelle 3.2 gezeigt - aus einer größeren Anzahl an Transistoren bestehen, und da Standardzellen eine weniger kompakte Implementierung als Full-Custom-Entwurf ermöglichen.

Art der Speicherzelle	Transistorzahl
Register	13
Latch	8
SRAM Zelle	6 / 4 [a]*)
DRAM Zelle	1

a. p-Kanal Transistor / Verarmungstyp-Transistor

Tabelle 3.2: Vergleich von Latch, Register und Speicherzellen

Speicherblöcke können entweder extern implementiert oder auf einem Chip integriert werden. Große Mengen an Speicher lassen sich extern am leichtesten implementieren, da auf Standardprodukte zurückgegriffen werden kann, die in großen Stückzahlen kostengünstig mit hochoptimierter Technologie hergestellt werden. Nachteilig ist jedoch, daß nur auf Standardgrößen zurückgegriffen werden kann und daß Signale über Chipgrenzen laufen und dadurch die Zugriffe mit geringer Geschwindigkeit erfolgen. Ein auf dem Chip integrierter Speicher, sogenanntes Embedded RAM (eRAM), vermeidet diese Nachteile. Neben schnelleren Zugriffen sind breitere Busse, größere Bandbreiten und eine fast beliebige feine Speicheraufteilung möglich. Embedded SRAM läßt sich bei Verwendung gängiger Logik-Technologien integrieren, beim Einsatz von Embedded DRAM muß jedoch eine spezielle Technologie verwendet werden, die sich sowohl für Logik (hohe Geschwindigkeit) als auch für Speicher (geringe Leckströme) eignet und somit einen Kompromiß zwischen Logikgeschwindigkeit und Refresh-Rate darstellt.

Neben ihrem Flächenbedarf unterscheiden sich Register und Speicherblöcke auch in Zugriffsmöglichkeiten, Latenz und Bandbreite. Register können jederzeit ausgelesen *und* parallel einmal pro Taktzyklus beschrieben werden. Speicher hingegen (abgesehen von weiter unten beschriebenen Multi-Port-RAMs) werden pro Taktzyklus (bei synchronen RAMs) bzw. innerhalb eines Lese- bzw. Schreibzyklus (bei asynchronen RAMs) entweder gelesen oder beschrieben. Während beim Zugriff auf Register nur kurze Setup- und Hold-Zeit einzuhalten sind, sind die Zugriffszeiten bei SRAMs und bei DRAMs vergleichsweise lange.

Während bei SRAMs der Inhalt einer Speicherstelle durch einen Lesezugriff nicht beeinflußt wird, fließt bei DRAMs die auf den Kapazitäten der zugegriffenen Zellen gespeicherte Ladung ab und muß nach Abschluß eines Zugriffes wieder zurückgeschrieben werden. Da Speicher in der Regel matrixförmig aufgebaut sind, wird bei DRAMs immer eine Zeile der Speichermatrix zugegriffen (die auch als Speicherseite bzw. Page bezeichnet wird). Aufgrund des komplizierteren Vorgangs sind die Zugriffszeiten bei DRAMs deutlich länger als bei SRAMs.

Um lange Zugriffszeiten von DRAMs abzumildern wurden verschiedenste architekturelle Maßnahmen eingeführt, während sich die Geschwindigkeit der DRAM-Matrix nur leicht verbessert hat [Sha97]. Inzwischen hat sich eine Vielfalt von DRAM-Typen entwickelt, von denen Page Mode DRAM, EDO DRAM (Extended Data Output DRAM) und SDRAM (Synchronous DRAM) die bekanntesten sind. Trotz der Typenvielfalt lassen sich die unterschiedlichen DRAMs durch ähnliche Eigenschaften charakterisieren. Im Gegensatz zu SRAMs, bei denen ein wahlfreier Zugriff auf alle Speicherstellen mit voller Geschwindigkeit möglich ist, ist bei DRAMs nur innerhalb einer Speicherseite ein wahlfreier Zugriff mit vergleichbarer Geschwindigkeit möglich, für den Seitenwechsel ist eine fünf bis zehnmal höhere Zeit zu veranschlagen. Oftmals ist der Datenpfad von DRAMs als Pipeline aufgebaut, so daß beim Lesen von Daten eine Pipelineverzögerung zu berücksichtigen ist.

3.2.2.2 Speicherhierarchie

In der Videosignalverarbeitung werden DRAMs zur Speicherung großer Datenmengen eingesetzt, da sie wesentlich weniger Fläche erfordern als SRAMs gleicher Kapazität. Zugriffe auf DRAMs erfolgen in Form von sogenannter Burst-Zugriffe, d.h. es wird nacheinander auf mehrere aufeinanderfolgende Speicherzellen zugegriffen, um eine hohe Bandbreite zu erzielen. Wenn hingegen eine geringe Zugriffszeit im Vordergrund steht und schnelle wahlfreie Zugriffe erforderlich sind, wird SRAM zur Datenspeicherung eingesetzt.

Großen Datenmengen müssen chipextern gespeichert werden, wenn der Flächenbedarf des Speichers zu groß für die Integration ist. Allerdings ist die Bandbreite und Anzahl der Schnittstellen, die über die Chipgrenze hinweg führen können, begrenzt durch die maximal implementierbare Pinzahl der eingesetzten Gehäusetechnologie. Darüberhinaus ist die Zugriffsgeschwindigkeit auf externe Speicher geringer als auf interne Speicher, da die Signale bei Schreibzugriffen einmal und bei Lesezugriffen zweimal über Chipgrenzen hinweg laufen. Signale, die über Chipgrenzen hinweg führen, laufen über einen Treiber, der die große Kapazität der externen Leitung lädt bzw. entlädt, und über eine Eingangszelle mit Schwellwertcharakteristik, so daß sich insgesamt eine lange Verzögerung ergibt. Kleine Datenmengen lassen sich chipintern speichern, mit dem Vorteil, daß der Datenbus zum Speicher wesentlich breiter dimensioniert werden kann als bei externem Speicher und daß keine Chipgrenzen überwunden werden müssen. Damit ergibt sich eine deutlich höhere Bandbreite und eine wesentlich kleinere Zugriffszeit. Die Anforderungen, einerseits große Datenmengen zu speichern und andererseits große Bandbreiten und kurze Zugriffszeiten zu erzielen, erweisen sich somit als gegensätzlich.

Bei viele Datenzugriffen bestehen jedoch Lokalitätseigenschaften, d.h. gerade zugegriffene Daten werden sehr wahrscheinlich erneut zugegriffen (zeitliche Lokalität) und es werden zukünftig sehr wahrscheinlich Daten aus der Umgebung zugegriffen (örtliche Lokalität). Bei Bilddaten ist örtliche Lokalität beispielsweise beim Einsatz von nachbarschaftsbasierten Operationen gegeben. Weitere Lokalität ist z.B. durch eine reguläre Verarbeitungsabfolge bzw. durch Verarbeitung benachbarter Bildpunkte bei Ausbreitungsprozessen gegeben.

Lokalität kann genutzt werden, um Speicherzugriffe zu reduzieren, indem häufig genutzte Daten in einem Pufferspeicher abgelegt werden, der kleiner als der Hauptspeicher ist. Dadurch kann er intern implementiert werden und es können schnelle aber flächenaufwendige Speichertypen eingesetzt werden, während der Hauptspeicher (bzw. Hintergrundspeicher) extern mit langsamen

aber flächeneffizienten Speichertypen implementiert werden kann. Der Pufferspeicher reduziert die Anzahl der Zugriffe auf den Hintergrundspeicher. Zusätzlich werden aktuell für die Verarbeitung benötigte Daten in Registern abgespeichert, um für sofortige parallele bzw. wiederholte Verwendung bereitzustehen. Somit ergibt sich typischerweise die in Bild 3.1 gezeigte dreistufige Speicherhierarchie, bei der in Richtung Verarbeitung die Zugriffsgeschwindigkeit und Bandbreite steigt, während in umgekehrter Richtung die Flächeneffizienz und Speicherkapazität steigt.

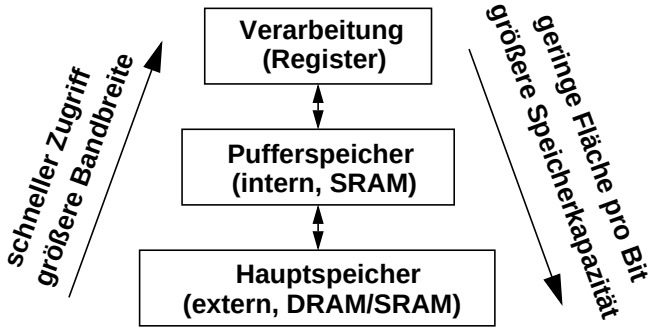


Bild 3.1: Das Prinzip der Speicherhierarchie

3.2.2.3 Arten von Pufferspeichern

Um die Anzahl von Zugriffen wirksam zu reduzieren, muß die Operation von Pufferspeichern den vorhandenen Lokalitätseigenschaften entsprechen, d.h. je nach Art der Lokalität müssen unterschiedliche Arten von Pufferspeichern eingesetzt werden.

Bei räumlich lokalen Bildoperationen, bei denen Daten innerhalb einer Nachbarschaft als Eingangsdaten für einen Verarbeitungsschritt benutzt werden und alle Bildpunkte in einer regulären Abfolge verarbeitet werden, ist eine gut nutzbare zeitliche und räumliche Lokalität gegeben. Wenn eine Nachbarschaft punktweise verschoben wird, erscheinen die Daten nacheinander an allen Positionen innerhalb der Nachbarschaft. Ohne einen Pufferspeicher muß auf jeden Punkt n-mal zugegriffen werden, wenn n Punkte in der Nachbarschaft enthalten sind. Da die Nachbarschaft (abgesehen von Zeilenwechseln) jeweils um einen Bildpunkt verschoben wird, ist es jedoch ausreichend, alle von der Nachbarschaft neu überdeckten Punkte zu laden und Punkten, die auch im vorherigen Schritt bereits von der Nachbarschaft überdeckt wurden, durch Verschieben der Daten innerhalb der Nachbarschaft neue Werte zuzuweisen. Dies ist in Bild 3.2 (a) gezeigt, wobei die eingezeichneten Pfeile das Ziel (Pfeilspitze) und den Ursprung (Pfeilanfang) von Datentransfers angeben.

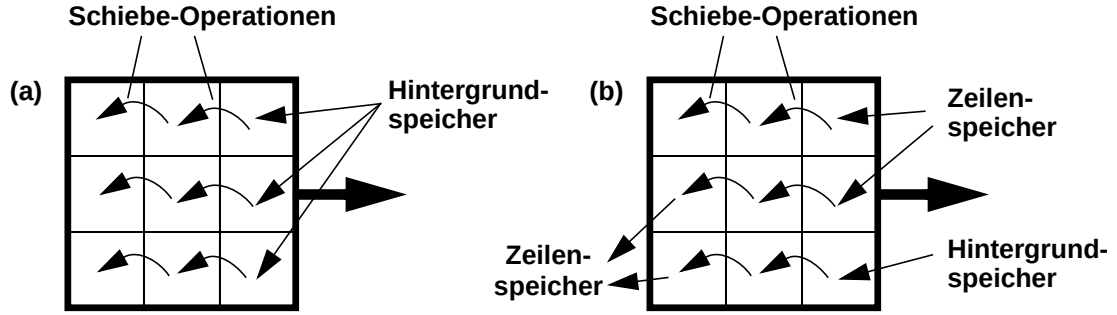


Bild 3.2: Mehrfache Nutzung von Bildpunkten mittels Schiebe-Operationen bei horizontaler Verschiebung einer CON8 Nachbarschaft

Da die Bilddaten (bei zeilenweiser Verarbeitungsabfolge) nach Bearbeitung einer Zeile wieder benötigt werden, können Zeilenspeicher eingesetzt werden, die nach dem FIFO-Prinzip arbeiten [Sch94], [PIMM1]. Wie in Bild 3.2 (b) gezeigt, werden die Daten, die beim Verschieben aus dem Bereich des Fensters fallen, dann in dem Zeilenspeicher solange zwischenspeichert, bis sie in der nächsten Zeile wieder in den Bereich des Fensters gelangen. Wenn die Bildweite und die Fensterhöhe, deren Produkt die vom Zeilenspeicher aufzunehmende Datenmenge bestimmt, klein genug sind, wird diese Lösung eingesetzt, ansonsten wird die Technik des Schiebens der Nachbarschaft und des Ladens der neu hinzugekommenen Bildpunkte verwendet.

Bei Verfahren der Videosignalverarbeitung werden auch häufig rechteckige Bildausschnitte verarbeitet, insbesondere bei blockbasierten Verfahren. In diesen Fällen kann anstelle einer automatischen Verwaltung des Pufferspeichers das Laden bzw. Speichern seines Inhaltes expliziter Programmkontrolle überlassen werden, gegebenenfalls unterstützt durch spezielle Einheiten für zweidimensionale Blocktransfers [C80]. Dies ist universeller, da keine auf die Art der Zugriffe abgestimmten Einheiten benötigt werden. Da jedoch keine automatische Steuerung erfolgt, ist sowohl für den Entwurf als auch zur Laufzeit erheblicher Aufwand für die Steuerung nötig.

3.2.2.4 Paralleler Datenzugriff

Insbesondere bei Pufferspeichern ist die Möglichkeit zu parallelem Datenzugriff ein wichtiges Kriterium, wenn diese Daten für parallele Operationsausführung mit sehr hoher Datenrate liefern müssen. Paralleler Datenzugriff kann auf verschiedene Arten implementiert werden. Ein Weg ist die Erhöhung der Wortbreite beim Einsatz nur eines Speichers. Der erhöhte Aufwand für zusätzliche Datenleitungen und Logik ist minimal, verglichen mit einem nicht parallel zugreifbaren Speicher gleicher Größe. Ein abgespeichertes Datum kann jedoch nur als Ganzes zugegriffen werden. Dadurch ergibt sich eine starre Festlegung der parallel zugreifbaren Daten, was in vielen Anwendungsfällen nicht ausreicht.

Eine weitere Möglichkeit ist die Aufteilung der Daten auf mehrere Speicher. Da die Speicher separat adressiert werden, ergibt sich nun ein höherer Aufwand als zuvor, der jedoch im Verhältnis zur Speichergröße immer noch gering ist. Jedes Datum wird einer parallel zugreifbaren Menge - und damit einer Speicherbank - zugeordnet. Da die zugegriffenen Daten aus verschiedenen Mengen beliebig kombiniert werden können, lassen sich alle Aufgabenstellungen lösen, bei denen eine konfliktfreie Mengenzuordnung gefunden werden kann.

Ein dritter Weg ist der Einsatz von parallel zugreifbaren Speichern, sogenannten Multi-Port-RAMs, die üblicherweise beliebige parallele Zugriffe über verschiedene Ports gestatten, abgesehen von Zugriffen auf dieselbe Speicherzelle. Obwohl Multi-Port-RAMs am universellsten einsetzbar sind, besteht aus Sicht der Implementierung das Problem, daß für jeden zusätzlichen Port eine Matrix an Leitungen über dem Speicher-Array und zusätzliche Adreßtransistoren implementiert werden müssen. Bei Dual-Port-RAMs sind z.B. acht statt sechs Transistoren pro Speicherzelle erforderlich. Aus diesem Grund ist diese Lösung aufwendiger und wegen der größeren Fläche langsamer als die vorherigen. Sie ist somit nur in den Fällen sinnvoll, in denen die vorherigen Lösungen nicht einsetzbar sind.

3.2.3 Flexibilität

Die Unterstützung von Algorithmen zur Videoobjekt-Segmentierung erfordert Architekturen, die ein breites Spektrum an Operationen abdecken. Im Gegensatz zu dedizierten Architekturen bieten flexible Architekturen die Möglichkeit, die Operation auch nach dem Architekturentwurf, d.h. vor oder während der Betriebszeit zu ändern. Flexibilität spielt dabei eine um so größere Rolle, je breiter das Spektrum der abzudeckenden Operationen ist. Hardware-Architekturen bieten jedoch keine Flexibilität, sofern nicht durch spezielle Maßnahmen beim Entwurf dafür gesorgt wird. Wichtige Konzepte für die Implementierung von Flexibilität, auf die im folgenden

eingegangen wird, sind Programmierbarkeit und Konfigurierbarkeit.

3.2.3.1 Programmierbarkeit und Konfigurierbarkeit

Bei konfigurierbaren Architekturen erfolgt die Implementierung der Flexibilität durch räumliches Zusammensetzen von Grundelementen, bei programmierbaren Architekturen hingegen durch zeitliches Zusammensetzen von Operationen [dHo00]. Bei universellen Architekturen spricht man von konfigurierbaren Computern im Falle konfigurierbarer Flexibilität bzw. von Prozessoren im Falle programmierbarer Flexibilität. Beide Konzepte lassen sich jedoch nicht scharf voneinander abgrenzen, es gibt Mischformen, wie z.B. rekonfigurierbare Systeme, die zur Betriebszeit umkonfiguriert werden. Je nachdem, ob die Umkonfiguration wesentlich seltener als die Operationen erfolgt, oder in ähnlicher Häufigkeit, ist die Nähe zu rekonfigurierbaren Systemen oder zu programmierbaren Systemen stärker ausgeprägt.

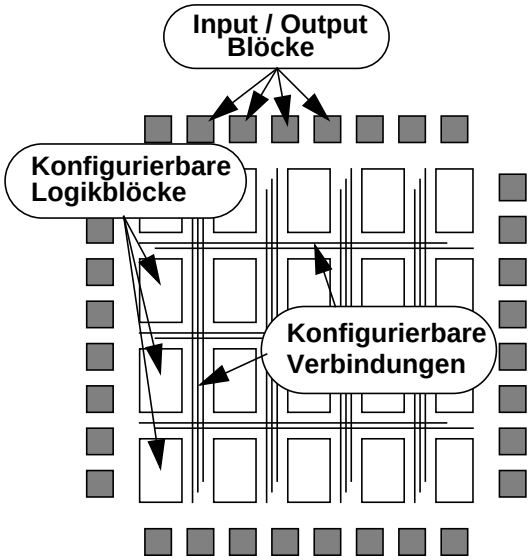


Bild 3.3: Schematischer Aufbau eines FPGAs

Typische Vertreter konfigurierbaren Systeme sind FPGAs [BrFr92]. Diese bestehen aus Logikressourcen, I/O-Ressourcen und Verdrahtungsressourcen, wie in Bild 3.3 gezeigt. Logikressourcen sind typischerweise Grundelemente, die aus einer Lookup-Table mit nachgeschaltetem Flip-Flop bestehen. Durch Konfiguration wird die Funktion der Grundelemente und deren Verschaltung festgelegt. Die Konfiguration erfolgt entweder mittels Antifuse-Technik, EPROM, EEPROM, Flash-Speicher oder SRAM. FPGAs, die auf Antifuse-Technik beruhen, sind nur einmal programmierbar, FPGAs, die Speicherzellen zur Konfiguration einsetzen, können wiederholt programmiert werden. SRAM basierte FPGAs verbrauchen vergleichsweise viel Energie und sind für mobile Anwendungen ungeeignet [Fre00].

Typische Vertreter programmierbarer Systeme sind Prozessoren. Sie bestehen, wie in Bild 3.4 gezeigt, aus einer oder mehreren Funktionseinheiten (FE) zur Durchführung von Operationen, die je nach Einsatzgebiet des Prozessors in Art, Anzahl und Wortbreite variieren. Zur Datenspeicherung wird typischerweise ein Registerblock eingesetzt, der über mehrere Pfade mit den Funktionseinheiten verbunden ist. Die Datenzufuhr erfolgt bei vielen Prozessoren über einen Daten-Cache, um die große benötigte Bandbreite bereitzustellen. Neben universell einsetzbaren Prozessoren gibt es für Signalverarbeitungsaufgaben bzw. für spezielle Anwendungen angepaßte Prozessoren. Bei Prozessoren ist eine kontinuierliche Befehlszufuhr nötig, um einen fortlaufenden Betrieb zu gewährleisten. Die Befehle müssen dekodiert und in Steuersignale für die Datenpfade umgesetzt werden. Um die für die Befehlszufuhr nötige Bandbreite zu reduzieren, werden

Befehls-Caches eingesetzt. Je nach Art der Architektur bzw. des Anwendungsgebietes wird mit Mikroprogrammen, statischer oder dynamische Zuordnung der Befehle zu Funktionseinheiten bzw. mit Befehlskompression gearbeitet.

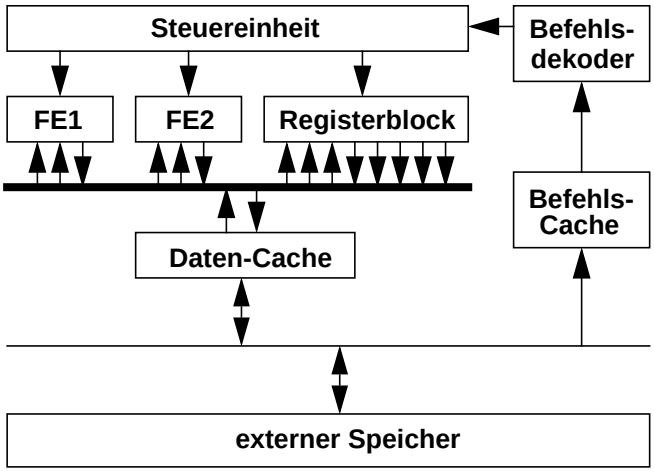


Bild 3.4: Schematischer Aufbau eines Prozessors

3.2.3.2 Zusatzaufwand für Flexibilität

Flexibilität verursacht gegenüber dedizierten Lösungen einen Mehraufwand, sowohl im Falle von Flexibilität durch Programmierbarkeit als auch im Falle von Flexibilität durch Konfigurierbarkeit. Universell konfigurierbare Chips, sogenannte FPGAs benötigen eine um 10- bis 20-mal größere Fläche wie festverdrahtete Logik [BrFr92]. Dadurch ergibt sich eine größere Fläche, längere Leitungen, höherer Leistungsverbrauch und niedrigere Taktfrequenz. Außerdem ist die Komplexität implementierbarer Algorithmen durch die Größe der eingesetzten FPGAs begrenzt. Beim Architekturentwurf muß beachtet werden, daß typischerweise nur die Hälfte des angegebenen Taktes erreicht wird, und die Ausnutzung der Ressourcen im Bereich 50%-85% liegt, so daß üblicherweise eine 50%ige Nutzung als konservative Abschätzung angenommen wird [Fre00].

Dedizierte Verarbeitungseinheiten in Prozessoren sind zwar wesentlich schneller und dichter gepackt als vergleichbare FPGA-Implementierungen, bei Prozessoren muß jedoch die Beschreibung umfangreicher Berechnungen in Form von Instruktionen auf dem Chip zwischengespeichert, dekodiert und verarbeitet werden, so daß auch hier ein hoher Zusatzaufwand erforderlich ist. Bei der Annahme, daß ein konfigurierbarer Logikblock eine Bitoperation pro Takt durchführen kann, wurde für verschieden FPGAs und Prozessoren ermittelt, daß mit einem FPGA etwa zehnmal mehr Berechnungen pro Zeiteinheit auf derselben Chipfläche durchgeführt werden können als mit einem Prozessor [dHo00].

Dieser Packungsdichtevergleich von FPGAs gegenüber Prozessoren betrifft die Anzahl an Operationen pro Fläche und pro Zeit. Eine getrennte flächen- und zeitmäßige Betrachtung zeigt, daß dies durch eine effizientere Flächennutzung bei FPGAs zustande kommt, die durch eine niedrigere erreichbare Taktfrequenz teilweise kompensiert wird. Der Packungsdichtevorteil kann jedoch nur dann in einen Geschwindigkeitsvorteil umgesetzt werden, wenn hinreichend starke Parallelisierung möglich ist, wie z.B. bei wiederholter Anwendung einer Operation auf eine große Datenmenge.

3.2.3.3 Nutzungsmöglichkeiten

Da bei konfigurierbaren Architekturen Operationen durch räumliche Zusammensetzung von Grundelementen aufgebaut werden, existiert eine Obergrenze für die maximal implementierbare

Komplexität. Eine Reihe gut parallelisierbarer bzw. pipelinebarer Berechnungen lassen sich aufgrund der gestiegenen Kapazitäten inzwischen effizient mit FPGAs implementieren, während sequentielle Berechnungen bzw. solche, die viele unterschiedliche Verarbeitungsschritte umfassen, sich mit Prozessoren effizienter implementieren lassen. Bei Berechnungen, die zwischen den beiden Extremen liegen, sind in manchen Fällen beide Wege ineffizient, so daß ein hybrider Ansatz in Betracht kommt.

Wenn ein Algorithmus die Komplexitätsobergrenze überschreitet (bzw. wenn die Fläche für die Implementierung reduziert werden soll) und nicht alle Teile des Algorithmus zur gleichen Zeit benötigt werden, besteht die Möglichkeit zur Aufteilung des Algorithmus und des dynamischen Entfernens und Nachladens von Hardware. Wechselt häufig die Art der Verarbeitung und lassen sich die enthaltenen Teiloperationen durch eine einheitliche Pipeline implementieren, besteht die Möglichkeit, parallel zu den Daten die Konfigurationsinformation der einzelnen Stufen durch die Pipeline zu schleusen. Dies wird als Wormhole-Rekonfiguration bezeichnet.

Für Architekturen, die eine Rekonfiguration während des Betriebs erfordern, muß dynamisch rekonfigurierbare Logik (DRL) anstelle von statisch rekonfigurierbarer Logik (SRL) eingesetzt werden. Während bei SRL serielle Konfiguration ausreicht, ist für effiziente DRL wahlfreie Rekonfiguration, partielle Rekonfiguration und Multi-Kontext-Konfiguration gebräuchlich. Für eine effiziente dynamische Rekonfiguration fehlen derzeit noch Techniken zur schnellen Generierung von Konfigurationen, für deren schnelles Laden, sowie eine Standardplattform für die Konfiguration [AdRo99].

Algorithmen, die mit pseudo-statischen Daten arbeiten, bieten die Möglichkeit zur Einsparung redundanter Berechnungen durch Flexibilität. Bei FPGAs wird diese Technik als partielle Evaluierung bezeichnet, d.h. es werden Berechnungen, die lediglich von Konstanten abhängen vor der Erstellung der Konfiguration ausgewertet und entsprechend vereinfachte Konfiguration generiert. Bei Prozessoren entspricht dies den in der Compilertechnik eingesetzten Optimierungstechniken, die sich um so nutzbringender einsetzen lassen, je häufiger der optimierte Code benutzt wird. Die Zeit zum Erstellen und Laden der Konfiguration ist der limitierende Faktor für die Geschwindigkeit mit der eine Adaption an Parameter möglich ist.

Im Gegensatz zu Prozessoren erfolgt bei FPGAs die Steuerung auf Bitebene und nicht auf Wortebene. Dies kann bei feingranularen Problemen von Vorteil sein, ist jedoch die Granularität der Steuerung wesentlich gröber, dann ist diese Eigenschaft aufgrund des unnötig hohen Ressourcenverbrauches nachteilig. Um die zur Implementierung der Flexibilität erforderlichen Ressourcen effizient einzusetzen, muß die Granularität der Konfiguration der nötigen Flexibilität angepaßt werden, so daß ausreichend viel, jedoch keine unnötige Flexibilität zur Verfügung gestellt wird. Der Einsatz dedizierter Einheiten ist sinnvoll, wenn diese häufig genutzt werden, und wenn keine übermäßig generalisierten Einheiten verwendet werden, so daß sich durch deren Einsatz ein Pakkungsdichtevorteil ergibt.

3.3 Bekannte Architekturen

3.3.1 Klassifikation

In diesem Kapitel werden bekannte Architekturen vorgestellt, die sich zur effizienten Implementierung von Algorithmen der Videoobjekt-Segmentierung eignen. Dabei werden sowohl Architekturen betrachtet, die komplette Segmentierungsalgorithmen abdecken, als auch solche, die lediglich Teile der Algorithmen unterstützen.

Bei der Klassifikation der Architekturen wird, wie in Bild 3.5 dargestellt, nach der Größe des Einsatzbereiches bzw. dem Grad der Universalität differenziert. Die Klasse der universellen Architekturen umfaßt auf Videosignalverarbeitung ausgerichtete Prozessoren und universell kon-

figurierbare Systeme. Als zentrales Unterscheidungsmerkmal wird die Art der Flexibilität herangezogen. Problemangepaßte Architekturen sind solche Architekturen, die hinsichtlich der durchzuführenden Operationen in begrenztem Maße flexibel sind, die sich jedoch aufgrund ihres Aufbaus nur für einen bestimmten Anwendungsbereich einsetzen lassen. Als Unterscheidungsmerkmal bei diesen Architekturen wird die Art der Parallelität verwendet. Bei der Diskussion der Architekturen wird auf den unterstützten Anwendungsbereich bzw. die Klasse durchführbarer Operationen eingegangen. Die Klasse dedizierter Architekturen umfaßt schließlich solche Architekturen, die keine oder sehr geringe Flexibilität bieten. Im Folgenden werden bekannte Architekturen allgemein charakterisiert und mit Beispielen erläutert. Dabei wird auf die Eignung für die in Kapitel 3.1 genannten Anforderungen eingegangen.

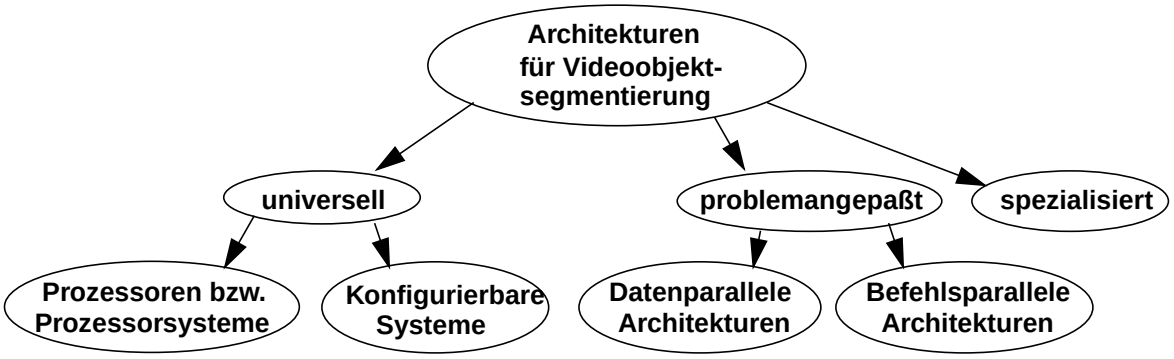


Bild 3.5: Klassifikation von Architekturen für die Videoobjekt-Segmentierung

3.3.2 Universelle Architekturen

3.3.2.1 Prozessoren und Prozessorsysteme

Prozessoren und Prozessorsysteme können grob nach ihrem Anwendungszweck klassifiziert werden. Neben der Klasse an Universalprozessoren, sind digitale Signalprozessoren (DSPs), Videosignalprozessoren und Multimediaprozessoren für Videoobjekt-Segmentierung von Interesse.

Universalprozessoren besitzen keine grundsätzliche Einschränkung hinsichtlich des Einsatzgebietes, sind jedoch aufgrund des breiten Anwendungsspektrums nicht spezifisch optimiert. Universalprozessoren, wie Celeron und Pentium, werden vielfach in Personalcomputern eingesetzt, erreichen hohe Rechenleistung und sind wegen ihrer starken Verbreitung sehr weit entwickelt. Aufgrund ihres breiten Einsatzgebietes erfordern Universalprozessoren hohen Flächenaufwand.

Auf Signalverarbeitung ausgerichtete Prozessoren erzielen für ihren Anwendungsbereich eine höhere Leistungsfähigkeit und eine bessere Nutzung ihrer Ressourcen, was zu größerer Flächeneffizienz und geringerer Verlustleistung führt. Digitale Signalprozessoren (DSPs) sind dadurch charakterisiert, daß sie auf Signalverarbeitung ausgerichtete architekturelle Merkmale besitzen, wie z.B. MAC-Einheiten (Multiplikation+Akkumulation), Einheiten zur Ausführung von Schleifen ohne Zeitaufwand für Sprungbefehle, strenge Harvard-Architektur und mehrere Speicherbusse. Um hohe Rechenleistung bereitzustellen finden in Signalprozessoren zunehmend Konzepte Einsatz, die sich in Universalprozessorarchitekturen bewährt haben, so daß sich der Unterschied zu Universalprozessoren zunehmend verwischt. [EyBi98].

Videosignalprozessoren (VSPs) bilden eine Unterklasse digitaler Signalprozessoren, die speziell auf die Verarbeitung zweidimensionaler Signale ausgerichtet ist. Beispielsweise besitzen VSPs zusätzliche architekturelle Merkmale für die Verwaltung zweidimensionaler Datenstrukturen. Seit kurzem finden Architekturen Verbreitung, die als Multimediaprozessoren bezeichnet werden. Dabei handelt es sich um Prozessoren, die auf die Verarbeitung sowohl ein- als auch zweidi-

mensionaler Signale zielen. Hauptanwendungsfeld von Multimediaprozessoren ist die Kodierung von Multimediadaten, wie sich an typischerweise integrierten Funktionseinheiten zeigt [Hut99].

3.3.2.1.1 Klassifikation von Prozessorarchitekturen

Prozessoren können im Hinblick auf zugrundeliegende Architekturkonzepte unterteilt und - wie in Tabelle 3.3 gezeigt - anhand der Art der genutzten Parallelität systematisiert werden. Instruction-Level-Parallelität (ILP) ist gegeben, wenn vorhandene Datenabhängigkeiten die Parallelisierung aufeinanderfolgender Befehle erlauben, Datenparallelität besteht, wenn mehrere bzw. viele Daten auf dieselbe Weise verarbeitet werden, und Task- bzw. Thread-Level-Parallelität (TLP) besteht, wenn ein Programm in mehrere parallel ausführbare Befehlsströme aufgeteilt werden kann. Die drei Parallelitätsarten lassen sich den in Kapitel 3.2.1.4 angegebenen Parallelitätsebenen nicht direkt zuordnen, jedoch korrespondieren die unteren Ebenen bis zur Komponentenoperation zu ILP und die mittleren Ebenen bis zur Bildoperation zu Datenparallelität, sofern es sich um reguläre Operationen handelt. In den höheren Ebenen tritt TLP und Datenparallelität gemischt auf. Im Folgenden wird näher auf die in Tabelle 3.3 genannten Konzepte eingegangen.

Art der Parallelität	Nutzung durch
Instruction-Level-Parallelität (ILP)	Superskalare Prozessoren
	VLIW-Prozessoren
Datenparallelität	SIMD-Prozessorsysteme
	Sub-Wort-Parallelität
	Pipeline-Vektorprozessoren
Task-Level-Parallelität (TLP)	MIMD-Prozessorsysteme
	Multithreading Prozessoren

Tabelle 3.3: Parallelitätsarten und Prozessorarchitekturen

Superskalare Prozessoren

Superskalare Architekturen haben sich aus RISC-Architekturen durch Einbringung von Superskalarität entwickelt, d.h. der Fähigkeit pro Taktzyklus mehrere Befehle zu starten. Sie sind durch einen Pipeline-Aufbau charakterisiert und besitzen mehrere parallel operierende Funktionseinheiten. Die Leistungsfähigkeit wird jedoch durch Ereignisse begrenzt, die den Pipeline-Ablauf stören, sogenannte Hazards. Man unterscheidet Struktur-, Daten- und Steuer-Hazards, die durch Ressourcenkonflikte, Datenabhängigkeiten und Programmverzweigungen entstehen. Um dem Auftreten bzw. der Auswirkung von Hazards entgegenzuwirken, wird typischerweise eine Mischung aufeinander abgestimmter Hardware-Einheiten eingesetzt, die durch Techniken wie dynamisches Scheduling, Bypassing, Registerumbenennung, Vorhersage von Verzweigungen und Spekulative Ausführung, eine hohe Auslastung der Prozessor-Pipeline sicherstellen.

Durch diese Hardware-Einheiten können superskalare Prozessoren mit dem Befehlssatz skalarer Prozessoren umgehen, was einen wesentlichen Beitrag zu deren Erfolg leistete. Da jedoch bei superskalaren Prozessoren zur Befehlsdekodierung und Steuerung ein sehr hoher Aufwand erforderlich ist, ist deren Flächeneffizienz prinzipbedingt gering. Superskalare Architekturen werden jedoch in Verbindung mit Multimediaerweiterungen für Videosignalverarbeitung eingesetzt (siehe Sub-Wort-Parallelität).

VLIW-Prozessoren

VLIW-Prozessoren liegt die Überlegung zugrunde, daß es nicht notwendig ist Hardware-Einheiten zur Auflösung von Pipeline-Hazards einzusetzen, da dies bereits bei der Programmerstellung durch einen optimierenden Compiler geschehen kann. Neben dem Verzicht auf dynamisches Scheduling wird bei VLIW-Prozessoren auch auf die Vorhersage von Verzweigungen und auf spekulative Ausführung verzichtet, stattdessen spiegelt sich der Pipelineaufbau direkt im Programmiermodell wieder, z.B. in Form sogenannter Delay Slots (d.h. Verzweigungen werden entsprechend der Anzahl an Pipelinestufen verzögert ausgeführt). Durch diesen Verzicht werden Hardware-Einheiten und der dafür nötige Entwurfs- und Flächenaufwand eingespart, so daß VLIW-Architekturen im Vergleich zu superskalaren Architekturen ein günstigeres Verhältnis von Flächenaufwand zu Verarbeitungsleistung erreichen.

Da das Scheduling vom Compiler übernommen wird, und die Befehlsausführung direkt durch den Befehlsstrom gesteuert wird, muß auf Befehlssatzkompatibilität zu skalaren Architekturen verzichtet werden. Wie superskalare Prozessoren enthalten auch VLIW-Architekturen mehrere parallel operierende Funktionseinheiten, um vorhandene ILP zu nutzen. Da viele Funktionseinheiten durch den Befehlsstrom gesteuert werden, ergibt sich ein sehr langes Befehlswort, ein Very Long Instruction Word (VLIW). Die hohe Befehlsbandbreite, wird typischerweise durch Begrenzung der Anzahl parallel ausführbarer Befehle und durch Befehlskompression abgemildert.

Während sich Grundzüge des VLIW-Konzeptes bereits seit langem in DSPs finden, wurde der Durchbruch von VLIW-Architekturen vor allem durch Fortschritte in der Compilertechnologie ermöglicht, die eine relative effiziente Programmierung in Hochsprache ermöglichen. Moderne VLIW-Compiler benutzen Methoden zur Code-Reorganisation, so daß ILP besser genutzt werden kann.

VLIW-Prozessoren eignen sich für Signalverarbeitungsaufgaben, da sie feingranulares Parallelisierungspotential nutzen, ohne hohen Steuerungsaufwand zu erfordern, wie superskalare Prozessoren bzw. MIMD-Prozessorsysteme [MeTa97]. Zur Kategorie auf Videosignalverarbeitung ausgelegter VLIW-Architekturen gehören der Philips Trimedia [TM1300] und Prozessoren der Familie TMS320C6xx [C60] auf die in Kapitel 3.3.2.1.2 näher eingegangen wird.

SIMD-Prozessorsysteme

SIMD-Prozessorsysteme bestehen aus mehreren gleichen Datenpfaden und einem zentralen Steuerwerk. Die Datenpfade führen gleichzeitig dieselbe Operation für verschiedene Daten durch, so daß sich SIMD-Architekturen gut zur Nutzung von Datenparallelität eignen.

Eine Variante von SIMD-Architekturen, sogenannte autonome SIMD-Architekturen [KnBe97], [RöKn96], unterstützt bedingte Ausführung, wobei die Bedingung von jedem Datenpfad getrennt berechnet wird. Bei diesen Architekturen lassen sich Verzweigungen implementieren, indem alle Zweige hintereinander durchlaufen und die Datenpfade entsprechend dem Ergebnis der Bedingung maskiert werden.

Vorteilhaft ist der niedrige Steuerungsaufwand, da Befehls-Cache, Befehlsdekoder und Steuereinheit nur einmal vorhanden sind. Die Programmierung eines SIMD-Prozessorsystems ist einfacher als die eines MIMD-Prozessorsystems, da nur ein Befehlsstrom verarbeitet wird, und die Synchronisation implizit gegeben ist. Die für Videosignalverarbeitung ausgelegte HiPar-Architektur, die an der Universität Hannover entworfen wurde, wird in Kapitel 3.3.2.1.2 beschrieben.

Pipeline-Vektorprozessoren

Prozessoren, die über spezielle Befehle zur Verarbeitung von Vektoren verfügen, werden als Vektorprozessoren bezeichnet. Während SIMD-Prozessorsysteme und Multimediaerweiterungen auf dem Prinzip der Parallelisierung beruhen, werden in diesem Kapitel Vektorprozessoren betrach-

tet, denen das Pipeline-Prinzip zugrunde liegt.

Pipeline-Vektorprozessoren erreichen ihre Geschwindigkeit durch schnelle Logikschaltungen und durch die Unterteilung von Operationen in Teiloperationen, die den Stufen einer gepipelinten Funktionseinheit zugeordnet werden. Bei vielen aufeinanderfolgenden identischen Operationen wird eine hohe Beschleunigung erzielt, die maximal gleich der Anzahl an Pipelinestufen ist [Hos88].

Pipeline-Vektorprozessoren erfordern spezielle Assemblerbefehle zur Programmierung, so daß keine Binärkompatibilität zu skalaren Prozessoren gegeben ist. Auf Hochsprachenebene ist die Strukturierung des Programmes entscheidend für die erzielbare Beschleunigung. Vektorisierende Compiler sind für einfache Konstrukte einsetzbar, komplizierte Konstrukte müssen jedoch bei der Programmerstellung entsprechend strukturiert werden, z.B. durch Vertauschung geschachtelter Schleifen bzw. durch Abrollen von Schleifen. Die durch Vektorisierung maximal erreichbare Beschleunigung ist durch den Anteil an nicht vektorisierbarem Code entsprechend dem Ahmdahlschen Gesetz begrenzt (siehe Kapitel 3.2.1.3).

Pipeline-Vektorprozessoren erzielen bei Algorithmen, die viele Gleitkomma-Berechnungen enthalten, hohe Beschleunigung, da diese komplexer als Festkomma-Berechnungen sind, und in viele Pipelinestufen zerlegt werden können. Sie werden typischerweise in Supercomputern für numerische Berechnungen eingesetzt, die viele Vektoroperationen enthalten, nicht jedoch zur Signalverarbeitung mit Festkomma-Berechnungen.

MIMD-Prozessorsysteme

MIMD-Prozessorsysteme bestehen aus mehreren unabhängigen Prozessoren mit getrennten Datenpfaden und Steuerwerken. Neben reinen MIMD-Systemen gibt es Mischformen von SIMD und MIMD, die auch als erweiterte SIMD-Architekturen bezeichnet werden. Diese bestehen aus mehreren Steuereinheiten und Datenpfaden, die einander statisch oder dynamisch zugeordnet werden können, so daß mehrere Datenpfade von einem Controller gesteuert werden [GeGa95]. Bei statischer Zuordnung spricht man von Clustered SIMD-Architekturen, bei dynamischer Zuordnung von Assoziativer Steuerung [PeGe94].

Die Prozessoren in einem MIMD-System arbeiten unabhängig und in der Regel nicht synchronisiert, so daß bei Zugriffs- oder Ressourcenkonflikten ein Prozessor bevorzugt werden muß, und somit selbst im Fall gleicher Operationen die Synchronisation verloren geht. Aus diesem Grunde eignen sich MIMD-Prozessorsysteme nicht zur Nutzung von ILP, bei der eine enge Synchronisation erforderlich ist. MIMD-Systeme werden vielmehr bei Anwendungen eingesetzt, bei denen Blöcke grober Granularität so auf Prozessoren abgebildet werden können, daß nur selten Kommunikation zwischen den Prozessoren erforderlich ist, und der Synchronisationsaufwand gering bleibt. Die Nutzung von TLP und Datenparallelität ist möglich, wobei im zweiten Fall der Einsatz von SIMD-Architekturen aufgrund des wesentlich niedrigeren Steuerungsaufwandes effizienter ist. Ein Beispiel für ein auf Videosignalverarbeitung ausgelegtes Single-Chip MIMD-System ist der TMS320C8x der Firma Texas Instruments [C80], der sich jedoch aufgrund seiner komplexen Programmierung nicht durchsetzen konnte.

Multithreading Prozessoren

Neben der hohen Latenz von Befehls-Cache-Misses bedingt eingeschränkte ILP und Datenabhängigkeiten zu Operationen mit hoher Latenz bei den zuvor beschriebenen Konzepten das Auftreten vieler Leerlaufzyklen. Multithreading zielt daher darauf, solche Latenzzeiten durch Umschalten zu einem rechenbereiten Thread zu verbergen, wobei durch Nutzung mehrerer Registerblöcke schnelles Umschalten von Threads ermöglicht wird. Architekturen, die Multithreading nutzen, lassen sich in die Kategorien nicht-gleichzeitiges Multithreading, d.h. nur ein Thread kann gleichzeitig die Befehlspipeline nutzen, und simultanes Multithreading (SMT) unterteilen,

d.h. mehrere Threads können gleichzeitig die Ausführung von Befehlen starten.

SMT ermöglicht dynamische Allokation von Verarbeitungsressourcen zu Threads und steigert die Auslastung von Funktionseinheiten und Speicher. SMT-Architekturen nutzen ILP und TLP. Durch Hinzufügen von SIMD-artigen Funktionseinheiten ist auch die Nutzung von Datenparallelität möglich.

Trotzdem wurden bisher - mit einer Ausnahme - keine SMT-Architekturen für Videosignalverarbeitung vorgeschlagen. Ein Grund dafür könnte darin liegen, daß derartige Algorithmen eine hohe Regularität aufweisen, und es durch Pipelining über Schleifengrenzen hinweg möglich ist, Latenzzeiten durch überlappende Operationsausführung zu verbergen, ohne Aufwand für mehrfache Registerblöcke und die Steuerung der Thread-Umschaltung zu investieren. Lediglich in [BePi97] wurde der Einsatz von SMT-Architekturen für MPEG-4 Kodierung und Dekodierung vorgeschlagen, die bei steigender Komplexität weniger Regularität als vorherige Kodierverfahren aufweisen.

Sub-Wort-Parallelität

Sub-Wort-Parallelität ist ein Konzept, um Datenparallelität in Prozessoren stärker zu nutzen. Ansatzpunkt ist dabei, daß viel Algorithmen nur einen Teil der verfügbaren Wortbreite von Funktionseinheiten tatsächlich nutzen. Bei Algorithmen mit Potential an Datenparallelität ist es zweckmäßig, Funktionseinheiten mit großer Wortbreite in mehrere Teileinheiten aufzuspalten, so daß diese unabhängig operieren können. Auf diese Weise können alle Teile einer Funktionseinheit gleichzeitig zur Verarbeitung mehrerer Daten eingesetzt werden. Bei Sub-Wort-Parallelität handelt es sich somit nicht um eine Grundarchitektur, sondern um ein Konzept, das sich mit den zuvor beschriebenen Grundarchitekturen kombinieren läßt. Sub-Wort-Parallelität ist bei Algorithmen der Videosignalverarbeitung oftmals vorteilhaft, da diese ein hohes Potential an Datenparallelität bei Operationen mit geringer Wortbreite aufweisen. Sub-Wort-Parallelität wird insbesondere bei Universalprozessoren in Form sogenannter Multimediaerweiterungen eingesetzt.

3.3.2.1.2 Prozessoren für Videosignalverarbeitung

Universalprozessoren mit Multimediaerweiterung

Bei Multimediaerweiterungen für Universalprozessoren handelt es sich um Einheiten, die eine SIMD-artige Parallelverarbeitung von typischerweise 2, 4 oder 8 Datenworten mit je 8 bit oder 16 bit gestatten. Der Befehlssatz umfaßt dabei typischerweise Additionen, Subtraktionen, Multiplikationen, MAC, Schiebeoperationen sowie Sortierfunktionen [LeSm96], [Kne98]. Da aufgrund der reduzierten Wortbreite und der parallelen Verarbeitung mehrere Wertebereichsüberschreitungen auftreten können, wird im Gegensatz zu konventionellen Befehlen, bei denen dies durch ein Status-Bit signalisiert wird, oftmals mit Sättigung gearbeitet.

Beispiele für Multimediaerweiterungen sind MMX für x86 Prozessoren (Firma Intel [ChMi97] bzw. AMD [Dra97]) und VIS (Visual Instruction Set) für UltraSparc Prozessoren (Firma Sun [Nor98]). Weitere Beispiele für Architekturen mit Multimediaerweiterungen sind die PaRISC Architektur der Firma Hewlett Packard [Kum97] und der V830R/AV Prozessor der Firma NEC [SuAr98].

Während Multimediaerweiterungen die Nutzung von Datenparallelität in einem begrenzten Maß gestatten, können allerdings nur reguläre Operationen beschleunigt werden, nicht jedoch irregulären Operationen, wie z.B. solche, die auf Segment-Adressierung beruhen.

Auch das Problem der Zufuhr großer Datenmengen, wie es bei Videosegmentierung auftritt, wird durch Multimediaerweiterungen in Prozessoren mit unveränderter Speicherschnittstelle nicht gelöst. Caches zeigen beispielsweise bei Zugriffen keine Wirkung, bei denen fortlaufende Daten

jeweils nur einmal verwendet werden. Außerdem sind Caches in Universalprozessoren nicht auf zweidimensionaler Datenstrukturen ausgerichtet. Bei Zugriffen auf Bilddaten tritt auch häufig das Problem der Adressierung nicht auf Wortgrenzen ausgerichteter Daten auf, da typischerweise nur für jeden vierten bzw. achten Bildpunkt die Ausrichtung an einer Wortgrenze gegeben ist.

Trotz hoher Regularität bei Videooperationen ist zusätzliche Speicherbandbreite für die Befehlszufuhr und Flächenaufwand für Befehls-Cache und Befehlsdekodierung erforderlich. Befehlssatz und Funktionseinheiten sind, abgesehen von den Befehlen der Multimediaerweiterung, nicht auf typische Operationen der Videosignalverarbeitung ausgerichtet.

Universalprozessoren enthalten auch viele Blöcke, die nicht für Videosignalverarbeitung genutzt werden, wie z.B. Gleitkomma-Einheiten, und sind daher weniger flächeneffizient. Da bereits Universalprozessoren ohne Erweiterung eine große Chipfläche aufweisen, gilt dies um so mehr für Universalprozessoren mit Multimediaerweiterungen.

Ein weiterer Nachteil von Universalprozessoren ist ihr hoher Leistungsverbrauch. In letzter Zeit wurde erkannt, daß niedriger Leistungsverbrauch für den mobilen Einsatz wichtig ist, und es wurden verstärkt Maßnahmen zur Leistungseinsparung eingesetzt, wie z.B. Speedstep bei Prozessoren der Firma Intel, und LongRun beim Crusoe Prozessor der Firma Transmeta. Eine Energieeinsparung durch Powermanagement ist jedoch nur möglich, wenn der Prozessor nicht vollständig durch Berechnungen ausgelastet ist.

TRIO-Prozessor

Eine im Vergleich zu Multimediaerweiterungen stärker auf Bildverarbeitung ausgerichtete Erweiterung wird beim TRIO-Prozessor [SiFa96] eingesetzt. Dieser enthält, neben einer Einheit für konventionelle Befehle (General Instruction Unit, GPU), eine spezialisierte Funktionseinheit (Image Processing Unit, IPU), die Bildverarbeitungsoperationen unterstützt. Der Befehlsumfang umfaßt 16 Bildverarbeitungsbefehle, zu denen Spezialbefehle für Faltung, Dilatation und Erosion zählen. Die Bildverarbeitungsbefehle sind auf eine Nachbarschaft von 3x3 Bildpunkten beschränkt. Im Gegensatz zu Prozessoren mit Multimediaerweiterungen, ist auch das Speicherinterface erweitert, die Datenzufuhr für konventionelle Befehle erfolgt über einen 32 bit breiten Bus und für Bildverarbeitungsbefehle über drei 8 bit breite Busse. In [SiFa96] ist eine Ausführungsgeschwindigkeit von 14 Bilder pro Sekunde für eine Bildgröße von 512x512 angegeben, dies entspricht 13,6 Takten pro Operation bei der angegebenen Taktfrequenz von 50 MHz.

Die Architektur bietet im Unterschied zu Universalprozessoren mit Multimediaerweiterungen eine andere Funktionalität und hat eine erweiterte Speicherschnittstelle, die parallele Zugriffe auf Bilddaten unterschiedlicher Quellen gestattet. Aufgrund der geringen Datenwortbreiten wird jedoch keine hohe Bandbreite erreicht, außerdem wird die Datenorganisation durch die Aufteilung auf mehrere Speicher komplizierter. Die Architektur besitzt keinen Pufferspeicher, um bei zeilen- bzw. spaltenweiser Verarbeitung die zu transferierende Datenmenge zu reduzieren.

TMS 320C6xx

Die Prozessoren TMS320C62x, TMS320C64x und TMS320C67x basieren auf derselben, in Bild 3.6 gezeigten, VLIW-Architektur [C60]. Während der C62x und der C64x acht parallele Funktionseinheiten für Festkomma-Operationen besitzt, umfaßt der C67x zusätzlich 6 Einheiten für Gleitkomma-Operationen. Der C64x ist mit dem C62x vergleichbar, besitzt jedoch die doppelte Anzahl an Registern, erweitere Sub-Wort-Parallelität, weitere Unterstützung von Bit-Level-Algorithmen und Spezialbefehle für fehlerkorrigierende Codes.

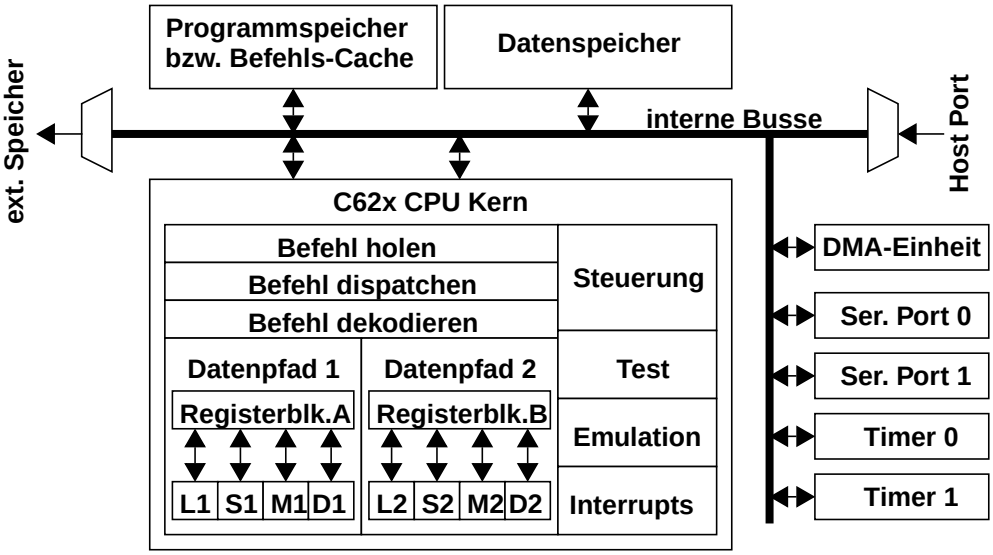


Bild 3.6: Blockdiagramm des TMS320C62x

Die Funktionseinheiten sind in zwei Blöcke je 4 bzw. 7 Funktionseinheiten aufgeteilt, die über 10 Lese- und 6 Schreibports auf den ihnen zugeordnete Registerblock bzw. über einen Leseport auf den anderen Registerblock zugreifen können. Die Größe der Registerblöcke beträgt jeweils 16x32 bit. Die Funktionseinheiten unterstützen Operationen mit 8, 16 bzw. 32 bit Wortbreite, Sub-Wort-Parallelität mit 2x16 bit, und einige Operationen mit 40 bit Wortbreite. Die L1/2- und S1/2-Funktionseinheiten unterstützen arithmetische und logische Operationen, die M1/2-Einheiten Multiplikationen, und die D1/2-Einheiten Additionen, Subtraktionen und Adreßrechnung für Speicherzugriffe. Weitere Merkmale sind Sättigung, Normalisierung und Extraktion/Setzen/Löschen/Zählen von Bits. Alle Operationen können bedingt ausgeführt werden, um Verzweigungen im Interesse einer hohen ILP zu vermeiden. Die Befehlsworte werden komprimiert im Speicher abgelegt. Je 8 Befehle werden zu einem Fetch-Packet zusammengefaßt, diese müssen jedoch nicht unbedingt im selben Takt ausgeführt werden. Das dekomprimierte Befehlswort hat eine Breite von 256 bit.

Der C6201 bzw. C6701 besitzt einen internen Datenspeicher (64kByte) auf den zwei Zugriffe parallel erfolgen können und einen internen Programmspeicher (64kByte), der entweder als Programm-Cache oder als lokaler Programmspeicher benutzt werden kann. Die Speicheranbindung erfolgt über einen 32 bit breiten Bus und unterstützt SDRAM und SRAM. Der Chip wurde in 0,25 µm / 2,5 V Technologie gefertigt, und kann mit maximal 250 MHz (C62x) bzw. 167 MHz (C67x) betrieben werden. Der C6414 besitzt hingegen eine zweistufige Speicherarchitektur. Die erste Stufe besteht aus einem Befehls-Cache (16kByte), einem Daten-Cache (16kByte), die zweite Stufe besteht einem gemeinsamen Cache (1MByte). Die Speicheranbindung erfolgt über einen 64 bit breiten Bus und unterstützt SDRAM und SRAM. Der C6414 ist mit 0,12 µm / 1,4V Technologie gefertigt, und kann mit maximal 600 MHz betrieben werden.

HiPAR

Der an der Universität Hannover entwickelte HiPAR-Videosignalprozessor [RöKn96], wurde aufgrund einer Analyse der Parallelität, Datenzugriffe und Steuerungsanforderungen typischer Bildverarbeitungsalgorithmen konzeptioniert. Es handelt sich um ein autonomes SIMD Prozessorsystem, das durch einen VLIW-Befehlsstrom gesteuert wird.

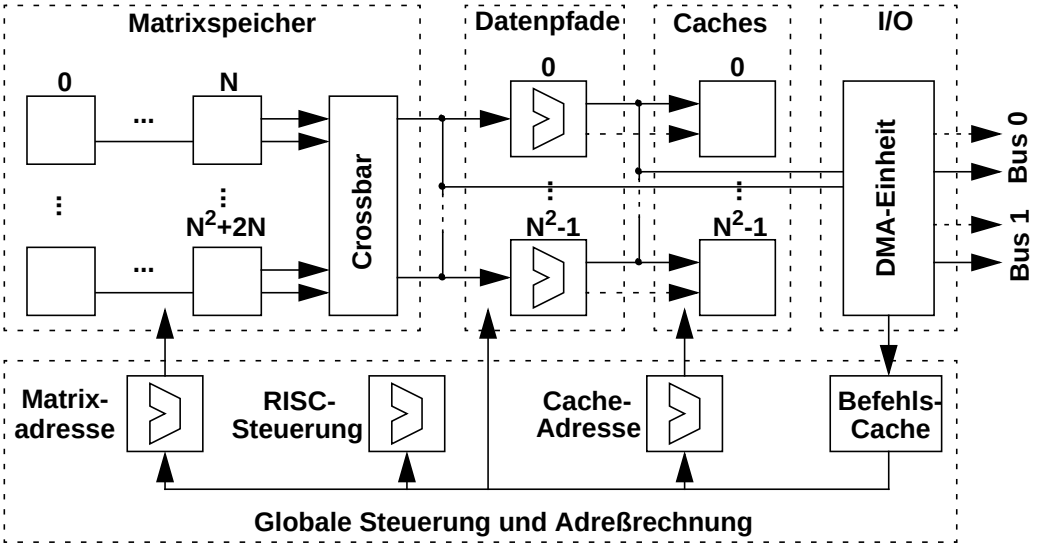


Bild 3.7: Blockdiagramm des HiPAR

Die HiPAR-Architektur besteht aus 4 (HiPAR-4) bzw. 16 (HiPAR-16) parallelen Datenpfaden und einer zentralen RISC-Steuereinheit. Die 4 bzw. 16 identischen Datenpfade bestehen jeweils aus einem lokalen Registerblock mit 16 Registern zu je 16 bit und drei parallel operierenden arithmetischen Einheiten für Daten- und Adreßrechnung. Die ALU unterstützt logische Operationen, Addition und Subtraktion für 16 bit oder 32 bit Daten bzw. für 2x16 bit Daten (Sub-Wort-Parallelität). Die MAC-Einheit übernimmt 16 bit x 16 bit Multiplikationen und anschließende Akkumulation. Zusätzlich ist eine Einheit für Schiebeoperationen und Rundung enthalten. Quelle und Ziel aller Operationen und Speicherzugriffe ist der Registerblock im Datenpfad.

Jeder Datenpfad hat über einen privaten Cache Zugriff auf den externen Speicher, und über eine Crossbar auf den gemeinsamen Matrixspeicher. Datentransfers vom und zum externen Speicher werden von einer autonomen DMA-Einheit durchgeführt. Gemeinsame Daten müssen im Matrixspeicher abgelegt und regulär zugegriffen werden. Irregulär zugegriffene Daten, deren Adresse lokal in einem Datenpfad berechnet wird, müssen im externen Speicher abgelegt werden und über den privaten Cache zugegriffen werden. Dabei gibt es keinen Mechanismus, der Cache-Kohärenz sicherstellt, so daß die Adreßräume der Datenpfade getrennt sein müssen. Auf den Matrixspeicher wird mit virtuellen 2D-Adressen zugegriffen, die aus der oberen linken Position und dem vertikalen und horizontalen Abstand zwischen den Matrixelementen bestehen. Überlappende und nicht-überlappende Matrixzugriffe, Vektor- und Skalarzugriffe sind möglich.

Der Matrixspeicher besteht aus 9 bzw. 25 Blöcken (HiPAR-4 bzw. HiPAR-16) zu 0,5 bzw. 1 kByte. Die Größe der privaten Caches beträgt 0,5 bzw. 2 kByte pro Datenpfad. Die Synthese des HiPAR-4 erfolgte mit einer 0,6 µm CMOS Technologiebibliothek mit 2 Metallebenen. Es ergab sich ein Flächenbedarf von 250 mm^2 für $1{,}2 \cdot 10^6$ Transistoren. Der Leistungsverbrauch bei 80 MHz Taktfrequenz liegt bei 5 W. Für den HiPAR-16 wurde für eine 0,35 µm CMOS-Technologie mit 3 Metallebenen eine Fläche von 200 mm^2, eine Transistorzahl von $4{,}3 \cdot 10^6$, eine erreichbare Taktfrequenz von 100 MHz und ein Leistungsverbrauch von 8 W angegeben.

3.3.2.2 Konfigurierbare Systeme

Wie in Kapitel 3.2.3.1 erläutert, ist Konfigurierbarkeit ein weiteres Konzept, um Flexibilität bereitzustellen. Während manche Systeme in nahezu beliebiger Weise konfiguriert werden können, betrifft die Flexibilität bei anderen lediglich die Wahl von Betriebsmodus und Parametern, wobei bereits eine Grundarchitektur vorgegeben ist. Im ersten Fall erfolgt die Erstellung der Kon-

figuration durch Schritte, die weitgehend einem konventionellen Hardware-Entwurfsablauf entsprechen. Typischerweise wird zuerst eine Architektur entworfen, diese anschließend mit einer Hardware-Beschreibungssprache modelliert, und die Konfigurationsinformation mittels automatischer Werkzeuge generiert. Im zweiten Fall ist die Grundarchitektur bereits vorgegeben, so daß für die Erstellung der Konfiguration lediglich die Funktion festgelegt werden muß. In manchen Fällen gibt es automatische Werkzeuge zur Umsetzung einer Funktionsbeschreibung in eine Konfiguration, in anderen Fällen erfolgt deren Erstellung manuell.

Im ersten Fall handelt es sich um konfigurierbare Systeme. Konfigurierbare Systeme sind meist aus FPGAs aufgebaut. Obwohl diese strenggenommen keine Architekturen sondern Plattformen zur Implementierung von Architekturen darstellen, werden konfigurierbare Systeme aufgrund ihrer Bedeutung in der Videosignalverarbeitung in diesem Kapitel diskutiert. Im zweiten Fall handelt es sich um konfigurierbare Architekturen, mit - zugunsten einer Spezialisierung auf bestimmte Aufgaben - eingeschränkter Flexibilität. Konfigurierbare Architekturen, die sich für Videosegmentierung einsetzen lassen, werden in Kapitel 3.3.3.2 behandelt.

Splash-1 und Splash-2

Splash-1 und Splash-2 sind aus statisch rekonfigurierbaren FPGAs aufgebaute Architekturen, die am Supercomputing Research Center entwickelt wurden [RaJa97]. Die Splash-2 Architektur besteht aus einem Array von Xilinx 4010 FPGAs, gegenüber der mit Xilinx 3090 FPGAs aufgebauten Splash-1 Architektur. Bild 3.8 zeigt die Zusammenschaltung aus Host Computer, Interface-Board und mehreren Splash-Boards mit je 16 PEs (X1 bis X16), die aus einem FPGA und 512 kByte RAM bestehen. Jedes PE besitzt eine 36 bit Anbindung an eine Crossbar die von einem weiteren FPGA (X0) gesteuert wird, außerdem läuft durch die PEs ein 36 bit breiter Datenpfad. Der Speicher besitzt zwei Ports und kann vom FPGA und vom Host aus zugegriffen werden. Für die Filterung eines 512x512 Bildes mit dem Sobel-Operator ist in [RaJa97] ein Zeitbedarf von 13,89 ms angegeben.

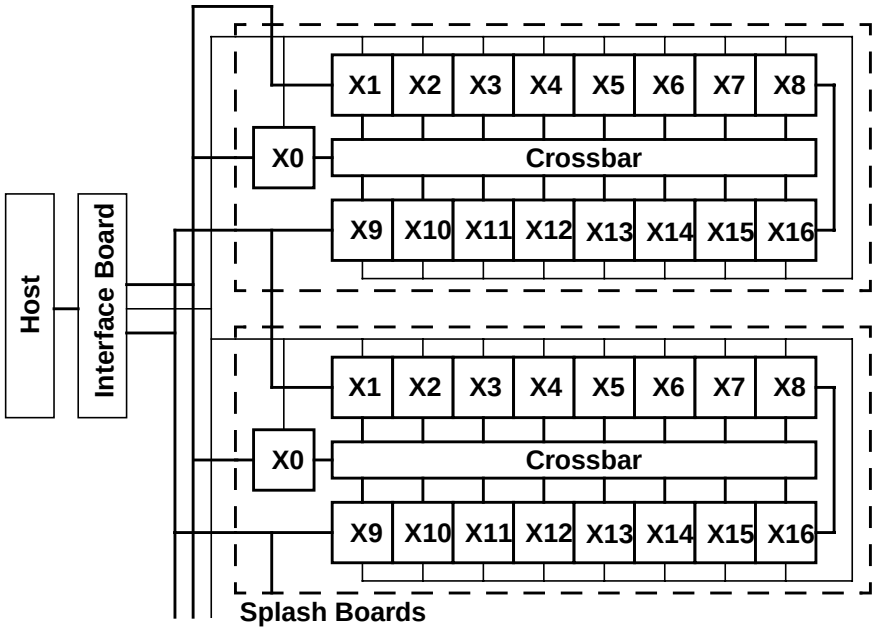


Bild 3.8: Blockdiagramm des Splash-2 Systems

Wildstar, Firebird und Wildforce

Die Wildstar- und Firebird-Karten der Firma Annapolis Micro Systems [Wild] enthalten dyna-

misch rekonfigurierbare Virtex FPGAs und werden oftmals für Video- und Bildverarbeitungsaufgaben eingesetzt. Die Wildforce PC-Einsteckkarte enthält statisch rekonfigurierbare Xilinx 4036XL FPGAs und hat einen Splash-2 ähnlichen Aufbau, jedoch sind nur 4 PEs enthalten.

Sonic-1

Eine weiteres FPGA basiertes System ist die Sonic Architektur [HaSt00], die Anwendungen im Bereich Echtzeit-Videosignalverarbeitung unterstützt. Die in Bild 3.9 gezeigte Architektur ist aus Plug In Processing Elements (PIPEs) aufgebaut, die über ein Bussystem verbunden sind. Der PIPE-Bus (32 bit) wird global von allen PIPEs gemeinsam genutzt, das Laden/Speichern von Bilddaten vom und zum Host, die Konfiguration und die Steuerung der PIPEs erfolgen über diesen Bus, dessen Bandbreite der des Host-Busses entspricht. Die PIPEflow Busse (16 bit plus 3 bit für Steuerungszwecke), die Verbindungen von Element zu Element, den A-Bus und B-Bus umfassen, stellen flexible Verbindungen für eine Datenpipeline bereit. Ihre Bandbreite entspricht der externen Video-Bandbreite.

Die PIPE-Elemente bestehen aus PIPE-Engine, PIPE-Memory und PIPE-Router. Beim PIPE-Memory handelt es sich um ein statisches RAM der Größe 1Mx32 bit, das die lokale Speicherung der Bilddaten übernimmt. Die PIPE-Engine übernimmt die Verarbeitung. Der Datenaustausch erfolgt über den PIPE-Router oder mittels direktem Zugriff auf das PIPE-Memory. Die PIPE-Router sind zuständig für die Konversion des Datenformates, das Routing der Daten über die Busse und für Zugriffe auf den Speicher mit horizontaler oder vertikaler Verarbeitungsabfolge oder streifenweisem Zugriff.

Die Sonic Architektur wurde als PC-Einsteckkarte implementiert, die als Sonic-1 bezeichnet wird und bis zu 8 PIPE-Elemente umfaßt. PIPE-Router und PIPE-Engine werden mit zwei separaten FPGAs des Typs FLEX10K50 bzw. FLEX10K100 implementiert. Die Taktfrequenz von Sonic-1 beträgt 33 MHz. Die Verarbeitungsgeschwindigkeit für ein 3x3 FIR-Filter angewandt auf ein 512x512 Punkte großes Bild beträgt 52,6ms bei Einsatz eines PIPE-Elements bzw. 30,0 ms beim Einsatz von zwei Elementen. Die Dauer der Rekonfiguration eines PIPE-Elementes beträgt 150 ms. Begrenzend für die Systemgeschwindigkeit und Skalierbarkeit wirkt sich die Bandbreite des Host Busses und des PIPE-Busses. Die Bildgröße ist durch die Größe der eingesetzten Speicher auf das CCIR601 Bildformat begrenzt.

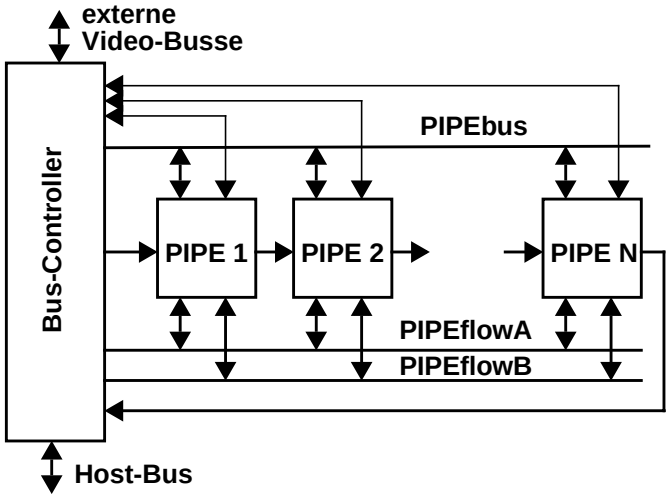


Bild 3.9: Blockdiagramm des Sonic-1 Systems

Prozessoren mit konfigurierbarer Logik

Neben rein konfigurierbaren Systemen gibt es auch solche, die Prozessoren mit konfigurierbarer Logik auf einem Chip integrieren. Beispiele hierfür sind die E5-Familie [TriE5] und die A7-Familie [TriA7] von Triscend. Chips der E5-Familie bestehen aus dem 8 bit Mikrokontroller 8032, 8 kByte bis 64 kByte RAM und 256 bis 3200 konfigurierbare Logikzellen und erreichen maximal 40 MHz Taktfrequenz. Chips der A7-Familie enthalten den 32 bit ARM7TDMI Prozessor, 8 kByte Cache, 16 kByte RAM und 512 bis 3200 konfigurierbare Logikzellen. Ein anderes Beispiel ist die Garp Architektur [CaHa00], die einen MIPS-II Prozessor zusammen mit rekonfigurierbarer Logik umfaßt.

Einsatzmöglichkeiten

Konfigurierbare Systeme sind grundsätzlich sehr flexibel einsetzbar, jedoch haben damit implementierte Architekturen einen um bis zu Faktor 10-20 größeren Flächenverbrauch und erreichen nur 1/3 bis 1/5 der Taktfrequenz dedizierter Architekturen [BrFr92]. Trotzdem bieten konfigurierbare Systeme einen Vorteil, wenn sie genutzt werden um Algorithmen mit hoher Parallelität in kurzer Zeit zu implementieren oder wenn die Möglichkeit zur Konfiguration während des Betriebes genutzt werden kann.

Letzteres erfordert schnelle Umkonfiguration. Bei Videosegmentierung ist eine Umkonfiguration im Fall regulärer Adressierung nach der Verarbeitung eines Bildes nötig, im Fall irregulärer Adressierung hingegen kann sie bereits nach der Verarbeitung einiger Bildpunkte erforderlich sein. Während die Umkonfiguration im ersten Fall nicht wesentlich mehr als 100 µs dauern darf, um nicht ins Gewicht zu fallen, ist im zweiten Fall eine sofortige Umkonfiguration nötig. In beiden Fällen ist die Konfigurationszeit der beschriebenen Systeme zu groß, abgesehen von Wildstar- bzw. Firebird-Karten, die mit dynamisch rekonfigurierbaren FPGAs arbeiten.

Bei Architekturen, die Umkonfiguration während des Betriebs nicht gestatten, ergibt sich ein Platzproblem bei Implementierung komplexer Verfahren, die aus vielen unterschiedlichen Operationen bestehen, da die Implementierung durch räumliche Zusammensetzung von Operationen erfolgen muß. Die räumliche Zusammensetzung wird jedoch durch die Chipgrößen der eingesetzten FPGAs, durch die Struktur der konfigurierbaren Logikblöcke und durch die Bandbreite der Verbindungsbusse limitiert. Außerdem ist bei der Implementierung eines Algorithmus eine gleichmäßige Lastaufteilung durch Zuordnung von Operationen zu verschiedenen FPGAs sicherzustellen. Bei Segmentierungsverfahren ergibt sich jedoch das Problem, daß die Verarbeitung von Segmenten nicht effizient aufgeteilt werden kann. Auch die Datenorganisation ist ein wichtiger Gesichtspunkt: Für die Speicherung von Daten ist externer Speicher wesentlich effizienter als Speicher innerhalb von FPGAs. Einfache aber datenintensive Operationen nutzen die Ressourcen des FPGAs nicht aus, wenn die Datenzufuhr den Engpaß darstellt.

Der Entwurfsaufwand ist für konfigurierbare Systeme im Vergleich zur Programmierung prozessorbasierter Systeme höher. Aus diesen Gründen werden konfigurierbare Systeme typischerweise für experimentelle Systeme eingesetzt, bei denen der hohe Hardware-Aufwand einhergehend mit hohen Kosten, großem Platzbedarf und hoher Verlustleistung toleriert werden kann, wenn dadurch Videosignalverarbeitungsverfahren mit hoher Geschwindigkeit simuliert werden können.

3.3.3 Problemangepaßte Architekturen

3.3.3.1 Datenparallele problemangepaßte Architekturen

Während bei Nutzung der Befehlsparallelität in der Regel lediglich ein einstelliger Parallelisierungsgrad erreicht wird, liegt dieser bei datenparallelen Architekturen deutlich höher, im Extremfall ist er gleich der Anzahl der Bildpunkte. In diesem Fall spricht man von massiv parallelen

Architekturen.

Massiv parallele Architekturen für Videosignalverarbeitung sind entsprechend der zweidimensionalen Struktur der Bilddaten als zweidimensionale Arrays aus verteilten Prozessorelementen (PEs) mit Verarbeitungseinheiten und Speicherzellen organisiert. In diesem Kapitel werden Architekturen mit programmierbaren bzw. konfigurierbaren Prozessorelementen betrachtet, auf Architekturen mit anwendungsspezifischen Prozessorelementen, die im Fall synchroner Verarbeitung und Kommunikation auch als systolische Arrays bezeichnet werden, wird in Kapitel 3.3.4 eingegangen.

3.3.3.1.1 Massiv parallele Architekturen

Die Kommunikation erfolgt bei massiv parallelen Architekturen meist nur zu den vier bzw. acht nächsten Nachbarn, bei manchen Architekturen gibt es zusätzliche Broadcast Mechanismen bzw. Pfade für zeilen- bzw. spaltenweise Kommunikation. Multi-Level-Architekturen haben zusätzlich pyramidenförmige Verbindungspfade, die sich zur Implementierung hierarchischer Algorithmen eignen, die Bilder mit hierarchisch abgestufter Auflösung verarbeiten.

PAPIA2

Bei der PAPIA2 Architektur [BiCa93], die an der Universität Pavia entwickelt wurde, handelt es sich um eine massiv parallele Architektur, die als virtuelle Pyramide organisiert ist. Jedem PE ist genau einem Bildpunkt zugeordnet, insgesamt besteht die Architektur aus 512x512 gleichartig aufgebauten PEs, von denen jeweils 16x8 PEs auf einem Chip integriert sind. Die pyramidenförmige Kommunikationsstruktur gestattet die Unterstützung von Multi-Resolution Algorithmen.

Die PEs arbeiten bit-seriell und können boolsche sowie arithmetische Operationen ausführen. Sie verfügen über einen Summierer und 256 bit Speicher für Bilddaten und Verarbeitungsergebnisse. Jedes PE besitzt ein Mask-Register, so daß einzelne PEs von der Befehlsausführung ausgenommen werden können. Datenaustauch ist bidirektional zu den acht nächsten Nachbarn möglich. Ein PE kann logische Operationen der Gestalt A = A $OPR_1(OPR_2(NN))$ ausführen, wobei OPR_1 einer der Operationen AND/OR/EXOR/EXNOR, OPR_2 einer der Operationen AND/NAND/OR/NOR und NN eine Teilmenge der nächsten Nachbarn bezeichnet. Mit diesen Operationen können binäre Erosion und Dilatation für kleine Strukturelemente (3x3 Punkte) direkt implementiert werden. Für größere Strukturelemente muß eine Dekomposition in Strukturelemente verketteter Operationen erfolgen. Die Markierung von Segmenten kann als iterativer Prozeß implementiert werden. Da allerdings keine Möglichkeit zur Ermittlung einer globalen Bedingung vorhanden ist, muß eine feste, garantiert ausreichende Anzahl an Iterationen ausgeführt werden. Grauwert Erosion und Dilatation können die Verbindungsstruktur nicht effizient nutzen, was deren Dauer für Wortbreite n auf 92*n-9 Takte gegenüber einem Takt für binäre Erosion und Dilatation erhöht.

CAPP, HDPP

Die Prozessoren CAPP und HDPP [GeHe96] zielen auf Echtzeit Low-Level-Bildverarbeitung und umfassen ebenfalls ein Prozessorelement pro Bildpunkt. Beim CAPP handelt es sich um einen Prozessor, der inhaltsadressierbaren Speicher (CAM) benutzt, und in CCD-CMOS Technologie gefertigt wurde. Der HDPP hingegen benutzt DRAM zur Datenspeicherung. Beide Architekturen bieten einen ähnlichen Funktionsumfang.

Der CAPP besitzt 64x1 trit Speicher (dreiwertige Logik) pro PE, der in mehrere Felder unterteilt ist. Suchoperation und bedingtes Schreiben entsprechen Zugriffen auf den CAM-Speicher. Zur Kommunikation der PEs bestehen Verbindungen zu den vier nächsten Nachbarn. Der CAPP arbeitet mit hybriden Operationen, bestehend aus Datentransfer und Berechnung. Ein Aktivitätsregister gestattet mit dem vorherigen Verarbeitungsergebnis verknüpfte Berechnungen.

Der HDPP besitzt 128x1 bit DRAM Speicher pro PE, wobei die Logik der PEs innerhalb der

DRAM Dekoderlogik integriert ist. Jedes PE besitzt Verbindungen zu den vier nächsten Nachbarn und enthält neben einem Funktionseinheit fünf Registern (Lesen, 2xPuffern, Schreiben, Write-Enable). Insgesamt besteht eine Operation aus Speicheroperation, boolscher Funktion, Kommunikation und Register Laden. Wegen der für den Datentransport über Chipgrenzen hinweg nötigen Zeit werden Daten von Nachbar-PEs nicht in die Verarbeitung mit einbezogen.

Die Steuerung erfolgt mittels Mikroprogramm. Ein als Schieberegister aufgebautes Select-Register übergibt globale Konstanten an das Prozessor-Array. Der Status der PEs kann über ein sogenanntes Array-Register abgefragt werden. Statuswerte können in dem Ausgangsschieberegister akkumuliert werden. Eine globale OR-Logik, deren Ausgang wieder direkt an die PEs geführt ist, verknüpft die Statusausgänge der PEs. Für die Bildung der Differenz zum Nachbar bei der Wortbreite n, benötigt ein Prozessorelement des CAPP 7*n Taktzyklen, und ein Prozessorelement des HDPP 4*n Taktzyklen. Die Gauß-Filterung mit Kernel-Anpassung und anschließender Schwellwertoperation eines Bildes mit 7 bit Wortbreite dauert im Fall der CAPP Architektur 4,3 ms bzw. 3,1ms im Falle der HDPP Architektur. CAPP und HDPP benötigen einen Taktzyklus pro Instruktion. In [GeHe96] wird ein Takt von 10 MHz angenommen, den ein gefertigter CAPP Chip mit 16x16 PEs jedoch nicht erreicht.

PAVLOV

Bei der PAVLOV-Architektur [KrKa99] (Parallel Array for VoLume prOcessing and Viewing) handelt es sich ebenfalls um eine zweidimensionale Matrixanordnung von PEs mit lokalem Speicher und direkten Verbindungen zu den nächsten Nachbarn. Für die Unterstützung von Volumendaten bei medizinischen Anwendungen wurde die Architektur um konfliktfreien Zugriff auf Schnitte durch dreidimensionale Daten entlang der Hauptachsen erweitert. Auch bei dieser Architektur hat es sich als vorteilhaft erwiesen, mit einer kleinen Wortbreite zu arbeiten: Im Gegensatz zu einem früheren Konzept mit 8x8 PEs und 8 bit Datenpfad [KrKa98] gestatten kleinere PEs mit einem 1 bit Datenpfad die Integration einer 64x64 PE Matrix und schnelleren Takt, so daß sich ein besseres Verhältnis von Aufwand zu Nutzen ergibt.

ACMAA

Die ACMAA-Architektur [BaIr93] (Access Constrained Memory Array Architektur), die ursprünglich von Scherson und Ma [ScMa87] und von Hwang [HwTs89] vorgeschlagen wurde, stellt ein anderes massiv paralleles Architekturkonzept dar. Es handelt sich um ein Konzept für die Organisation und Koppelung von NxN Speichermodulen und N-Prozessoren. Die Speichermodule sind als Array mit Zeilen und Spaltenbussen organisiert, die jeweils einem Prozessor fest zugeordnet sind. Ein Bus-Controller schaltet zwischen den Betriebsarten Zeilen-Zugriff, Spalten-Zugriff, Zeilen-Broadcast und Spalten-Broadcast um, d.h. jeder Prozessor kann auf ein beliebiges Speichermodul oder auf alle Speichermodule in der ihm zugeordneten Zeile oder Spalte zugreifen. Die Untersuchung verschiedener Algorithmen ergab, daß für Operationen wie Flächen-, Umfangs- und Histogramm-Bestimmung eine Verringerung auf lineare Komplexität erreicht werden kann - gegenüber quadratischer Komplexität bei einer sequentiellen Architektur - während bei der Markierung zusammenhängender Bereiche keine Reduzierung der quadratischen Komplexität erzielbar ist. Diese Speicherorganisation ist vorteilhaft, da sie sich für Parallelisierung von nachbarschaftsbasierten Operationen und von globalen Operationen eignet. Nachteilig ist jedoch, daß die Architektur auf schnell zugreifbare Speicher angewiesen ist, und keine Möglichkeit zum Aufbau einer Speicherhierarchie besteht. Die feine Aufteilung des Speichers verursacht aufgrund des pro Speichermodul nötigen Aufwandes hohen Flächenbedarf. Lange Busse begrenzen die Geschwindigkeit der Speicheranbindung und führen aufgrund ihrer großen Anzahl zu Verdrahtungsproblemen.

Einsatzmöglichkeiten

Massiv parallele Architekturen führen binäre Operationen sehr schnell aus, jedoch ist die Skalierung der verarbeiteten Wortbreite oftmals nicht effizient, da geschwindigkeitssteigernde Spezialbefehle nur mit 1 bit Operanden verwendet werden können, so daß Grauwert-Operationen überproportional lange dauern. Massiv parallele Architekturen sind dann sehr effizient, wenn das Problem exakt zur Architektur paßt, sie sind jedoch ungeeignet für Probleme mit anderer Kommunikationsstruktur und für High-Level-Algorithmen [Cro99].

Auch die Bildgröße ist bei massiv parallelen Architekturen durch die Anzahl der PEs festgelegt. Implementierungsprobleme ergeben sich an den Chipgrenzen, da diese das Timing verlangsamen und eine große Anzahl an Verbindungen zu Nachbarzellen über diese hinweg läuft. Auch globale Signale zur Überwachung der Aktivität und zur Verteilung von Daten begrenzen die erreichbare Taktfrequenz. Die Implementierung verteilter Speicher ist vergleichsweise ungünstig, da Register bzw. Speichermodule wenig flächeneffizient sind.

Im Vergleich zur Effizienz bei Operationen, die unabhängig voneinander die einzelnen Bildpunkte verarbeiten, erreichen massive parallele Architekturen bei Operationen, die auf Ausbreitungsprozessen beruhen, lediglich geringe Effizienz. Dies ist dadurch bedingt, daß bei Ausbreitungsprozessen Operationen nur an der Ausbreitungsfront parallel stattfinden können, so daß sich ein insgesamt geringer Nutzungsgrad der Prozessorelemente ergibt. Für eine massiv parallele, dedizierte Implementierung des Wasserscheide-Verfahren ist beispielsweise in [Nog97] für eine Bildgröße von 128x128 Punkten ein Nutzungsgrad von nur 0,09% angegeben. Mit Vorverarbeitung läßt sich der Nutzungsgrad auf lediglich 0,17% verbessern. Im Kombination mit dem Verzicht auf die Nutzung von Bitparallelität ergibt sich eine sehr schlechte Effizienz und eine geringe Geschwindigkeit obwohl beim Wasserscheide-Verfahren ein vergleichsweise großes Potential an Datenparallelität gegeben ist.

Aufgrund der großen Anzahl an Prozessorelementen ergibt sich für massiv parallele Architekturen ein hoher Flächenbedarf. Da diesem hohe Kosten, eine herstellungsbedingt begrenzte Chipfläche und langsame Kommunikation über Chipgrenzen hinweg gegenüberstehen, ist ein kompakter Aufbau der Prozessorelemente wichtig. Trotz Verzicht auf die Nutzung von Bitparallelität und trotz Flächenoptimierung der PEs wird bei massiv parallelen Architekturen keine so kleine Chipfläche erzielt, daß alle PEs auf einen Chip integriert werden können, typischerweise finden zwischen 128 und 4096 Prozessorelemente auf einen Chip Platz.

3.3.3.1.2 Architekturen mit reduzierter Array-Größe

Alternativ zur Kommuniaktion über Chipgrenzen hinweg kann die Anzahl an PEs reduziert werden. Eine feste Zuordnung von Prozessorelementen zu Bildpunkten ist bei reduzierter Anzahl von PEs nicht mehr gegeben, stattdessen können z.B. alle Punkte innerhalb eines Bildausschnittes einem Prozessorelement zugeordnet werden. Problematisch ist dabei jedoch, daß die PEs eine verhältnismäßig große Datenmenge speichern müssen, so daß das Platzproblem auf diese Weise nicht gelöst wird. Daher wird bei Architekturen mit reduzierter Array-Größe, weiterhin jedem PE einen Bildpunkt zugeordnet, so daß lediglich ein Ausschnitt eines Bildes gleichzeitig gespeichert und verarbeitet wird.

VSP

Dieses Konzept liegt beispielsweise dem für Videocodierung entworfenem VSP [GoMe95] zugrunde. Der VSP besteht aus einem 32 bit Universalprozessor plus einem 16x16 Array von PEs, das gegebenenfalls in 4 Quadranten zu je 8x8 PEs aufgeteilt werden kann, was zu typischen Blockgrößen in Videocodierungsalgorithmen korrespondiert. Die Architektur verarbeitet Bilder als Sequenzen von Bildausschnitten, die auf das PE-Array abgebildet werden, während der Universalprozessor High-Level-Algorithmen und die Systemsteuerung übernimmt. Der Datentrans-

fer vom DRAM zum PE-Array erfolgt über einen Pufferspeicher, der die interne Operation vom externen Speicher entkoppelt, und über den sogenannten Corner-Turn Puffer, der den Datenstrom auf das Array verteilt. Der Datenzufuhr liegt das Paradigma kommunizierender sequentieller Prozesse zugrunde. Das PE-Array wurde mittels Full-Custom-Entwurf mit einem 0,7µm CMOS-Prozeß implementiert und erfordert eine Fläche von 8 mm x 7 mm. Die Architektur besteht insgesamt aus 1,7 M Transistoren und erreicht einen Systemtakt von 80 MHz.

Morphosys

Auch die Morphosys-Architektur [LeSi00] besteht aus einem Array reduzierter Größe. Es setzt sich aus 8x8 rekonfigurierbaren Zellen und einem TinyRISC Prozessor zusammen. Eine rekonfigurierbare Zelle enthält eine ALU/Multiplizierer-Einheit und einen Block von 4 Registern. Der RISC-Prozessor wählt aus den Konfigurationsworten im Context Memory einen Satz von acht Worten aus, die zeilen- oder spaltenweise an das rekonfigurierbare Array verteilt werden. Nicht genutzte Konfigurationsworte können im Hintergrund geladen werden. Für die Datenzufuhr zum rekonfigurierbaren Array dienen zwei 128x64 bit große Pufferspeicher, die alternierend eingesetzt werden können. Die maximale Taktfrequenz der in CMOS 0,35 µm / 3,3 V Technologie implementierten Architektur beträgt 100 MHz.

ISATEC

Die Parallelrechnerkarte [Isatec] der Firma ISATEC arbeitet nach den Prinzip instruktions-systolischer Arrays, d.h alle PEs auf einer Diagonalen führen gleichzeitig denselben Befehl aus. Es handelt sich dabei um eine PC-Einsteckkarte mit 32x32 PEs mit 1 bit Wortbreite in 4x4 Chips. Die Befehlseingabe in das Array erfolgt links oben, die Befehle werden Zeilen- und Spaltenweise weitergereicht. Eine Maskierung der Befehlsausführung ist mit UND-verknüpften Zeilen- und Spaltenselektoren möglich. Der Datenaustausch erfolgt durch ein Kommunikationsregister, das Verbindungen zu 4 Nachbarn besitzt. Außerdem ist die Verteilung von Daten innerhalb einer Zeile möglich. Die Datenversorgung erfolgt über Randprozessoren und Speicherblöcke im Norden und Westen des Prozessor-Arrays. Die Taktfrequenz beträgt 50 MHz, typischerweise werden über 1000 MIPS bei 16 bit Wortbreite erreicht.

Einsatzmöglichkeiten

Bei Architekturen mit reduzierter Array-Größe ist zur laufenden Verarbeitung eine parallele Datenzufuhr erforderlich, um eine hohe Auslastung des PE-Arrays zu erreichen. Wenn die Verarbeitung aus pipelinebaren Operationen besteht, kann der Datenfluß - wie im Fall der ISATEC Karte - über PEs hinweg so erfolgen, daß dabei die verschiedenen Schritte der Operation von den nacheinander durchlaufenen PEs übernommen werden, bei den anderen Architekturen erfolgt eine Aufteilung des Verarbeitungsbereiches in Blöcke. Während bei regulären Operationen und bei blockbasierten Operationen eine Aufteilung des Bildbereiches in Blöcke unproblematisch ist, besteht bei den für Segmentierung wichtigen, auf Ausbreitungsprozessen beruhenden Operationen die Schwierigkeit, daß während eines Ausbreitungsprozesses bearbeitete Bereiche in unvorhersehbarer Weise in Bereiche verschiedener Blöcke fallen. Um eine Aufteilung vornehmen zu können, müssen bei Ausbreitungsprozessen die Verarbeitungsergebnisse an den Blockgrenzen abgeglichen werden, was wiederholte nochmalige Bearbeitung von Blöcken zur Folge hat. In [KlSa98] wurden eine Aufteilung für das Wasserscheide-Verfahren untersucht, wobei sich trotz geringer Aufteilung auf lediglich 4 Prozessoren nur ein geringer Speedup von 1,92 ergab. Da eine stärkere Aufteilung von Bildern, wie sie bei Architekturen mit reduzierter Array-Größe erforderlich wäre, eine noch geringere Auslastung ergäbe, sind derartige Architekturen für Segmentierungsverfahren unvorteilhaft.

3.3.3.2 Befehlsparallele problemangepaßte Architekturen

Im Unterschied zum vorherigen Kapitel wird hier auf problemangepaßte Architekturen eingegangen, die hauptsächlich Befehlsparallelität nutzen. Dabei werden Architekturen betrachtet, die aufgrund ihrer Architekturmerkmale für bestimmte Operationen optimiert sind und sich für die Implementierung in Kapitel 2.4 vorgestellter Operationen eignen. Das Augenmerk liegt auf flexiblen Architekturen, die nicht auf eine bestimmte Operation festgelegt sind, sondern sich für mehrere Operation einsetzen lassen. Dazu zählen konfigurierbare Architekturen, die im Unterschied zu den zuvor beschriebenen konfigurierbaren Systemen eine feste Grundarchitektur besitzen und parametrisierbare Architekturen, deren Operationsweise lediglich durch Setzen von Parametern gesteuert wird. Da eine strenge Abgrenzung zwischen beiden Arten schwierig ist und lediglich eine Frage des Grades an Flexibilität darstellt, wird im Folgenden auf diese Unterscheidung verzichtet, und der Begriff konfigurierbare Architektur für beide Arten verwendet.

3.3.3.2.1 Architekturen für reguläre Bildoperationen

Da viele Operationen der Videosignalverarbeitung auf der zeilenweisen Verarbeitung von Bilddaten beruhen, bei der eine Nachbarschaft von Eingangsdaten benötigt wird, werden zunächst konfigurierbare Architekturen betrachtet, die diese Art der Datenadressierung unterstützen.

RIC

Der Reconfigurable Image Coprocessor (RIC) [WaHo96] ist eine Architektur für morphologische Verarbeitung von Binärbildern, die aus zwei 256kx1 bit RAMs und einem Xilinx XC4006 FPGA besteht, die an einen Host-Computer angeschlossen sind. Bild 3.10 zeigt ein Blockdiagramm des RIC. Der Morphological Processing Block (MPB) beinhaltet mehrere PEs, die jeweils eine Basisfunktion wie Komplement, Subtraktion, Dilatation und Erosion für eine 3x3 Nachbarschaft durchführen, und das im Kernel Store (KS) enthaltene Strukturelement benutzen.

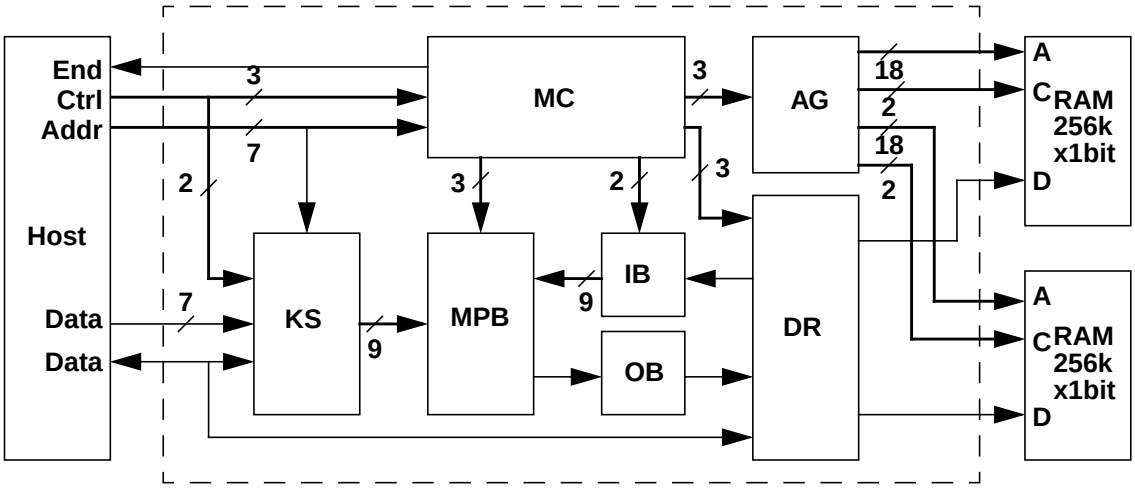


Bild 3.10: Blockdiagramm des Reconfigurable Image Coprocessor

Der Adressgenerator (AG) adressiert die RAMs und erzeugt die Steuersignale für die RAMs, der Data Router (DR) leitet die Daten von einem der RAMs an den Einganspuffer (IB) und die Daten vom Ausgangspuffer (OB) an das andere RAM. Der Ablauf der Operationen, der einen Durchlauf (z.B. Erosion, Dilatation) oder mehrere Durchläufe (z.B. Öffnen, Schließen) umfassen kann, wird vom Master Controller (MC) gesteuert. Die beiden RAMs, die jeweils ein 512x512 Punkte großes Bild aufnehmen können, speichern in wechselnder Abfolge das Eingangs- und Ausgangsbild. Vor und nach Operationen kann der Host-Computer auf die RAMs zugreifen, um die Eingangsdaten in eines der RAMs zu schreiben bzw. die Ausgangsdaten aus dem anderen zu laden.

Der externe Prozessor lädt außerdem das Strukturelement, wählt die Operation und startet die Ausführung.

PIMM-1, PIMM-10

Beim PIMM-1 [PIMM1], [Lem96] handelt es sich um eine Architektur, die morphologische und Punkt-zu-Punkt Transformationen für Binärbilder und 8 bit Grauwertbilder unterstützt, die als quadratisches oder hexagonales Gitter organisiert sind. PIMM-1 unterstützt sowohl nicht-rekursive als auch rekursive Verarbeitung. Durch Parallelschaltung zweier PIMM-1-Chips lassen sich 16 bit Daten verarbeiten. Zur Implementierung verketteter Operationen lassen sich mehrere PIMM-1-Chips als Pipeline zusammenschalten.

Die Datenzufuhr erfolgt über zwei 8 bit Dateneingänge (DIA und DIB) und einen Carry-Eingang (DIC), das Verarbeitungsergebnis wird über einen 8 bit Datenausgang (DIO) ausgegeben. Um die Bandbreite für die Bereitstellung der Eingangsdaten klein zu halten, wird mit einem aus drei externen Delay Lines bestehenden Zeilenspeicher gearbeitet, der drei Bildzeilen zwischenspeichert.

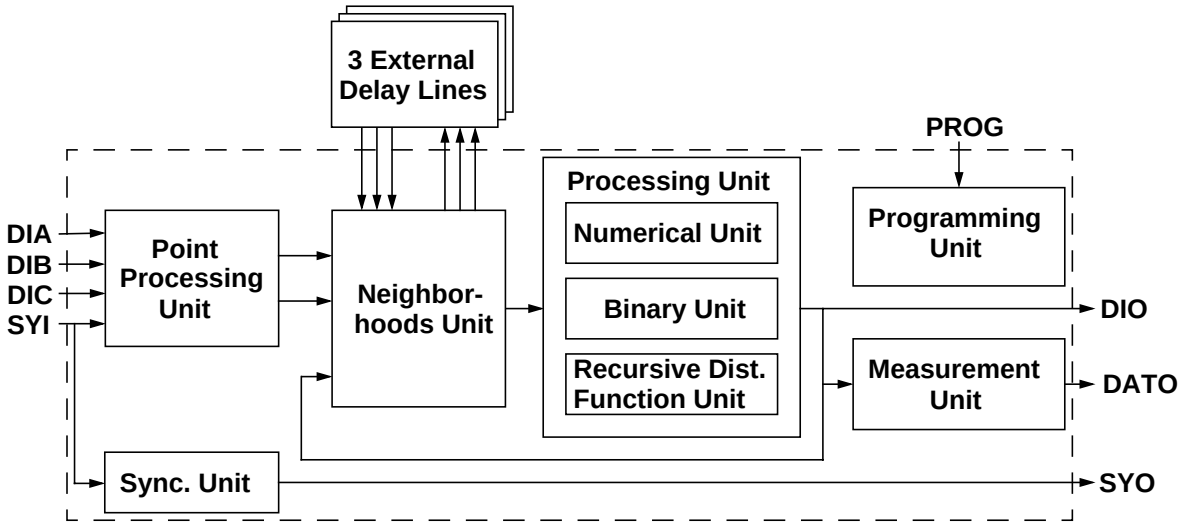


Bild 3.11: Blockdiagramm der PIMM-1 Architektur

Die in Bild 3.11 gezeigte Architektur wird über 34 Register der Programming Unit von einem Host konfiguriert. Die Point Processing Unit übernimmt die Vorverarbeitung, die neben reiner Weitergabe der Daten im numerischen Modus entweder Addition, Subtraktion, Minimum, Maximum, Schwellwert oder Sättigung sein kann, bzw. im Binärmodus eine beliebige boolsche Operation. Die Neighborhoods Unit stellt unter Verwendung der externen Delay Lines eine 3x3 Nachbarschaft der Eingangsdaten bereit, im numerischen Modus eine 8 bit Nachbarschaft (plus einer binären), im Binärmodus ausschließlich 8 binäre Nachbarschaften.

Diese Nachbarschaften werden von der aus Binary Unit, Numerical Unit, und Distance Function Processor bestehenden Processing Unit weiterverarbeitet. Die Binary Unit besteht aus 8 binären Prozessoren mit je einem eigenen Strukturelement, deren Funktionsumfang die Hit-or-Miss-Transformation, Erosion und Dilatation umfaßt. Drei der Prozessoren stellen zusätzliche Spezialfunktionen bereit (Table-Look-Up, Bestimmung des ersten weissen Punktes, binäre Rekonstruktion). Die binären Prozessoren können entweder als Pipeline oder parallel zusammen geschaltet werden. Die zur Verarbeitung von Grauwertbildern dienende Numerical Unit besteht aus zwei Modulen zur parallelen Bestimmung von Minimum und Maximum zweier unabhängiger Nachbarschaften, die Ergebnisse werden durch eine ALU verknüpft. Ein Distance Function Processor gestattet die Berechnung der Distanz für quadratische, hexagonale und dodekagonale Gitter. Die

Berechnung von Flächen in Binärbildern kann von der Measurement Unit durchgeführt werden. Die Synchronisation des Videosignals und die Pipeline-Steuerung übernimmt die Synchronisation Unit.

Der PIMM-1-Chip ist mit einer CMOS-Technologie der Strukturgröße 1,5 µm in einem Gehäuse mit 144 Pins implementiert. Bei der Berechnung von Distanzfunktionen kann der Takt maximal 10 MHz betragen, während ansonsten 20 MHz möglich sind.

Beim PIMM-10 [Lem96] handelt es sich um eine Weiterentwicklung der PIMM-1 Architektur, die eine höhere Taktfrequenz erreicht und zwei Bildpunkte parallel verarbeitet. Der Aufbau ist mit dem PIMM-1 vergleichbar, wobei die Verarbeitungseinheiten doppelt vorhanden sind. Die Architektur umfaßt 60 kGatter und 9 kByte FIFO-Speicher. Bei Fertigung mit einer Atmel-ES2-Technologie mit der Strukturgröße 0,7 µm wäre eine Fläche von 180 mm^2 erforderlich und eine Taktfrequenz von 40 MHz erreichbar. Die Architektur wurde jedoch nicht gefertigt, sondern stattdessen ein Emulator mit vier PIMM-1-Chips und vier FPGAs aufgebaut.

MoM-PDA

Die an der Uni Kaiserslautern entwickelte MoM-PDA-Architektur [HaHe98] (Map oriented Machine with Parallel Data Access) ist eine datenstromgesteuerte Architektur, deren Verarbeitungseinheit bzw. rekonfigurierbare ALU durch ein Kress-Array implementiert wird. Bild 3.12 (a) zeigt den Aufbau der Architektur.

Die Datenzufuhr erfolgt durch einen Data Sequencer, der einen zweidimensional organisierten Datenspeicher adressiert, zu ladende Daten über das sogenannte Smart Interface der rekonfigurierbaren ALU zur Verfügung stellt und deren Verarbeitungsergebnisse abspeichert. Die Eingangsdaten der Verarbeitung befinden sich innerhalb eines Scan-Fensters, das einen Ausschnitt aus den Daten des zu verarbeitenden Bildes enthält und entsprechend einem definierten Scan-Pfad verschoben wird. Um parallele Zugriffe zu ermöglichen sind benachbarte Bildzeilen in unterschiedlichen Speicherbänken abgelegt, und es führen zwei Datenpfade von und zum Kress-Array. Das Smart Interface implementiert das Scan-Fenster und vermeidet dabei mehrfaches Laden derselben Daten im Falle überlappender Scan-Fenster und nutzt Burst-Transfers zur Optimierung der Speicherzugriffe.

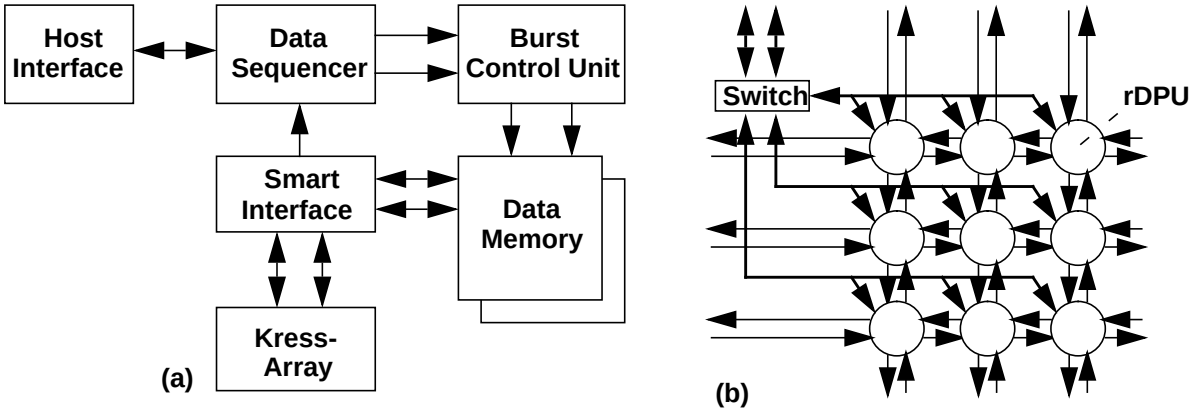


Bild 3.12: Blockdiagramm der (a) MoM-PDA Architektur und des (b) Kress Arrays

Der Gundgedanke des Kress-Arrays ist die Nutzung hoher Regularität durch Full-Custom-Entwurf zur Erzielung einer hohen Integrationsdichte. Das Kress-Array arbeitet auf Operator Ebene mit 32 bit Worten. Ihm liegt eine Gitterarchitektur zugrunde. Der Aufbau eines Arrays aus 3x3 Prozessorelementen, genannt rDPUs, ist in Bild 3.12 (b) gezeigt. Die rDPUs können arithmetische Operationen und/oder Routing-Operationen ausführen. Kommunikation ist zu den vier

nächsten Nachbarn mit 32 bit Wortbreite möglich, über Chipgrenzen hinweg erfolgt serielle Kommunikation, um die Pinzahl zu reduzieren. Zusätzlich existiert ein hierarchisches Routing-Netzwerk. Die rPDUs enthalten Register und eine ALU, die alle Operationen der Programmiersprache C abdeckt. Vier umschaltbare Konfigurationssätze gestatten schnelle Kontextwechsel und Konfiguration im Hintergrund.

Der Data Sequencer wird durch ein Altera FLEX10K100 FPGA implementiert, das Smart Interface durch ein Xilinx XC6216 FPGA, der Datenspeicher durch MDRAMs und die rekonfigurierbare ALU durch ein KressArrayIII. Die Konfiguration des KressArrays wird mit einem Hochsprachen-Compiler erzeugt, der auch die Optimierung der Ladezugriffe vornimmt. Ein lineares 3x3 FIR-Filter, das 2 Resultate parallel berechnet, wurde mit 12x5 rPDUs implementiert. Bei größtmöglicher Parallelisierung lassen sich 8 Ergebnisse parallel in 19 Taktzyklen berechnen. Bei der erzielbaren Taktfrequenz von 25 MHz lassen sich maximal $10{,}5 \cdot 10^6$ Bildpunkte pro Sekunde filtern.

Einsatzmöglichkeiten

Die vorgestellten Architekturen für reguläre Bildoperationen unterscheiden sich in der Wortbreite der verarbeiteten Daten. Während sich der RIC mit einer Wortbreite von 1 bit nur für binäre Operationen einsetzen läßt, gestattet der PIMM-1/10 die Verarbeitung von Videodaten mit 8 bit Wortbreite. Die MoM-PDA Architektur hat mit 32 bit eine deutlich zu große Wortbreite für die Verarbeitung von Grauwertbildern, wobei sie sich ansonsten gut für reguläre Verarbeitung zweidimensionaler Datenstrukturen eignet.

Auch hinsichtlich der Effizienz der Datenzufuhr unterscheiden sich die Architekturen: Während der RIC und MoM-PDA lediglich eine Nachbarschaft an Eingangsdaten zwischenspeichern, benutzt der PIMM-1/10 einen Zeilenspeicher zur Pufferung von Bildzeilen. Dadurch benötigen RIC und MoM-PDA bei Operationen, die auf einer 3x3 Nachbarschaft basieren, im Vergleich zum PIMM-1/10 die dreifache Anzahl an Speicherzugriffen zum Laden der Eingangsdaten. Andererseits ist der Speicher des MoM-PDA in zwei Bänke unterteilt, so daß die doppelte Speicherbandbreite zur Verfügung steht und das Smart Interface deutlich flexibler im Bezug auf die Verarbeitungsabfolge ist.

Für den RIC ist in [WaHo96] keine Operationsdauer angegeben. Die Taktfrequenzen des PIMM-1/10 und des MoM-PDA sind mit 20 MHz bzw. 25 MHz vergleichsweise gering. Die niedrige Taktfrequenz der PIMM-1/10-Architektur ist durch die große Strukturgröße der verwendeten Technologie bedingt. Die Taktfrequenz des MoM-PDA-Prototyps läßt sich möglicherweise durch Austausch des mit FPGAs implementierten Kress-Arrays durch dedizierte Module erhöhen. Problematisch beim MoM-PDA-Konzept ist jedoch die Erweiterung des Kress-Arrays über Chipgrenzen hinweg, da aufgrund der hohen Zahl verbindender Signale langsame serielle Kommunikation eingesetzt werden muß.

Die Architekturen weisen einen unterschiedlichen Grad an Konfigurierbarkeit auf, am flexibelsten ist dabei die MoM-PDA-Architektur, bei der sich jedoch die unnötig hohe Wortbreite nachteilig auf die Flächeneffizienz auswirkt. Daß mögliche Freiheitsgrade durch die jeweils vorgegebene Grundarchitektur eingeschränkt sind, ist vorteilhaft, da weniger Aufwand für schaltbare Verbindungen und konfigurierbare Elemente erforderlich ist.

Während sich diese Architekturen gut für Bildoperationen eignen, die ganze Bilder regulär bearbeiten, scheiden sie zur Implementierung von Operationen aus, die auf Ausbreitungsprozessen beruhen, da sie rekursive Adressierung durch rekursive Verarbeitung emulieren müssen (siehe Kapitel 2.5.4). Im Fall des PIMM-1 dauert beispielsweise die Wasserscheide-Operation für ein 512x512 Punkte großes Bild 13 Sekunden, gegenüber 13,10 ms bei einer regulären Bildoperation. Da jedoch Operationen, die auf Ausbreitungsprozessen für Segmentierung ebenfalls eine große Rolle spielen, wird auf die Architekturen für diese Operationen im nächsten Kapitel einge-

gangen.

3.3.3.2.2 Architekturen für Ausbreitungsprozesse und Segmentoperationen

Architektur nach [NoMe95]

In [NoMe95] (und [MeNo95]) wurde eine Architektur für Queue-basierte morphologische Algorithmen beschrieben. Ausgehend von einer Algorithmenanalyse wurden zwei Schemata der Verarbeitung identifiziert, die als Grundlage der Hardware-Architektur dienen. Beide Schemata beinhalten einen Initialisierungsprozeß und einen Ausbreitungsprozeß, und unterscheiden sich darin, daß im einen Fall der Ausbreitungsprozeß in die Initialisierung eingebettet ist und im anderen Fall beide getrennt stattfinden. In beiden Fällen besteht die Möglichkeit zur Wahl einer klassischen bzw. einer hierarchischen Queue, der Art des Tests während der Initialisierungsphase und der Art des Tests während der Ausbreitungsphase.

Die Hardware-Architektur besteht aus zwei Einheiten. Einheit 1 umfaßt neben der Steuerung zwei Module für die Adressierung (Raster Address Processor und Neighborhood Extractor), eines für das Management einer hierarchischen Queue und die zwei Bildspeicher. Einheit 2 umfaßt den algorithmenorientierten Teil: Zwei Einheiten zur Durchführung von Tests und eine Einheit zur Durchführung von Operationen.

Neben der Taktfrequenz von 10 MHz sind jedoch weder Technologiedaten, Flächendaten noch Details über die Implementierung angegeben. Simulationsergebnisse gemittelt über mehrere Bilder im Format 512x512 geben im Vergleich zu einer SUN Sparc 10 eine Beschleunigung um den Faktor 3,1 bis 8,9 für die Operationen Binär-Rekonstruktion, Grauwert-Rekonstruktion, Schwellwert mit Hysterese, Wasserscheide-Verfahren, und Extremumsbestimmung. Aufgrund der Dauer von 3 Takten für einen Queue-Zugriff ergibt sich ein Verarbeitungszyklus von ca. 11 Takten pro Bildpunkt, was im ungünstigsten Fall einer Verarbeitungszeit von ca. 300 ms pro 512x512 Bild entspricht.

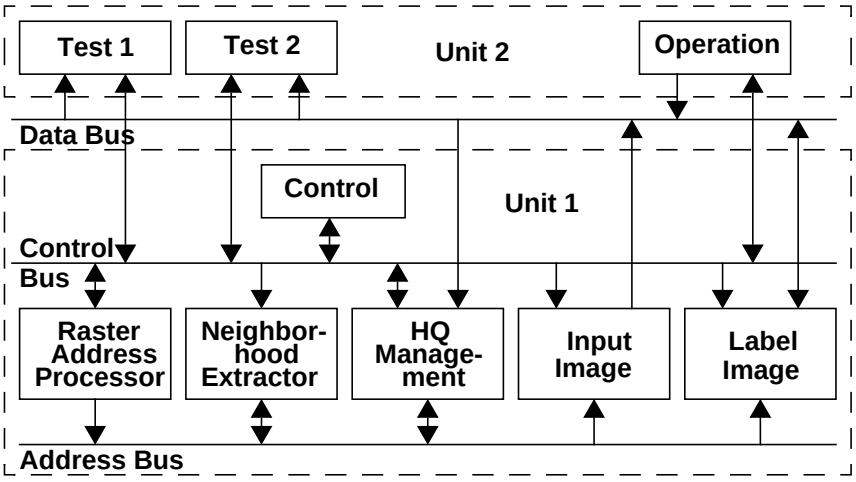


Bild 3.13: Aufbau der Architektur [NoMe95]

Architektur nach [SaKl99]

Die in [SaKl99] beschriebene Hardware/Software-Architektur dient zur Implementierung morphologischer Operationen, die auf der Nutzung hierarchischer Queues beruhen. Die Architektur besteht aus drei Einheiten, die Datenpfad und Steuerung umfassen. Die erste Einheit übernimmt Speicherzugriffe und arithmetisch-logische Operationen und ist RISC-artig aufgebaut. Die zweite Einheit besteht aus Speicher und Kontroller zur Verwaltung einer hierarchischen Queue.

Die dritte Einheit dient der Durchführung von Nachbarschaftsoperationen. Sie greift auf fünf Speicherbänke zu und enthält eine ALU die im SIMD-Modus operieren kann. Dadurch, daß die Bilddaten in fünf Speicherbänke untergebracht sind, können auf einen Bildpunkt und seine horizontalen und vertikalen Nachbarn parallel zugegriffen werden. Die in [SaKl99] beschriebene Architektur braucht bei einer Taktfrequenz von 100 MHz für die Filterung eines Bildes mit einem alternierenden sequentiellen Filter der Größe 5x5 im Verhältnis 770:960 weniger Zeit als ein Pentium mit 450 MHz. Weitere Implementierungsdaten sind jedoch nicht angegeben.

Einsatzmöglichkeiten

Die Architekturen [MeNo95] und [SaKl99] unterstützen sowohl Operationen, denen gewöhnliche Queues zugrundeliegen, als auch solche, denen hierarchische Queues zugrundeliegen. Damit können Ausbreitungsprozesse mit gleichmäßiger Geschwindigkeit und solche, bei denen die Ausbreitungsgeschwindigkeit von den Bilddaten abhängt, implementiert werden. Gegenüber der Emulation rekursiver Adressierung durch rekursive Verarbeitung ergibt sich ein deutlicher Geschwindigkeitsvorteil. Die Wasserscheide-Operation für ein 512x512 Punkte großes Bild dauert beispielsweise nur mehr 300 ms im Fall der [MeNo95] Architektur gegenüber 13 Sekunden im Fall des PIMM-1. Da jedoch in Algorithmen zur Videoobjekt-Segmentierung Operationen mit Inter-, Intra- und Segment-Adressierung eingesetzt werden, ist sowohl die Unterstützung regulärer Adressierung als auch die Unterstützung von Ausbreitungsprozessen erforderlich, bei Operationen mit Segment-Adressierung sogar beides in schnellem Wechsel. Somit ergibt sich Bedarf an einer Architektur die beide Arten von Operationen implementieren kann.

3.3.4 Spezialisierte Architekturen

Spezialisierte Architekturen werden zur Beschleunigung einzelner rechenintensiver Operationen, wie z.B. Erosion/Dilatation [DrAn92], [ChTs93] und Wasserscheide-Verfahren [KlLe95], [Nog97] eingesetzt, in manchen Fällen auch für komplette Algorithmen, wie z.B. Fahrbahnranderkennung [Sch94] und klassifikationsbasierte Segmentierung für automatische Fertigungslinien [MiBa97]. Auch in modularen Systeme, die - wie z.B. Cheops [BoWa95] - aus dedizierten Modulen bestehen und nach Einsetzen der Module einen bestimmten Algorithmus implementieren, kommen spezialisierteArchitektzuren zum Einsatz.

Spezialisierte Architekturen, wie z.B. [DrAn92], [ChTs93], [Sch94] und [MiBa97], werden hauptsächlich für Operationen eingesetzt, die Bilddaten zeilenweise verarbeiten, da bei diesen Operationen die Verarbeitung synchron zu einem durch zeilenweise Bildabtastung erzeugten Datenstrom erfolgen kann und so die Architekturen mit geringem Speicher auskommen.

Sie können durch starkes Pipelining sehr komplexe Operationen mit hoher Geschwindigkeit unterstützen. Die Architektur [ChTs93] implementiert z.B. eine Erosion/Dilatation mit einem Strukturelement der Größe 13x13, die in [DrAn92] beschriebene Architektur für Erosion/Dilatation verarbeitet 30 Bilder der Größe 512x512 pro Sekunde, die in [Nog97] beschriebene Architektur für das Wasserscheiden-Verfahren benötigt lediglich 10,65µs für ein 128x128 Bild und die Architektur für Klassifikationsbasierte Segmentierung verarbeitet 50 Bilder pro Sekunde im Format CCIR601.

Allerdings sind die Einsatzmöglichkeiten begrenzt, da spezialisierte Architekturen auf eine bestimmte Operation ausgelegt und optimiert sind. Wenn sie zur Beschleunigung einzelner Operationen einsetzt werden, müssen die restlichen Teile des Algorithmus, von anderen Hardware-Architekturen oder von einem Prozessor übernommen werden, so daß sich dieser Ansatz nur eignet, wenn die Komplexität der restlichen Operationen gering ist. Wenn ein kompletter Algorithmus mit dedizierten Architekturen implementiert wird, muß es sich um einen Algorithmus handeln, der aus wenigen unterschiedlichen Operationen besteht, für die eigene Einheiten vorgesehen werden müssen, damit eine Implementierung mit vertretbarer Komplexität möglich ist. In

Segmentierungsalgorithmen sind jedoch i.d.R. viele unterschiedliche Operationen enthalten, was einer kompakten Implementierung insbesondere für mobile Anwendungen entgegensteht. Bei experimentellen Systemen zur Algorithmenuntersuchung steht die Systemgröße nicht im Vordergrund, so daß sich spezialisierte Architekturen gut hierfür eignen. Aufgrund ihrer fehlenden Flexibilität lassen sich spezialisierte Architekturen nur für dedizierte Anwendungen einsetzen, und sind auch für die Unterstützung von adaptiven Algorithmen in der Regel zu unflexibel.

3.3.5 Weitere Ansätze

Als Alternative zu Prozessoren bzw. Prozessorsystemen, wurde in [PaAs97] das Konzept Intelligenten RAMs (IRAM) vorgeschlagen. Ansatzpunkt ist die Integration von Prozessor und Speicher auf einem Chip und deren Fertigung mit einem Speicher-Herstellungsprozeß. Im Vergleich zu embedded RAM, d.h. in Form großer Blöcke integriertem Speicher, sieht das IRAM-Konzept eine wesentliche engere Integration vor. Es verspricht Potential hinsichtlich Erzielung geringerer Speicherlatenz, höherer Speicherbandbreite und höherer Energieeffizienz, bietet jedoch nur Vorteile wenn eine Architektur in sehr hohen - mit DRAMs vergleichbaren - Stückzahlen hergestellt wird. Neben der Frage nach geeigneten Architekturen bestehen jedoch auch technologische Hindernisse, wie z.B. kleinere Tansistorgeschwindigkeit und weniger Metallebenen. Der Einsatz von IRAM für Videoobjekt-Segmentierung ist aufgrund der geforderten Stückzahlen nur zweckmäßig, wenn sich dieselbe Architektur auch für andere Anwendungen einsetzen läßt. Aus diesen Gründen wird dieser Ansatz für Verfahren der Videoobjekt-Segmentierung im weiteren nicht betrachtet.

Alternativ zur digitalen Verarbeitung wurde die Verwendung analoger Schaltungen vorgeschlagen. In [LuWy91] wird der Einsatz nichtlinearer analoger Schaltunger für die Aufteilung von Bildern in Segmente untersucht, die separat geglättet werden können. Motivation ist dabei die Erzielung einer hohen Geschwindigkeit. Die vorgeschlagenen Architekturen basieren auf der Formulierung der Aufgabenstellung als Optimierungsproblem. Das Anwendungsgebiet ist jedoch auf Linienprozesse bzw. Relaxation beschränkt.

Ein weiterer Ansatz, der auf analogen Schaltungen basiert sind zellulare neuronale Netzwerke, die abgekürzt als CNNs bezeichnet werden. Dabei handelt es sich um programmierbare analoge Array-Architekturen, die sowohl mit analogen Werten als auch mit logischen Werten arbeiten [RoCh93]. Während die Dynamik im Rahmen des allgemeinen CNN-Konzeptes durch ein System aus Differentialgleichungen mit Rückkopplungs- und Steuerungsanteil bestimmt wird, die jeweils aus einer beliebigen Nachbarschaft an Array-Elementen berechnet werden, sind reale CNN-Architekturen aus Implementierungsgründen auf eine Schicht mit linearer Verknüpfung einer 3x3 Nachbarschaft eingeschränkt [VeWe95]. Damit lassen sich lineare nachbarschaftsbasierte Operationen direkt und stückweise lineare nachbarschaftsbasierte Operationen durch Dekomposition implementieren [LiDo00]. Im Vergleich zu den zuvor beschriebenen massiv parallelen digitalen Architekturen kann mit analoge Berechnungen Parallelität auf Bitebene genutzt werden, jedoch ergibt sich eine begrenzte Rechengenauigkeit, die oftmals nur 7 bit beträgt. Wie bei digitalen massiv parallelen Architekturen ergibt sich durch die hohe Anzahl an Zellen ein Platzproblem, so daß wie z.B. bei [LiDo00] typischerweise nur 64x64 Zellen auf einem Chip implementiert werden. Solange die Ergebnisse von Berechnungen im Array gespeichert bleiben können ist einen schnelle Verarbeitung möglich, falls jedoch Bilder aus bzw. in das Array transportiert werden müssen vergrößert sich die Verarbeitungszeit auf ein Vielfaches. Außerdem problematisch ist notwendige Aufteilung von Bildern entsprechend der Arraygröße bzw. eine Kopplung mehrerer Arrays, bei der die analogen Signale über Chipgrenzen laufen müßten.

3.4 Zusammenfassung

Die für Echtzeitimplementierungen vorgeschlagenen Architekturen wurden in universelle, problemangepaßte und dedizierte Architekturen unterteilt. Die Klasse universeller Architekturen wurde nach Art der Flexibilität in Prozessoren/Prozessorsysteme und konfigurierbare Systeme aufgeteilt, die Klasse problemangepaßter Architekturen wurde nach der Art vorwiegend genutzter Parallelität in datenparallele problemangepaßte Architekturen und befehlsparallele problemangepaßte Architekturen untergliedert. Zu jeder Klasse wurden typische Beispiele vorgestellt, und diese hinsichtlich ihrer Eigenschaften diskutiert.

Bei Universalprozessoren mit Multimediaerweiterung wurde deutlich, daß das hohe Maß an Generalität, das mit nicht optimaler Verarbeitungsgeschwindigkeit, hohen Kosten und hohem Leistungsverbrauch verbunden ist, diese für den Einsatz in mobilen Endgeräten ungeeignet macht. Ein weiteres Problem ist die Datenversorgung, da Cache und Speicherschnittstelle von Universalprozessoren nicht auf Videosignalverarbeitung ausgerichtet sind. Im Hinblick auf den Einsatz für Segmentierung ist auch problematisch, daß Multimediaerweiterungen lediglich reguläre Operation unterstützen, die Datenparallelität aufweisen, nicht jedoch irreguläre Operationen wie sie bei Segment-Adressierung auftreten.

Videoprozessoren sind auf zweidimensionale Signalverarbeitung ausgerichtet, oftmals besonders auf Videocodierung. Videoprozessoren lassen sich jedoch nicht auf eine Grundarchitektur zurückführen, vielmehr gibt es Videoprozessoren die auf dem VLIW-, SIMD- oder MIMD-Konzept beruhen. VLIW-Prozessoren sind in der Lage ILP gut zu nutzen, ohne den Steuerungsaufwand superskalarer Prozessoren zu erfordern, der Gewinn an Verarbeitungsleistung ist jedoch durch Datenabhängigkeiten auf typischerweise 5-6 begrenzt. Nachteilig ist der hohe Aufwand für eine Crossbar zur Verbindung von Funktionseinheiten und Register bzw. die Erforderlichkeit eines Multi-Port-Registerblockes. Auch die hohe Bandbreite für die Befehlszufuhr und die lange Leerlaufzeit bei Befehls-Cache-Misses ist ungünstig. Einen Engpaß kann auch das Speicherinterface darstellen, das neben der Befehlszufuhr die Datenversorgung mehrerer Funktionseinheiten übernehmen muß. MIMD-Prozessorsysteme bieten zwar eine sehr hohe Flexibilität, die Programmierung ist jedoch sehr aufwendig, da zur Erzielung einer guten Beschleunigung eine sorgfältige Aufteilung eines Programmes auf die Prozessoren vorgenommen werden muß, so daß sich dieses Architekturkonzept nicht durchgesetzt hat. SIMD-Prozessoren eignen sich am besten zur Nutzung hoher Datenparallelität, vorteilhaft ist auch der verhältnismäßig geringe Aufwand für die Befehlszufuhr und Dekodierung. Das SIMD-Konzept eignet sich jedoch nicht besonders gut für Segment-Adressierung, bei der geringe Datenparallelität gegeben ist. Viele Videoprozessoren enthalten zusätzliche Spezialeinheiten zur Beschleunigung von Videocodierverfahren, die sich jedoch nicht für Videoobjekt-Segmentierung nutzen lassen.

Konfigurierbare Systeme sind sehr flexibel einsetzbar, jedoch haben auf diese Weise implementierte Architekturen einen mehrfach höheren Flächenverbrauch und erreichen deutlich geringere Taktfrequenzen. Sie eignen sich zur Implementierung von Architekturen, wenn ein hohes Maß an Parallelität und die Möglichkeiten der Umkonfiguration genutzt werden kann. Bei Segmentierungsverfahren ist eine schnelle Umkonfiguration mit dynamisch rekonfigurierbaren Chips erforderlich bzw. hoher Hardware-Aufwand für räumliches Zusammensetzen von Operationen. Die Aufteilung auf mehrere Chips ist jedoch bei Operationen, die auf Ausbreitungsprozessen beruhen, problematisch. Außerdem können interne Speicher nicht effizient implementiert werden, was die Möglichkeiten zur Datenpufferung bzw. zur schnellen Datenzufuhr einschränkt.

Die Kategorie datenparalleler problemangepaßter Architekturen umfaßt sowohl massiv parallele Architekturen, als auch Array-Architekturen mit reduzierter Array-Größe. Massiv parallele Systeme erreichen ein hohes Maß an Parallelität auf Bildpunktebene. Da jedoch wegen der hohen Anzahl an Prozessorelementen die Fläche einzelner PEs klein sein muß, wird bei massiv parallelen Architekturen auf die Nutzung von Bitparallelität verzichtet. Dies ist bei Ausbreitungsprozes-

sen äußerst ineffizient, da bei diesen der Nutzungsgrad der PEs im Promillebereich liegt und der hohe Hardware-Aufwand ungenützt beleibt. Zusätzlich ist die Größe verarbeiteter Bilder nicht skalierbar, und kann es vorkommen, daß Grauwertbilder im Gegensatz zu binären Bildern überproportional langsam verarbeitet werden, wenn geschwindigkeitssteigernde Spezialbefehle nicht genutzt werden können.

Array-Architekturen mit reduzierter Array-Größe umgehen zwar die Problematik hohen Flächenbedarfs, dafür ergibt sich jedoch die Notwendigkeit der Aufteilung des Verarbeitungsbereiches in Blöcke. Bei irregulären, auf Ausbreitungsprozessen beruhenden Operationen fällt jedoch der verarbeitete Bereich in mehrere Blöcke, so daß nach der Verarbeitung von Blöcken i.d.R ein mehrfacher Abgleich mit der Konsequenz wiederholter Bearbeitung erforderlich ist. Da bereits eine schwache Unterteilung zu geringer Effizienz führt, ist der Einsatz derartiger Architekturen nicht zweckmäßig.

Zur Klasse befehlsparalleler problemangepaßter Architekturen gehören Architekturen für reguläre Bildoperationen. Diese arbeiten mit einem Fenster, das entsprechend der Verarbeitungsabfolge über den Bildbereich geschoben wird und Daten für nachfolgende Verarbeitungseinheiten bereitstellt. Die Architekturen eigenen sich gut für die Verarbeitung ganzer Bilder, da sie zwar hinsichtlich der durchzuführenden Operation flexibel sind, jedoch aufgrund der vorgegebenen Grundarchitektur relativ wenig unnötige Ressourcen zur Implementierung von Flexibilität enthalten. Andere Architekturen eigenen sich zur Implementierung von Ausbreitungsprozessen und Segmentoperationen. Da jedoch zur Videoobjekt-Segmentierung beide Arten von Operationen erforderlich sind, ergibt sich ein Bedarf an Architekturen, die beide Arten von Operationen effizient unterstützten.

Bei Einschränkung der Anwendungsbreite und der unterstützten Algorithmenteile wäre auch der Einsatz von spezialisieren Architekturen möglich, die in der Regel die höchste Verarbeitungsleistung und den geringsten Fläche- und Leistungsverbrauch erreichen. Jedoch gehen algorithmische Änderungen in der Regel zu Lasten der Segmentierungsqualität. Außerdem sind die Einsatzmöglichkeiten spezialisierter Architekturen sehr begrenzt, da sie sich lediglich für die Beschleunigung einzelner Operationen bzw. zur Implementierung aus wenigen unterschiedlichen Operationen bestehender Algorithmen eignen.

4. Konzeption einer flexiblen Architektur

In diesem Kapitel wird ausgehend von den Ergebnissen der Algorithmenuntersuchung und dem Vergleich bekannter Architekturen das Grundkonzept der vorgeschlagenen Architektur entwikkelt. Der Ansatzpunkt ist die Trennung von Adressierung und Verarbeitung, mit dedizierter Implementierung der generischen Adressierungsarten und konfigurierbarer Implementierung der Verarbeitungsoperationen, da hierdurch ein hohes Maß an Flexibilität und Effizienz erzielt werden kann. Ausgehend von diesem Grundansatz wird abgeleitet, mit welcher Parallelisierungsstrategie ein hohes Maß an Parallelität bei gleichzeitig hoher Flexibilität erreicht werden kann, wie die verschiedenen Arten zu verarbeitender Daten organisiert werden müssen, und wie die Steuerung erfolgen kann. Nach der Entwicklung des Grundkonzepts wird die Integration in ein Gesamtsystem für objektbasierte Kodierung bzw. objektbasierte Videoanalyse dargestellt.

4.1 Grundkonzept

High-Level-Operationen sind, wie in Kapitel 2.4.1 erläutert, irregulär und datenabhängig, verarbeiten jedoch nur kleine Datenmengen. Aus diesem Grund lassen sich High-Level-Operationen am besten durch Universalprozessoren implementieren. Low-Level-Operationen sind dagegen regulär und rechenaufwendig und verarbeiten große Datenmengen. Aus diesem Grund eignet sich für die Echtzeitimplementierung kompletter Algorithmen am besten ein System aus einem Universalprozessor und einer speziell für Videosegmentierung ausgelegten flexiblen Architektur.

Aufgrund der Regularität und der geringen Anzahl generischer Adressierungsarten eignet sich die Adressierung der Bilddaten gut für die Implementierung in Form einer dedizierten Hardware-Architektur. Die Operationen für die Adreßrechnung können in einem dedizierten Modul schneller und kompakter durchgeführt werden als durch eine programmierbare Einheit, und durch speziell ausgelegte Pufferspeicher und breite Datenpfade lassen sich Daten mit größerer Bandbreite zugreifen, als in einer programmierbaren Einheit. Auch die Anordnung von Adreßrechnung, Laden, Operation und Speichern in Form einer Pipeline ist im Falle einer dedizierten Hardware-Architektur problemlos möglich. Aus diesen Gründen ist die Implementierung der Adressierung in Form dedizierter Hardware die am besten geeignete Lösung.

Um universelle Einsetzbarkeit im Bereich Videosegmentierung zu erreichen muß die Verarbeitung sehr flexibel sein. Wie in Kapitel 3.2.3 dargestellt, ist bei der Implementierung von Flexibilität zwischen den Konzepten Programmierbarkeit und Konfigurierbarkeit abzuwägen. Bei regulären, häufig wiederholten Operationen, sind konfigurierbare Architekturen gegenüber programmierbaren Architekturen vorteilhaft, da sie einen Packungsdichtevorteil gegenüber programmierbaren Architekturen hinsichtlich der pro Fläche und Zeit ausführbaren Operationen besitzen [DHo00], und da sie Parallelisierung in einem stärkeren Maße erlauben. Bei Low-Level-Operationen wiederholen sich die auf die einzelnen Bildpunkte angewandten Operationen typischerweise zwischen 100 und 10.000 mal. Da die Operationen außerdem von begrenzter Komplexität sind, ist für die Unterstützung der in Kapitel 2.4 gezeigten Operationen eine konfigurierbare Architektur am besten geeignet.

Das Grundkonzept beinhaltet somit die Trennung von Adressierung und Verarbeitung, mit dedizierter Implementierung der generischen Adressierungsarten und konfigurierbarer Implementierung der Verarbeitungsoperationen. Der Ansatz liegt zwischen einem Prozessor und einem dedizierten ASIC, und zielt darauf ab, die Vorteile der hohen Flexibilität eines Prozessors und der hohen Geschwindigkeit eines ASICs miteinander zu verbinden.

4.1.1 Adressierung

Ein Hardware-Beschleuniger für Videosegmentierung muß die in Kapitel 2.5 identifizierten generischen Adressierungsmethoden und die in Kapitel 2.4 aufgelisteten Verarbeitungsfunktionen unterstützen.

Für die Implementierung regulärer Low-Level-Operationen muß Inter- und Intra-Adressierung unterstützt werden. Operationen, die räumlichen Zusammenhang berücksichtigen oder bestimmen, können durch rekursive Adressierung oder rekursive Verarbeitung unterstützt werden. Rekursive Adressierung ist vorzuziehen, da rekursive Verarbeitung die wiederholte Verarbeitung des ganzen Bildbereiches erfordert, die Anzahl der Wiederholungen von den Bilddaten abhängt und die Operationen für mehrere Bildpunkte nicht überlappend durchgeführt werden können. Da zum Satz der zu implementierenden Operationen auch solche gehören, die Distanzfelder berechnen, und solche, die eine gleichmäßige Ausbreitung mehrerer Bereiche implementieren, muß es die Segment-Adressierung ermöglichen, Bereiche in Reihenfolge steigender geodesischer Distanz zu bearbeiten. Die Implementierung der Segment-Adressierung mit einer alternierenden LIFO wird gegenüber der Implementierung mit einer FIFO vorgezogen, da sich diese besser auf mehrere Hierarchiestufen erweitern läßt, und sich gleichzeitiges Entnehmen und Ablegen von Daten schneller durchführen läßt. Für die Unterstützung von Ausbreitungsprozessen mit adaptiver Geschwindigkeit (wie z.B. im Falle des Wasserscheide-Verfahrens) ist die Erweiterung zu einer hierarchischen LIFO erforderlich. Neben PQI muß auch PQO unterstützt werden, um sowohl Funktionen zu unterstützen, die Bildpunkte während des Ausbreitungsprozesses lokal bearbeiten, als auch solche, die eine Segmentnummer bzw. einen Distanzwert propagieren. Da bei Segment-Adressierung der Verarbeitungszustand der Bildpunkte für Verarbeitung und für Adressierung benötigt wird, muß dieser automatisch verwaltet werden.

Die Suche nach Startpunkten, die mit der Intra-Adressierung vergleichbar ist, erfolgt bei manchen Operationen vorab im kompletten Bild, und bei anderen Operationen schrittweise im Wechsel mit Ausbreitungsprozessen. Aufgrund der engen Verzahnung von regulärer und irregulärer Adressierung und Ausbreitungsprozessen ist es wichtig, daß beide Adressierungsarten von derselben Architektur übernommen werden. Intra-, Inter- und Segment-Adressierung stellen Betriebsmodi dar, die Art und Reihenfolge der Verarbeitung bestimmen und sich gegenseitig ausschließen. Die segmentindizierte Adressierung hingegen kann parallel zu den anderen Adressierungsmethoden eingesetzt werden, d.h. sie stellt eine Option zu obigen Betriebsmodi dar.

Da in manchen Fällen Bedarf besteht, das Bildformat bzw. das Farbformat bei Ein- bzw. Ausgabe von Bilddaten zu konvertieren, muß zur Verdoppelung bzw. Halbierung der Bildgröße eine 1:2 Über- bzw. Unterabtastung durchgeführt werden können. Es reicht aus, wenn Über- bzw. Unterabtastung bei Intra-Adressierung unterstützt wird, die Kombination verschiedener Bilder bzw. der Einsatz bei Ausbreitungsprozessen ist nicht erforderlich.

Gelegentlich werden Bewegungsmodelle eingesetzt, die auf affinen bzw. allgemeineren Koordinatentransformationen beruhen. Beim punktweisen Vergleich transformierter Bildausschnitte bzw. Segmente mit einem Referenzbild werden die Bildpunkte mit transformierten Koordinaten adressiert. Bei einer Interpolation ist außerdem die Adressierung einer Nachbarschaft mit transformierten Koordinaten erforderlich. Die Parameter derartiger Modelle können jedoch auch ohne Adressierung mit Koordinatentransformation bestimmt werden [MoHe99], so daß eine derartige Adressierung, die darüber hinaus aufwendig zu implementieren ist, nicht vorgesehen werden muß.

In Videoanalysealgorithmen werden Low-Level-Operationen vom High-Level-Algorithmus aufgerufen. Da der High-Level-Algorithmus von einem Universalprozessor übernommen wird, ist es dessen Aufgabe den Hardware-Beschleuniger zu steuern. Der Prozessor bestimmt den Operationsmodus und weitere Parameter wie Bildgröße, Bildausschnitt, usw. und startet die Operation.

4.1.2 Verarbeitung

Die zu konzeptionierende Architektur muß sowohl bezüglich High-Level-Operationen als auch bezüglich Low-Level-Operationen ausreichend Flexibilität bieten, um die in Kapitel 2.4 aufgelisteten Operationen zu unterstützen. Flexibilität bezüglich High-Level-Operationen bedeutet, daß viele Möglichkeiten zur Verkettung unterschiedlicher Operationen bestehen, so daß viele Algorithmen durch die Architektur unterstützt werden können. Flexibilität bezüglich Low-Level-Operationen bedeutet die Anpaßbarkeit und Parametrisierbarkeit der Operationen, so daß ein breites Spektrum an Operationen zur Verfügung steht.

Während der Verarbeitung eines Bildes bzw. eines Segments wird wiederholt dieselbe Operation zur Verarbeitung einzelner Bildpunkte verwendet. Eine konfigurierbare Verarbeitungseinheit ist einer programmierbaren vorzuziehen, denn konfigurierbare Einheiten eignen sich gut für häufig wiederholte einfache Operationen, benötigen im Gegensatz zu programmierbaren Einheiten keine Befehlszufuhr, und besitzen gegenüber diesen einen Packungsdichtevorteil, wie in Kapitel 3.2.3 erläutert.

Konfigurierbare Einheiten besitzen zwei Arten von Flexibilität: Flexibilität durch Konfiguration der Operation von Blöcken, aus denen die konfigurierbaren Einheiten aufgebaut sind, und die Konfiguration der Verschaltung der Blöcke. Wenn die Verarbeitungseinheit - in ähnlicher Weise wie FPGAs - aus sehr kleinen Grundelementen bzw. Blöcken aufgebaut ist, handelt es sich um eine feingranular konfigurierbare Architektur. In diesem Fall ergibt sich in der Regel eine sehr hohe Flexibilität, die jedoch gleichzeitig einen hohen Aufwand für Ressourcen zur Konfiguration (Multiplexer, Verbindungen, etc.) und eine entsprechend langsamere Operation bedingt.

Daher ist es zweckmäßig, komplexere Grundelemente einzusetzen, so daß weniger Möglichkeiten zur Verschaltung bzw. Konfiguration bestehen und der Aufwand für Ressourcen zur Konfiguration geringer ist. Dabei handelt es sich um grobgranulare Konfigurierbarkeit. Im Extremfall besteht die Verarbeitungseinheit aus dedizierten Blöcken, und gestattet ausschließlich eine Umschaltung zwischen diesen. Mit Hinblick auf die Flexibilität der Verarbeitungseinheit wäre dies jedoch eine ungünstige Realisierung, da lediglich ein Satz fester Operationen unterstützt würde. Eine Nutzung von Hardware-Blöcken für mehrere Operationen wäre nicht möglich, so daß bei der Ausführung einer Operation der Großteil der Blöcke unbenutzt bliebe.

Um die Flexibilitätsanforderungen ohne unnötigen Aufwand zu erfüllen ist die Granularität der erforderlichen Flexibilität anzupassen. Außerdem soll die Architektur der Verarbeitungseinheit Ähnlichkeiten im Aufbau der unterstützten Operationen berücksichtigen und nach Möglichkeit Blöcke für mehrere Operationen gemeinsam nutzen (Ressource Sharing). Bei der Umsetzung der Bildpunktoperationen ist also darauf zu achten, aus welchen Teiloperationen diese bestehen und welche Teiloperationen Ähnlichkeiten aufweisen bzw. mehreren Operatoren gemeinsam sind. Unterschiedliche Teiloperationen werden getrennt implementiert, damit sie effizient umgesetzt werden können. Dabei muß darauf geachtet werden, daß die Teiloperationen zur Implementierung von Bildpunktoperationen möglichst weitgehend einsetzbar und kombinierbar sind. Auf diese Weise kann nicht nur die Menge der benötigten Hardware-Blöcke verringert werden, es werden auch Kombinationen von Teiloperationen ermöglicht, die über die Menge zu unterstützender Bildpunktoperationen hinausgenende zusätzliche Operationen darstellen.

Für Videoanalyse- und Videosegmentierungsverfahren, die unter Nutzung aller drei Farbkomponenten alle im Bildmaterial verfügbare Information zur Abgrenzung von Objekten bzw. zur Extraktion von Bild- und Objekteigenschaften benutzen, ist die gemeinsame Bearbeitung aller Farb- und Zusatzkomponenten in einer Architektur wichtig. Bedingte Ausführung wird durch Multiplexer im Datenpfad implementiert, so daß Operationen mit bedingter Ausführung keine zusätzliche Steuerung im Vergleich zu anderen Operationen erfordern. Aufwendig zu implementierende Operationen, die selten durchzuführen sind, wie beispielsweise die Division im Fall der Schwerpunktsberechnung, müssen nicht in der Verarbeitungseinheit implementiert werden, son-

dern können - wie auch High-Level-Operationen - von einem Universalprozessor übernommen werden.

Neben diesen speziellen Anforderungen spielen allgemeinen Anforderungen bezüglich geringer Chipfläche und kurzer Verarbeitungszeit eine Rolle. Die Anforderungen sind jedoch oftmals gegensätzlich. Für die Verarbeitungseinheit steht die Zielsetzung möglichst kurzer Verarbeitungszeit im Vordergrund, da die Hardware hohen Performance-Ansprüchen genügen muß.

4.1.3 Registermatrix

Die Registermatrix enthält einen Ausschnitt der Daten aus dem Bildspeicher, um die bei Intra- und Segmentadressierung benötigte Nachbarschaft an Eingangsdaten für die Verarbeitungseinheit bereitzustellen. Da keine Parallelisierung auf Bildpunktebene verfolgt wird (siehe Kapitel 4.2.2), gibt es genau einen Punkt (den Zentralpunkt der Nachbarschaft), für den ein Verarbeitungsergebnis berechnet wird.

Hinsichtlich der Größe der Registermatrix ist zu beachten, daß Register verhältnismäßig viel Chipfläche benötigen. Außerdem verursacht eine große Nachbarschaft eine hohe Anzahl an Verarbeitungseinheiten und an Verbindungen zu diesen. Da für viele zweidimensionale Operationen, die in Segmentierungsalgorithmen eingesetzt werden, eine CON8 Nachbarschaft ausreicht, und da weitere zweidimensionale Operationen in zwei aufeinanderfolgende eindimensionale Operationen separiert werden können, werden neun Register implementiert, die entweder eindimensional als 9x1 bzw. 1x9 Vektor oder zweidimensional als 3x3 Matrix angeordnet werden können.

In Abhängigkeit der gewählten Adressierungsart werden die passenden Daten für eine Operation in die Registermatrix geladen, in der sie für die Dauer der Verarbeitung bleiben. Nach dem Laden der Daten startet die Verarbeitung und nach deren Ende wird das Verarbeitungsergebnis im Ergebnisbild gespeichert.

Um eine überlappende Operation von Laden der Bilddaten und Verarbeitung zu ermöglichen, werden weitere Register vorgesehen, in denen während einer laufenden Verarbeitung Daten für den nächsten Schritt abgelegt werden können, ohne die laufende Verarbeitung zu beeinträchtigen (Vorladen). Nach dem Abschluß der Verarbeitung werden durch Schieben bzw. durch Aktualisieren der Daten in der Registermatrix die für die folgende Bildpunktoperation nötigen Eingangsdaten bereitgestellt.

.

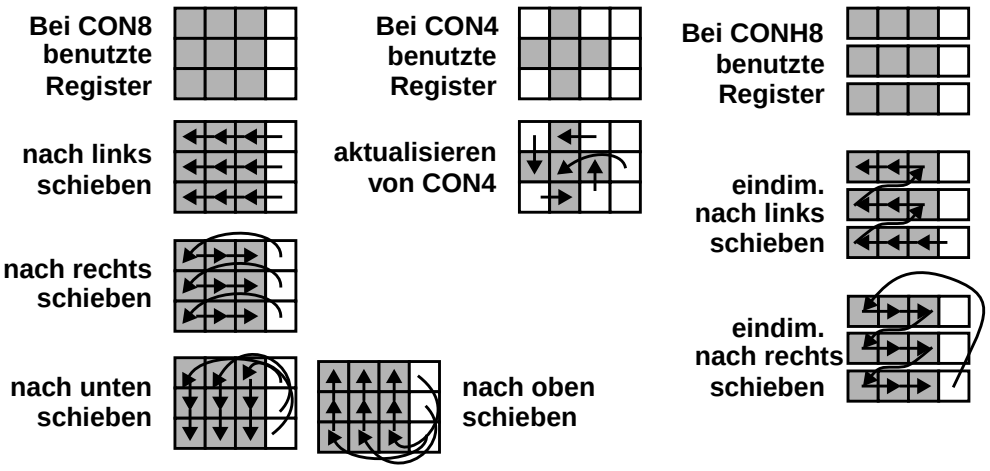


Bild 4.1: Operation der Registermatrix

Die Anzahl der zum Vorladen erforderlichen Register hängt davon ab, wieviele Bildpunkte gleichzeitig geladen werden können. In Bild 4.1 ist exemplarisch eine Registeranordnung

gezeigt, die für den Fall ausgelegt ist, daß nur ein Bildpunkt gleichzeitig geladen werden kann. Die gezeigte Registeranordnung ist für Inter- und Intra-Adressierung mit bis zu CON8 bzw. CONH8/V8 Nachbarschaft und für Segment-Adressierung mit bis zu CON4 Nachbarschaft ausreichend. Schieben ist ein- und zweidimensional in alle Richtungen möglich, darüberhinaus ist das Aktualisieren einer CON4 Nachbarschaft möglich.

4.2 Parallelisierungsstrategie

Um eine hohe Verarbeitungsgeschwindigkeit zu erreichen ist zusätzlich zu dem durch Hardware-Implementierung gegenüber Software erreichbaren Geschwindigkeitsvorteil eine effiziente Parallelisierung erforderlich. Auch im Hinblick auf eine Leistungseinsparung ist ein hohes Maß an Parallelität wichtig, da eine niedrige Taktfrequenz eine Reduzierung der Versorgungsspannung ermöglicht.

Die Parallelisierung kann in Form überlappender oder gleichzeitiger Operationsausführung auf den in Kapitel 3.2.1.4 genannten Parallelitätsebenen erreicht werden, wobei die im Algorithmus vorhandenen Datenabhängigkeiten berücksichtigt werden müssen. Da allerdings das Ziel nicht die Beschleunigung eines festen Algorithmus ist, sondern ein System entworfen werden soll, das unterschiedliche Segmentierungsalgorithmen unterstützt, muß darauf geachtet werden, daß bei der Parallelisierung die Flexibilität gewahrt bleibt.

Die Trennung von Adressierung und Verarbeitung bedeutet eine Trennung auf Ebene der Bildpunktoperationen. Durch diese Trennung können Laden, Verarbeiten und Speichern der Daten überlappend ausgeführt werden. Eine Parallelisierung, die unterhalb dieser Ebene angesiedelt ist, kann separat für Adressierungs- und Verarbeitungseinheit betrachtet werden, d.h. sie erfolgt vollständig innerhalb der jeweiligen Einheit, ohne deren äußeres Verhalten zu ändern.

4.2.1 Adressierungs- und Verarbeitungsebene

Zu den Parallelisierungsebenen, die separat für Adressierung und Verarbeitung betrachtet werden können, zählen die Ebenen der Bitoperationen, Datenwortoperationen und Komponentenoperationen. Eine dort angesiedelte Parallelisierung wird als Parallelisierung auf Verarbeitungsebene bezeichnet, wenn sie nur innerhalb der Verarbeitungseinheit stattfindet, bzw. als Parallelisierung auf Adressierungsebene, wenn sie nur innerhalb der Adreßeinheit erfolgt. Im letzteren Fall entspricht eine Parallelisierung auf Bitebene dem gleichzeitigen Laden oder Speichern eines Datenwortes, und eine Parallelisierung auf Komponentenebene dem gleichzeitigen Laden oder Speichern mehrerer Helligkeits-, Farb- bzw. sonstiger Komponenten. Eine Parallelisierung auf Ebene der Datenwortoperationen entfällt im Fall der Adressierung. Die Verarbeitungsabfolge auf Ebene der Bildpunkte und damit auch das Programmiermodell sind unabhängig von einer Parallelisierung auf Adressierungs- bzw. Verarbeitungsebene. Somit erfolgt keine Einschränkung der Flexibilität bezüglich unterstützter Algorithmen. Da die Parallelität auf dieser Ebene ein hohes Potential bietet, ist eine maximale Parallelisierung auf Adressierungs- und Verarbeitungsebene anzustreben.

Dies gilt jedoch nicht für das Pipelining der Verarbeitungseinheit. Der Nutzen, der durch Pipelining erzielt werden kann, ist nur gegeben, wenn die Verarbeitungszeit aller Stufen ähnlich groß ist. Dies ist jedoch im Falle eines weiten Anwendungsspektrums schwierig zu erreichen. Das Einfügen von Pipelineregistern schränkt außerdem die Flexibilität der Verarbeitungseinheit ein, da die Kombination von Teiloperationen erschwert wird, und es erhöht Latenz und Fläche der Verarbeitungseinheit. Gegenüber Parallelisierung hat Pipelining auch den Nachteil, daß es sich bei rekursiver Verarbeitung nicht einsetzen läßt, bei der ein Verarbeitungsergebnis wieder direkt als Eingangsdatum benutzt wird und diese Datenabhängigkeit nicht aufgebrochen werden kann.

4.2.2 Bildpunktebene

Auf der Ebene der Bildpunkte besteht bei nicht-rekursiver Verarbeitung mit regulärer Adressierung eine hohe inhärente Parallelität, da sehr viele Bildpunkte unabhängig auf die gleiche Weise verarbeitet werden. Sobald jedoch rekursive Verarbeitung eingesetzt wird, ergeben sich durch die Verwendung zuvor berechneter Ergebnisse als Eingangsdaten Abhängigkeiten zwischen den Bildpunkten, die die Möglichkeiten zur Parallelisierung einschränken. Eine punktweise bzw. blockweise Aufteilung und Zuordnung zu unabhängigen Verarbeitungseinheiten ist damit nicht mehr möglich. Bei zeilenweiser Verarbeitung ist lediglich eine versetzt parallele Operation möglich, bei der wie in Bild 4.2 gezeigt, jeder Prozessor eine unterschiedliche Zeile bearbeitet, und der horizontale Abstand zwischen verarbeiteten Bildpunkten so groß ist, daß jede Nachbarschaft eines gerade bearbeiteten Bildpunktes keinen anderen gerade bearbeiteten Bildpunkt überdeckt.

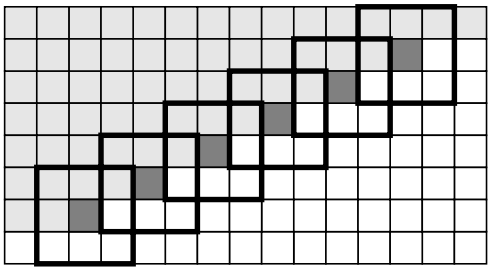


Bild 4.2: Parallelisierung auf Bildpunktebene bei rekursiver Verarbeitung mit CON8 Nachbarschaft

Bei Segment-Adressierung ist die Reihenfolge der bearbeiteten Bildpunkte nicht vorhersagbar, da die Abarbeitungsfolge von den Bilddaten abhängt. Massive Parallelisierung auf Bildpunktebene macht keinen Sinn, da nur die Bildpunkte auf einer Ausbreitungsfront parallel verarbeitet werden können. Das sind im Fall des Wasserscheide-Verfahrens beispielsweise lediglich 0,09% [Nog97] aller Bildpunkte. Bei Aufteilung des Bildbereiches in Blöcke und paralleler Verarbeitung durch verschiedene Einheiten besteht die Schwierigkeit, daß zusammenhängende, während eines Ausbreitungsprozesses bearbeitete Bereiche in unvorhersehbarer Weise in Bereiche verschiedener Blöcke fallen. Um eine Aufteilung vornehmen zu können, müssen bei Ausbreitungsprozessen die Verarbeitungsergebnisse an den Blockgrenzen abgeglichen werden, was wiederholte nochmalige Bearbeitung von Blöcken zur Folge hat. In [KlSa98] wurde eine Aufteilung für das Wasserscheide-Verfahren untersucht, wobei sich trotz geringer Aufteilung auf nur 4 Prozessoren nur eine geringe Beschleunigung von 1,92 ergab. Trotzdem ist bei Segment-Adressierung ein gewisses Maß an Parallelität möglich. Mehrere Punkte auf einer Wellenfront können parallel verarbeitet werden, z.B. dadurch, daß mehrere Verarbeitungseinheiten einen gemeinsamen Speicher für auf der Wellenfront liegende Punkte benutzen.

Auch segmentindizierte Adressierung stellt bei manchen Operationen ein Hindernis für die Parallelisierung auf Bildpunktebene dar. Wenn die segmentindizierten Daten sowohl als Eingangs- als auch als Ausgangsdaten für die Verarbeitung benutzt werden, ergibt sich eine rekursive Verarbeitung bezüglich der segmentindizierten Daten. Da die Lage der Segmente datenabhängig ist, sind die resultierenden Datenabhängigkeiten unregelmäßig und somit ungünstig für die Parallelisierung auf Bildpunktebene.

Eine Parallelisierung der Verarbeitung auf Bildpunktebene ist somit nur im Falle bestimmter Operationen möglich. Da das Ziel eine für alle Adressierungsarten einsetzbare Architektur ist, wird eine Parallelverarbeitung auf Bildpunktebene nicht verfolgt.

4.2.3 Ebene der Bild- / Segmentoperation

Auch auf Ebene der Bild- bzw. Segmentoperationen kann eine Parallelisierung erfolgen. Aus Implementierungssicht ist eine derartige Parallelisierung einfach durch den Einsatz mehrerer Prozessoren umzusetzen, aus Algorithmensicht besteht jedoch die Schwierigkeit, daß Datenabhängigkeiten zwischen vielen Bild-/Segmentoperationen bestehen.

Wenn keine Abhängigkeiten bestehen, lassen sich die Operationen parallel durchführen. Da unterschiedlichen Bild- bzw. Segmentoperationen meist unterschiedliche Zeit für die Verarbeitung benötigen, ergibt sich die Notwendigkeit zum Scheduling der Bild-/Segmentoperationen. Bei einer Serie aufeinanderfolgender Bildoperationen mit regulärer Adressierung kann Pipelining eingesetzt werden. Da innerhalb von Segmentierungsalgorithmen häufig Low-Level-Operationen in Serie aufeinander folgen, ist es sinnvoll eine Kaskadierung von Koprozessoren zu ermöglichen, die jeweils eine Bildoperation bzw. Segmentoperation unterstützen. Für eine derartige Kaskadierung muß ein Datenpfad zwischen den Koprozessoren aufgebaut werden, d.h. jeder Koprozessor muß einen Datenpfadeingang und -ausgang besitzen.

4.3 Datenorganisation

4.3.1 Bilddaten

Im Bereich der Videokommunikation spielen die in Kapitel 2.6 gezeigten Bildformate eine Rolle. Die Bildformate CIF und QCIF werden am häufigsten eingesetzt, so daß deren Unterstützung für den Einsatz bei Kommunikationsanwendungen in jedem Fall erforderlich ist. Für andere Anwendungen mit größeren Bildformaten ist die Unterstützung bis hin zum Bildformat 4CIF wünschenswert.

Da bei Segmentierungsverfahren der Einsatz unterabgetasteter Videodaten nicht zweckmäßig ist, muß lediglich das 4:4:4 Farbformat unterstützt werden. Neben dem YUV Farbraum, der bei Videokommunikationsanwendungen typischerweise verwendet wird, ist auch die Unterstützung anderer Farbräume sinnvoll. Bei Segmentierungsverfahren müssen jedoch im Gegensatz zu anderen Verfahren der Videosignalverarbeitung nicht nur die Helligkeit und Farbe, sondern auch weitere Informationen für jeden Bildpunkt gespeichert und verarbeitet werden. Diese werden als Komponenten bezeichnet und im Folgenden erläutert.

Da Segmentierungsverfahren mit Objekten bzw. Segmenten arbeiten, muß eine Segmentzuordnung der Bildpunkte unterstützt werden, d.h. es muß für jeden Bildpunkt definiert werden, ob er zu einem Segment gehört bzw. zu welchem. Dies läßt sich durch Einsatz einer Segmentierungsmaske erreichen, in der für jeden Bildpunkt die zugehörige Segmentnummer gespeichert ist. Um in Bildformaten bis zu 4CIF alle Segmente zu unterscheiden sind 65536 Segmentnummern bzw. eine Wortbreite von 16 bit ausreichend. Bei der Komposition von Szenen aus Objekten wird anstelle der Segmentnummer mit Transparenzwerten gearbeitet, die angeben, ob bzw. wie stark die jeweiligen Punkte den Hintergrund überdecken. Für den Transparenzwert ist bereits eine Wortbreite von 8 bit ausreichend. Da für Transparenzwerte die Bezeichnung Alphamaske gebräuchlich ist, wird die Komponente, welche die Segmentnummer bzw. den Transparenzwert enthält im Folgenden als Alpha-Komponente (A) bezeichnet.

Zur Darstellung von Bilddaten sind drei Komponenten für Helligkeit und Farbe erforderlich. Im YUV-Farbraum sind das die Komponenten Y, U und V, in anderen Farbräumen handelt es ich um drei andere Komponenten wie z.B. R, G und B, die sich in Y, U und V umrechnen lassen. Obwohl auch andere Farbräume unterstützt werden sollen, werden die drei Komponenten entsprechend dem typischerweise eingesetzten Farbraum als Y, U und V bezeichnet. Für die Speicherung dieser Komponenten ist eine Bitbreite von jeweils 8 bit ausreichend.

Bei Operationen wie Wasserscheide-Verfahren, Sekeletterzeugung und -rekonstruktion spielen

Distanzwerte eine wichtige Rolle. Diese müssen für jeden Bildpunkt gespeichert und verarbeitet werden, so daß hierfür eine eigene Komponente, die Auxilary-Komponente (AX), erforderlich ist. Um Distanzen darzustellen ist bei einer maximalen Bildgröße von 4CIF und in der Praxis auftretenden Segmentgrößen eine Wortbreite von 16 bit ausreichend.

Bei Ausbreitungsprozessen ist die Speicherung des Verarbeitungszustandes jedes Bildpunktes notwendig. Der Verarbeitungszustand umfaßt die drei Zustände Valid, Queued, Processed. Da die Verarbeitung außerdem davon abhängt, ob sich die Bildpunkte der Eingangs-Nachbarschaft innerhalb des Bildbereiches befinden, wird dies durch den Pseudozustand Outside signalisiert. Der Verarbeitungszustand wird im Folgenden auch als Marker-Komponente (M) bezeichnet. Tabelle 4.1 zeigt eine Übersicht über die möglichen Verarbeitungszustände, für deren Speicherung 2 bit erforderlich sind. Insgesamt sind sechs Komponenten mit zusammen 58bit pro Bildpunkt zu speichern, wie in Tabelle 4.2 gezeigt ist.

Verarbeitungs-Zustand		**Bedeutung**
0	Outside	Der Punkt liegt außerhalb des Bildbereiches und wurde durch Ergänzung von Bilddaten erzeugt (z.B. durch Kopieren des nächstliegenden Randpunktes)
1	Valid	Der Punkt liegt innerhalb des Bildbereiches und wurde noch nicht verarbeitet
2	Queued	Der Punkt liegt innerhalb des Bildbereiches und wurde als zu verarbeitender Nachbarpunkt identifiziert
3	Processed	Der Punkt liegt innerhalb des Bildbereiches und wurde bereits verarbeitet

Tabelle 4.1: Übersicht über mögliche Verarbeitungszustände

Komponente		**Verwendung**	**Wort-breite**
A	Alpha	Segmentmaske oder Transparenz	16bit
Y	Helligkeit	Helligkeit bzw. erste Komponente eines anderen Farbraumes	8bit
U	Farbe	Farbkomponente U bzw. zweite Komponente eines anderen Farbraumes	8bit
V	Farbe	Farbkomponente V bzw. zweite Komponente eines anderen Farbraumes	8bit
AX	Auxilary	Distanzwert bzw. andere mit Bildpunkten assoziierte Daten	16bit
M	Marker	Verarbeitungszustand bei Segment-Adressierung	2bit

Tabelle 4.2: Übersicht über die pro Bildpunkt gespeicherten Komponenten

Um Intra- und Segment-Adressierung zu unterstützen ist die Verwaltung eines Eingangsbildes und eines Ausgangsbildes erforderlich, für die Unterstützung von Inter-Adressierung muß ein weiteres Eingangsbild verwaltet werden, so daß insgesamt die Verwaltung von zwei Eingangsbildern und einem Ausgangsbild erforderlich ist. Da auch rekursive Verfahren unterstützt werden sollen, muß anstelle eines separaten Ausgangsbildes auch ein gemeinsames Ein- und Ausgangsbild verwaltet werden können. Um den Wechsel von Ein- und Ausgangsbild zu ermöglichen, und um die Unterstützung von Algorithmen zu ermöglichen, die mit einer größeren Anzahl an Bildern arbeiten, müssen die Bilddaten in einem großen externen Speicher abgelegt werden, und es muß die Speicheradresse der für eine Operation verwendeten Ein- und Ausgangsbilder einstellbar sein. Da sowohl Intra- als auch Inter-Adressierung nicht für die Verarbeitung ganzer Bilder, sondern auch für die Verarbeitung rechteckiger Bildausschnitte eingesetzt werden sollen, muß auch die Einstellung der Größe des verarbeiteten Bildausschnittes und dessen relative Lage in den Ein- und Ausgangsbildern möglich sein. Bei Inter-Adressierung muß auch der Fall unterstützt werden, daß zwei Ausschnitte desselben Bildes als Eingangsdaten für einen Verarbeitungsschritt verwendet werden.

Bei Inter- oder Intra Adressierung soll zusätzlich eines der Eingangsbilder über den Datenpfadeingang zugeführt und/oder das Ausgangsbild über den Datenpfadausgang weitergeleitet werden können.

4.3.2 Segmentindizierte Daten

Bei Segmentierungsverfahren werden auch Daten benutzt, die nicht einzelnen Bildpunkten zugeordnet sind, sondern ganzen Segmenten, wie z.B. Segmenteigenschaften und Histogramme. Diese Daten werden für jedes Segment getrennt gespeichert und als segmentindizierter Datensatz bezeichnet. Um die in Kapitel 2.4.5 identifizierten Operationen zu unterstützen, müssen die im Anhang in Tabelle A.1 aufgelisteten Segmenteigenschaften verwaltet werden. Zusätzlich muß für jedes Segment ein Histogramm verwaltet werden können. Um Histogramme zu verwalten, die den Wertebereich einer Helligkeits- bzw. Farbkomponente und die im Videomaterial auftretenden Segmentgrößen abdecken, müssen 256 Einträge zu je 16 bit pro Segment verwaltet werden können. Auf Segmenteigenschaften wird über die Segmentnummer als Index zugegriffen, auf Histogramme über die Segmentnummer und die Zufallsgröße. Wie aus Tabelle A.2 ersichtlich, umfassen die Segmenteigenschaften, die zur Durchführung von Low-Level-Operationen gebraucht werden, insgesamt 409 bit.

4.4 Steuerungskonzept

Der Koprozessor soll einen weiten Satz an Operationen unterstützen, die sich sowohl hinsichtlich der Adressierung unterscheiden, d.h. im Bezug auf Art und Menge der pro Operation verarbeiteten Daten, als auch hinsichtlich der Verarbeitung, d.h. im Bezug auf die Art durchzuführender Operationen. Je nach Art und Menge zugegriffener Daten ist eine unterschiedliche Anzahl von Speicherzugriffen, und damit eine unterschiedliche Anzahl von Taktperioden pro Operation nötig. Auch die Verarbeitungsdauer variiert in Abhängigkeit der durchzuführenden Operation.

Aufgrund der großen verarbeiteten Datenmenge sind Speicherzugriffe bzw. deren Geschwindigkeit kritisch für eine schnelle Operation. Daher muß die Bandbreite des verwendeten Speichers effizient genutzt werden, so daß der Systemtakt des Koprozessors zweckmäßigerweise an dem maximalen Takt der Speicherschnittstelle ausgerichtet wird bzw. als ein Vielfaches dieses Taktes gewählt wird. Andererseits ist trotz fester Taktfrequenz eine Anpassung des Ablaufes an die Verarbeitungsdauer pro Bildpunkt erforderlich.

Bei fester Taktfrequenz bedeutet eine Anpassung an die Verarbeitungsdauer, daß sich die Anzahl an Taktzyklen zur Durchführung der Verarbeitung je nach Art der durchgeführten Operation

unterscheidet. Da die Aufteilung der Verarbeitungseinheit in Pipelinestufen nur bei nicht-rekursiver Verarbeitung möglich ist, sie jedoch andererseits die Flexibilität der Verarbeitungseinheit einschränkt und deren Latenz und Fläche erhöht, werden Verarbeitungsoperationen stattdessen als sogenannte Multi-Zyklus Operationen implementiert, d.h. die Verarbeitungseinheit enthält Signalpfade, deren Laufzeit größer als die Dauer einer Taktperiode ist. Die Anpassung der Steuerung an die Verarbeitungsdauer erfolgt auf Ebene ganzer Taktzyklen. Die Verarbeitung beginnt, wenn alle Eingangsdaten in der Registermatrix bereitstehen. Die Steuereinheit muß dafür sorgen, daß alle Eingangsdaten während der Verarbeitung über mehrere Taktzyklen hinweg stabil bleiben, und daß nach Abschluß der Verarbeitung die Speicherung des Ergebnisses erfolgt. Die Anzahl der Taktzyklen bestimmt sich aus dem kritischen Pfad bei der jeweils durchgeführten bzw. konfigurierten Operation. Da der kritische Pfad durch eine statische Analyse bestimmt werden kann, ist es nicht erforderlich, die Verarbeitungsdauer durch Hardware-Einheiten zu ermitteln. Vielmehr ist es ausreichend, wenn die zur Verarbeitung erforderliche Anzahl an Taktzyklen konfiguriert werden kann.

4.5 Systemkonzept

Aus den zuvor dargestellten Anforderungen und Überlegungen ergibt sich das in Bild 4.3 gezeigte Architekturkonzept. Die konzeptionierte Architektur besteht aus Speicher für Bilddaten und segmentindizierte Daten, Adreßeinheit, Verarbeitungseinheit, Registermatrix und Koordinatenspeicher. Die dedizierte Adreßeinheit übernimmt Laden und Speichern von Daten, die konfigurierbare Verarbeitungseinheit ist für die Durchführung der Bildpunktoperationen zuständig. Beide Einheiten sind durch eine Schnittstelle für segmentindizierte Daten, eine Schnittstelle für Bilddaten, sowie eine Schnittstelle zur Signalisierung der bearbeiteten bzw. benachbarten Positionen verbunden. Für die Bereitstellung der Eingangsdaten der Verarbeitungseinheit dient die Registermatrix, die Bilddaten innerhalb der gewählten Nachbarschaft aufnimmt. Bei Segment-Adressierung wird die Position der Nachbarn, die von der Verarbeitungseinheit bestimmt wurden, im Koordinatenspeicher zwischengespeichert.

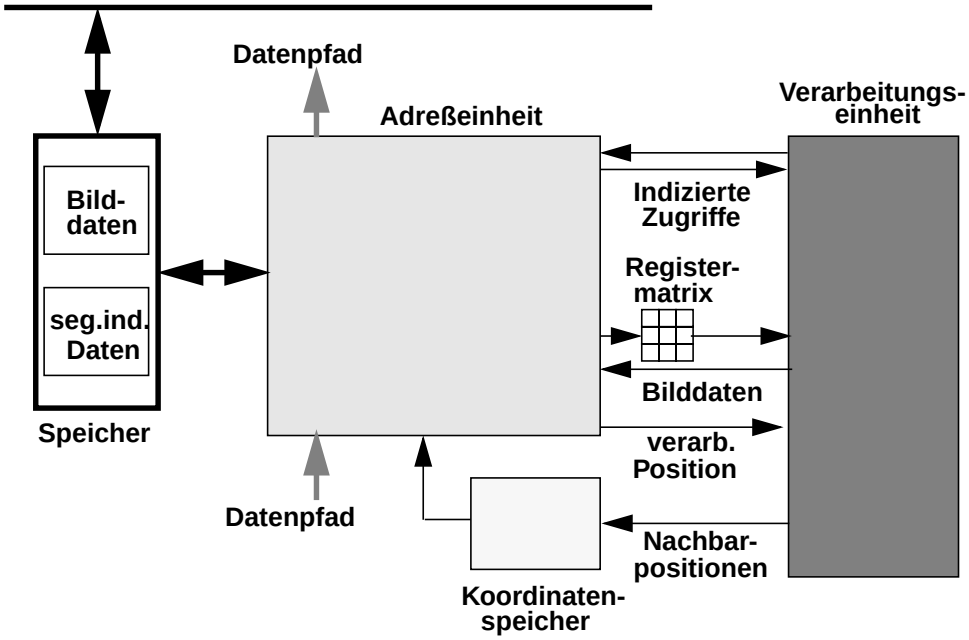


Bild 4.3: Blockbild des vorgeschlagenen Architekturkonzeptes

Werden mehrere Koprozessoren in Form einer Pipeline zusammengeschaltet, wird die Speicherschnittstelle entlastet und überlappende Ausführung mehrerer Low-Level-Operationen ermöglicht. Hierfür ist ein Datenpfadeingang und -ausgang der Adreßeinheit vorgesehen. Das vorgeschlagene Architekturkonzept wird im Folgenden als **CON**figurable **I**mage **AN**alysis Koprozessor bezeichnet, bzw. kurz als CONIAN.

Algorithmen zur Bewegungsschätzung erfordern eine sehr hohe Verarbeitungsleistung, sie enthalten jedoch zugleich eine sehr hohe Parallelität auf Bildpunktebene bei geringerer Komplexität der elementaren Operationen im Vergleich zu anderen Low-Level-Operationen. Da das Konzept der flexiblen Architektur für Videosegmentierung und -analyse keine Parallelisierung auf Bildpunktebene vorsieht, eignen sich dedizierte Architekturen besser für Bewegungsschätzung, zumal diese aufgrund der großen Bedeutung der Videocodierung bereits sehr weit entwickelt sind. Daher wird die Bestimmung der Bewegung innerhalb der Videodaten nicht von der vorgeschlagenen Architektur übernommen, sondern von einer auf Bewegungsschätzung spezialisierten Architektur, so daß das Gesamtsystem insgesamt aus drei Einheiten besteht. Bild 4.4 zeigt die Zusammenschaltung der drei Einheiten über einen gemeinsamen Bus. Diese ist zweckmäßig, da die drei Einheiten im Vergleich zu Datentransfers komplexe und lange dauernde Berechnungen übernehmen. Bei Erweiterung um weitere CONIAN Koprozessoren kann wie abgebildet eine Zusammenschaltung über den Datenpfad erfolgen.

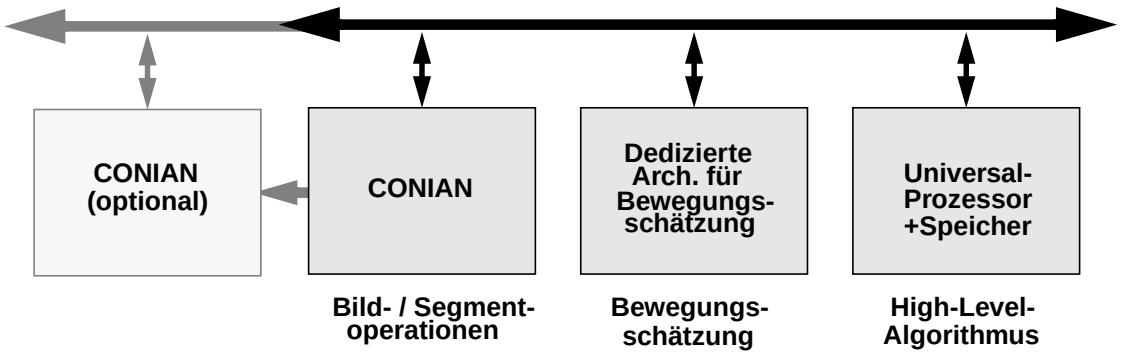


Bild 4.4: Blockbild des Systemkonzeptes

5. Entwurfsmethodik

Im vorherigen Kapitel wurde anhand der Algorithmenuntersuchungen das Grundkonzept der vorgeschlagenen Architektur und des HW/SW-Systems insgesamt vorgestellt. Um von diesem Konzept zu einer implementierbaren Architektur zu gelangen, muß die Architektur der Adreßeinheit und der Verarbeitungseinheit entworfen werden. Zentrale Fragestellungen sind dabei die Zu- und Abfuhr großer Datenmengen und die Balancierung der Ressourcen, insbesondere der Speicherschnittstelle und der Verarbeitungseinheit. Aufgrund datenabhängiger Operationen ist es unmöglich, eine effiziente Architektur ohne Simulation von Algorithmen mit realen Daten zu entwerfen.

In diesem Kapitel wird auf die Probleme bei derzeit üblichen Entwicklungsabläufen für Hardware/Software-Systeme eingegangen, und es werden Defizite weiterentwickelter Entwurfsmethoden, im Hinblick auf einen schnellen Vergleich verschiedener Architekturen zur Implementierung eines komplexen datenabhängigen Algorithmus, aufgezeigt. Ausgehend von den Anforderungen im vorliegenden Fall wurde eine neue Simulationsmethodik entwickelt, die auf dem Prinzip der Trennung von Algorithmen- und Architekturmodell beruht und in Kapitel 5.4 dargestellt ist. Dort wird erläutert, wie durch gekoppelte Simulation komplementärer Modelle eine Architektursimulation für datenabhängige Algorithmen ermöglicht wird und wie durch einfach austauschbare Architekturmodelle der Vergleich von Architekturalternativen ermöglicht wird. In Kapitel 5.5 wird schließlich auf das Algorithmenmodell und die verschiedenen Architekturmodelle für das in Kapitel 4 vorgeschlagene Architekturkonzept eingegangen.

5.1 Anforderungen beim Architekturentwurf für CONIAN

Obwohl das in Kapitel 4 vorgeschlagene Architekturkonzept bereits wichtige Architekturentscheidungen beinhaltet, ist es noch weit von einer implementierbaren Architektur entfernt. Im Rahmen des Architekturkonzeptes wurde beispielsweise keine Festlegung getroffen, welche Teile der Architektur sich auf einem Chip befinden und es wurde auch nicht betrachtet, wie Adreßeinheit, Verarbeitungseinheit, Registermatrix und Koordinatenspeicher implementiert werden können.

Zentrales Ziel beim Architekturentwurf muß die Balancierung der Auslastung der Architektureinheiten sein, da es nur Sinn macht, für diejenigen Einheiten hohen Aufwand zu investieren, die kritische Ressourcen darstellen, d.h. solche, die einen die Systemgeschwindigkeit begrenzenden Engpaß darstellen. Aufgrund der hohen verarbeiteten Datenmenge ist auch von zentralem Interesse, wie die Daten kompakt abgespeichert werden können und wie eine effiziente Datenzufuhr erfolgen kann. Bei der vorgeschlagenen Architektur sind konkret folgende Fragen von Interesse:

- Verarbeitungsgeschwindigkeit: Im Hinblick auf die Balancierung von Verarbeitung und Adressierung ist von Interesse, welche Geschwindigkeit die Verarbeitungseinheit erreichen muß.
- Ort der Datenspeicherung: Wenn Daten chipintern gespeichert werden, sind schnelle Zugriffe möglich, jedoch kann nur Speicher für eine begrenzte Datenmenge implementiert werden. Wenn Daten extern gespeichert werden, können große Datenmengen verwaltet werden, jedoch sind Datenzugriffe langsamer und es können weniger Daten parallel zugegriffen werden.
- Speichertechnologie: Dynamische Speicher können große Datenmengen kompakt speichern, Zugriffe haben jedoch höhere Latenz als bei statischen Speichern und können nur innerhalb begrenzter Bereiche wahlfrei effizient stattfinden. Außerdem ist eine periodische Auffrischung nötig.
- Gemeinsame oder getrennte Speicherung von Daten: Das Architekturkonzept sieht die Verwaltung von Bilddaten, segmentindizierten Daten und Koordinaten vor. Während eine getrennte Speicherung den Vorteil hat, daß auf die jeweiligen Daten schnell und unabhängig

zugegriffen werden kann, vermeidet eine gemeinsame Speicherung der Daten die Aufteilung der begrenzten Bandbreite über die Chipgrenze hinweg, die bei ungleichmäßiger Nutzung der Schnittstellen nachteilig ist.

- Speicherorganisation: Mit welcher Speicherorganisation können möglichst viele Zugriffe auf benachbarte Bildpunkte parallel durchgeführt werden? Bildspeicher müssen die Speicherung mehrerer Komponenten unterstützen. Ist eine komponentenweise Bildspeicherorganisation anzustreben, die die Flexibilität der Operationen erhöht, indem sie komponentenweise Kombination von Verarbeitungsergebnissen ermöglicht, oder ist eine punktweise Speicherorganisation zweckmäßig, da typischerweise ein aus Y, U, V, A und AX bestehendes Bild auf einmal verarbeitet wird?
- Datenpufferung: Die Daten von Bildpunkten werden entsprechend der Größe der Nachbarschaft mehrfach geladen. Daher ist es von Interesse, ob bzw. wie eine Pufferung der zugegriffenen Daten möglich ist, so daß mehrfache Zugriffe sowohl im Falle regulärer als auch irregulärer Abfolge von einem Pufferspeicher reduziert werden können.

Zur Umsetzung des in Kapitel 4 beschriebenen Architekturkonzeptes ist vor der Erstellung eines Hardware-Modells die Klärung offener Architekturfragen erforderlich. Neben der Flexibilität, die gewährleistet sein muß, um die erforderlichen Algorithmen zu unterstützen, ist die Verarbeitungsgeschwindigkeit von zentralem Interesse, da die Systemverarbeitungsleistung darüber entscheidet, ob die Architektur einen zu implementierenden Algorithmus unter Einhaltung von Echtzeitbedingungen umsetzen kann.

Bedingt durch hohen Rechenleistungsbedarf besitzt die Optimierung der Verarbeitungsgeschwindigkeit die höchste Priorität. Bei mobilen Anwendungen ist eine möglichst hohe Parallelität anzustreben, so daß ein geringer Leistungsverbrauch erzielt werden kann. Auch hierbei spielt die Bestimmung der Verarbeitungsgeschwindigkeit eine zentrale Rolle, um zu ermitteln, auf welche Weise die erforderliche Parallelität erreicht werden kann. Die Ermittlung der Verarbeitungsgeschwindigkeit ist auch für die Bestimmung des Leistungsverbrauches erforderlich, da für dessen Berechnung die Kenntnis der Systemtaktfrequenz Voraussetzung ist. Im Hinblick auf Implementierungskosten ist die Flächeneffizienz ein weiteres Optimierungsziel, d.h. daß einem Flächenmehraufwand eine entsprechende Steigerung der Verarbeitungsgeschwindigkeit gegenübersteht.

Die Umsetzung des Architekturkonzeptes ist sehr komplex. Bei der zu unterstützenden Segment-Adressierung ist der Ablauf des Algorithmus datenabhängig, und somit auch die Verarbeitungsgeschwindigkeit der Architektur. Zu deren Untersuchung müssen reale Eingangsdaten berücksichtigt werden, um aus dem konkreten Verarbeitungsablauf die Geschwindigkeit zu ermitteln. Um die Auswirkungen von Ressourcen- und Zugriffskonflikten zu beurteilen, die Datenorganisation zu optimieren und die Ressourcennutzung zu balancieren, muß die Auswirkung statistischer Effekte simuliert und über einen ausreichend langen Zeitraum analysiert werden. Im Folgenden werden bekannte Entwurfsmethoden im Hinblick auf die Eignung zur Umsetzung des CONIAN Architekturkonzeptes betrachtet. Da die zu implementierende Architektur entsprechend Kapitel 4 Teil eines HW/SW Systems ist, wird neben Entwurfsmethoden für HW-Architekturen auch auf Entwurfsmethoden für HW/SW-Systeme eingegangen.

5.2 Entwurf digitaler HW-Architekturen und HW/SW-Systeme

5.2.1 Typischer Entwurfsablauf

Der typische Ablauf beim Entwurf digitaler Hardware/Software-Systeme [Ern98] beginnt, wie in Bild 5.1 gezeigt, mit der Erstellung einer Spezifikation, die Funktion, Geschwindigkeit, Verlustleistung usw. festlegt. Zur Umsetzung der Spezifikation wird ein geeigneter Algorithmus gewählt und durch Simulation die Erfüllung der geforderten Funktion kontrolliert. Wenn ein Problem im Hinblick auf die Erreichung der spezifizierten Funktionalität oder im Hinblick auf zu große Kom-

plexität auf funktionaler Ebene auftritt, wird auf dieser Entwurfsebene eine Optimierung vorgenommen, d.h. der Algorithmus wird modifiziert und neu simuliert.

Nach der Festlegung des Algorithmus erfolgt dessen Partitionierung in Teile, die in Hardware implementiert werden, und solche, die in Software implementiert werden. Zentrale Kriterien für die Partitionierung sind Komplexität, Regularität und zwischen den Algorithmenteilen ausgetauschten Datenmengen sowie Anforderungen hinsichtlich Flexibilität, Geschwindigkeit und Kosten. In diesem Schritt muß auch die Architektur des Gesamtsystems entworfen werden, die Anzahl und Art eingesetzter Prozessoren, die Funktion und Anzahl eingesetzter Hardware-Module und die Art der Zusammenschaltung von Hardware und Software.

In Hardware und in Software zu implementierende Teile werden derzeit typischerweise getrennt implementiert, es gibt jedoch Entwicklungen mit dem Bestreben eines gemeinsamen Entwurfes (siehe Kapitel 5.3.2). Während Software-Teile oftmals ausgehend von dem zuvor erstellten Algorithmus implementiert bzw. optimiert werden können, ist bei den Hardware-Teilen zuvor der Entwurf eine Hardware-Architektur und eine Aufteilung in Module erforderlich, erst danach können die einzelnen Module entworfen werden. Auch auf dieser Ebene besteht Potential für Optimierungen, jedoch ist der Rahmen dafür durch den gewählten Algorithmus eingeschränkt.

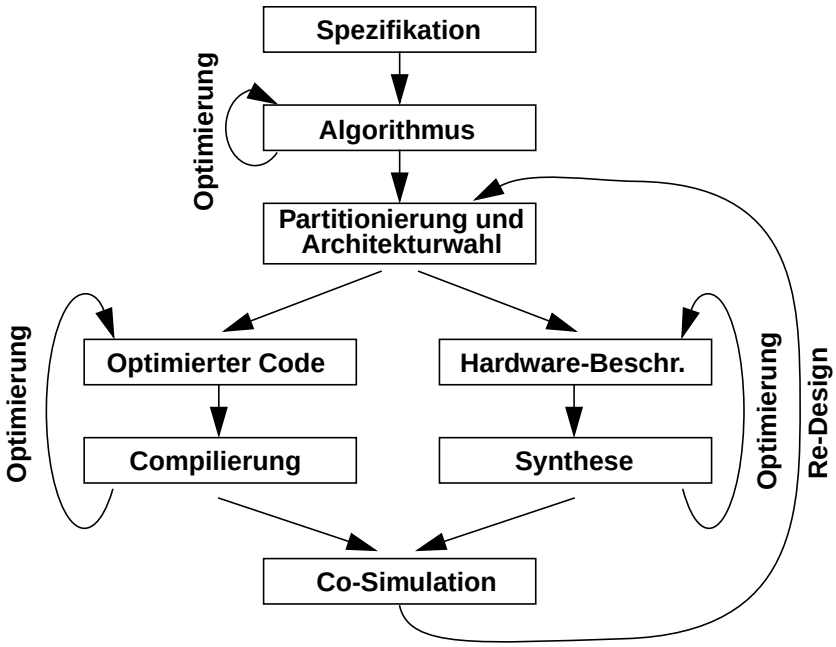


Bild 5.1: Typischer Ablauf beim Entwurf digitaler HW/SW-Systeme

5.2.2 Entwurfswerkzeuge

Aufgrund der hohen Komplexität heutiger Hardware-Architekturen und Hardware/Software-Systeme ist deren Entwurf nur mittels automatischer Entwurfswerkzeuge zu bewältigen. Für den Entwurf von Hardware/Software-Systemen gibt es einen vielfältigen Satz an Entwurfswerkzeugen, die verschiedene Schritte im Entwurfsablauf übernehmen.

Für Software-Entwicklung werden Compiler und Linker eingesetzt, mit deren Hilfe in Hochsprachen erstellte Programme in ausführbaren Code übersetzt und mehrere Module zu einem Programm zusammengebunden werden. Zum Test auf Korrektheit bzw. zur Fehlersuche werden sogenannte Debugger eingesetzt, mit denen sich der Ablauf eines Programmes verfolgen läßt, sowie weitere Testwerkzeuge, mit denen kontrolliert werden kann, daß die Speicherverwaltung fehlerfrei funktioniert (Memory Debugger) bzw. daß alle Programmteile durch Funktionstests

erfaßt wurden (Code Coverage Tools). Zur Optimierung der Laufzeit werden Profiler eingesetzt, die den Zeitverbrauch Programmteilen zuordnen können, bzw. Instruktions-Profiler, die Befehlshäufigkeiten erfassen. Werden hoch optimierte bzw. hardwarenahe Programme erstellt, werden Assembler anstelle von Compilern eingesetzt und die Simulation erfolgt mit einem Prozessorsimulator oder mit Emulations-Software und einem Testsystem, das den zu simulierenden Code ausführt.

Zu Entwurfswerkzeugen für Hardware zählen Simulationswerkzeuge, die es gestatten, mittels Hardware-Beschreibungssprachen wie VHDL und Verilog erstellte Hardware-Modelle zu simulieren. Mit Synthesewerkzeugen wird aus der Hardware-Beschreibung eine Netzliste erzeugt, die vollständig aus Elementen einer Technologiebibliothek besteht. Für spezielle Architekturen oder für Speicher werden oftmals Generatoren benutzt, die anhand von Parametern automatisch Hardware-Blöcke erzeugen. Für die Analyse des Leistungsverbrauches existieren Werkzeuge, die ausgehend von einer Netzliste und der Charakterisierung des Leistungsverbrauches der Bibliothekselemente, den Leistungsverbrauch von Hardware-Modulen simulieren. Um von der Netzliste zum ASIC zu gelangen, werden Werkzeuge für Plazierung und Verdrahtung (Place & Route), und für die Erzeugung eines Taktbaumes eingesetzt. Für den Test auf Herstellungsfehler existieren Werkzeuge zur Einfügung von Testpfaden in Hardware-Module und zum Erzeugen von Testmustern.

Für einen effizienten Entwurf sind nicht nur leistungsfähige einzelne Werkzeuge wichtig, sondern auch der Gesamtablauf und ein nahtloses Zusammenspiel der verschiedenen Werkzeuge, der sogenannte *Design-Flow* [tHa95].

5.2.3 Entwurfsproblematik

Während des Entwurfsablaufes müssen in jedem Schritt Implementierungsentscheidungen getroffen werden. Idealerweise lassen sich diese Entscheidungen automatisieren, so daß für einen Entwurfsschritt automatische Synthesewerkzeuge erstellt werden können. Bei komplexen Implementierungsentscheidungen mit vielen Freiheitsgraden ist jedoch i.d.R. keine automatische Synthese möglich, so daß die Entscheidungen manuell getroffen werden müssen. Wenn sich die Auswirkung der getroffenen Entscheidungen im selben Schritt simulieren lassen, können die Konsequenzen sofort beurteilt werden, und die am besten geeignete Alternative ausgewählt werden. Oftmals lassen sich die Konsequenzen von Entwurfsentscheidungen jedoch erst am Ende des Entwurfsablaufes beurteilen, so daß die Auswirkungen zu Beginn des Entwurfsablaufes getroffener Entscheidungen, die meist sehr weitreichende Konsequenzen haben, nur schwer abgeschätzt werden können.

Bei Software-Implementierung, kann beispielsweise die Verarbeitungsgeschwindigkeit mit Befehlssatzsimulatoren ermittelt werden, so daß sich Implementierungsentscheidungen durch Programmänderung und Neusimulation relativ leicht überprüfen lassen. Bei Hardware-Implementierung muß zunächst eine Architektur festgelegt werden, bevor die Module mit einer Hardware-Beschreibungssprache entworfen und zusammengeschaltet werden können. Vergleiche und Optimierung einzelner Blöcke lassen sich leicht durch Änderung des Hardware-Modells und Neusynthese vornehmen. Änderungen an der Architektur erfordern jedoch ein aufwendiges Re-Design vieler Module und Schnittstellen.

Werden kritische Stellen bzw. nachteilige Entwurfsentscheidungen rechtzeitig erkannt, ist die Durchführung kleiner Optimierungsschleifen möglich, die späte Erkennung ist hingegen problematisch, da sie meist hohen Aufwand für erneut durchzuführende Entwurfsschritte und Verifikation als Konsequenz hat. Besonders problematisch ist, wenn sich im Lauf des Entwurfsprozesses die Notwendigkeit zur Änderung des zugrundeliegenden Algorithmus, der Partitionierung oder der Architektur ergibt. Aufgrund des hohen Aufwandes für die Entwurfsschritte, ist es auch nicht praktikabel, verschiedene Alternativen zu entwerfen oder systematische Vergleiche anzustellen,

um zu einer optimalen Entscheidung zu gelangen. Stattdessen werden Entscheidungen oftmals aufgrund vorhandener Erfahrung bzw. aufgrund von Vermutungen getroffen.

5.3 Weiterentwickelte Entwurfstechniken

5.3.1 Syntheseverfahren

Ausgangspunkt für automatischen Hardware-Synthesewerkzeuge sind derzeit in der Regel Modelle auf Register-Transfer-Ebene (RTL), erstellt mit Hardware-Beschreibungssprachen wie VHDL oder Verilog. Die Menge der synthetisierbaren Sprachelemente ist gegenüber dem vollen Sprachumfang von Hardware-Beschreibungssprachen deutlich eingeschränkt. In RTL-Modellen werden Register und dazwischenliegende Operationen explizit beschrieben. Die Synthese der Schaltung erfolgt durch direkte Generierung der Register und durch Umsetzung der dazwischenliegenden Operationen mit Methoden der Logiksynthese.

Um den Entwurf von Hardware zu beschleunigen, wurden Methoden der High-Level- bzw. Architektursynthese entwickelt [JeDi97]. Wie bei RTL-Synthese wird von einer synthetisierbare Hardware-Beschreibung ausgegangen, die jedoch einen größeren Sprachumfang umfaßt, und sich auf einer abstrakteren Beschreibungsebene bewegt. Dabei werden lediglich durchzuführende Operationen beschrieben, nicht jedoch Register und Funktionseinheiten. Aufgabe von High-Level-Synthesewerkzeugen ist es, eine Hardware-Beschreibung auf dieser Ebene mittels Allokierung von Ressourcen, Scheduling (Festlegung des zeitlichen Ablaufes) und Assignment (Zuordnung von Operationen zu Ressourcen) in eine RTL-Beschreibung zu überführen, wie es in Bild 5.2 exemplarisch gezeigt ist. Da das Synthesewerkzeug anhand von Vorgaben des Designers selbständig die Anzahl an Registern, Funktionseinheiten und deren Bitbreiten auswählt, die Zuordnung zu Taktzyklen übernimmt und einen Kontroller generiert, ist die High-Level-Beschreibung einfacher und schneller zu erstellen, als die RTL-Beschreibung. Ein Beispiel für ein kommerzielles High-Level-Synthesewerkzeug ist der Behavioural Compiler [Kna96].

Obwohl durch spezielle Techniken die gemeinsame Nutzung von Ressourcen, gepipelinte Funktionseinheiten, Multizyklus-Funktionseinheiten, und Verkettung von Operationen innerhalb eines Taktes unterstützt werden, ist das Einsatzfeld von High-Level-Synthesewerkzeugen eingeschränkt durch den hohen Zeitbedarf der High-Level-Synthese. Aus diesem Grunde können nur Module begrenzter Komplexität automatisch synthetisiert werden. Wenn hierarchisch geschachtelte Module erzeugt, oder Daten in einem Speicher abgelegt werden sollen, sind manuelle Anweisungen an das Synthesewerkzeug notwendig, da eine automatische Evaluierung des Lösungsraumes nicht in praktikabler Zeit möglich ist.

Aus diesem Grund ist trotzt dieser Techniken ein manueller Entwurf der über der Modulebene liegenden Hardware-Architektur notwendig. High-Level-Synthese ist nützlich, wenn die Bandbreite der zu verarbeitenden Daten gering gegenüber dem Systemtakt ist, da in diesem Fall automatisch eine Steuerungseinheit erzeugt wird, bei hohem Datendurchsatz ergeben sich jedoch typischerweise stark gepipelinte Architekturen, die direkt aus der algorithmischen Beschreibung abgeleitet werden können, und keine Steuerungseinheit benötigen. High-Level-Synthesewerkzeuge sind auf den Entwurf von Hardware-Modulen ausgerichtet, die eine festen Algorithmus implementiert, nicht jedoch auf den Entwurf von Modulen, die sich flexibel für verschiedene Algorithmen einsetzen lassen.

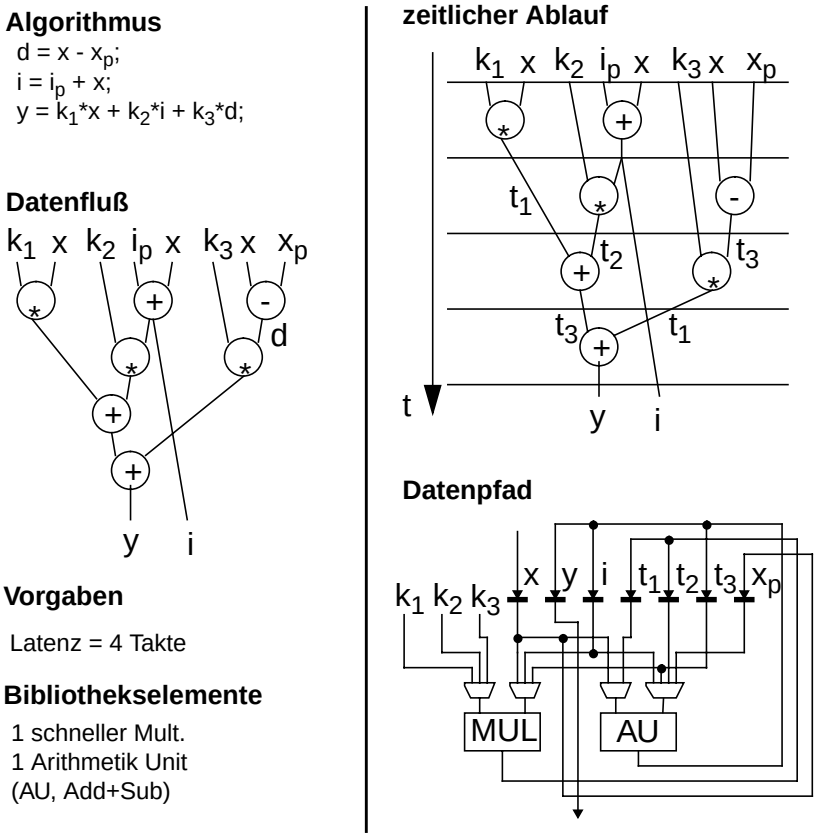


Bild 5.2: Eingangsdaten (links) und Ergebnisse (rechts) der Architektursynthese

5.3.2 Hardware/Software-Co-Design

Mit zunehmender Komplexität Integrierter Schaltungen wurde es möglich, Prozessoren und Hardware-Einheiten auf einem Chip zu integrieren. In diesem Zusammenhang ist die Bezeichnung *System on Chip* gebräuchlich, bzw. man spricht von eingebetteten Prozessoren (embedded Processors*)* oder allgemeiner von eingebetteten Systemen (embedded Systems*)*. Aufgrund der Bedeutung von Hardware/Software-Systemen wurden Methoden entwickelt, die einen gemeinsamen Entwurf von Hardware und Software unterstützen, sogenanntes *Co-Design* [Ern98]. Co-Design teilt sich in verschiedene Disziplinen auf, die sich mit unterschiedlichen Aspekten des gemeinsamen Entwurf von Hardware/Software-Systemen befassen. Insbesondere sind dabei folgende Disziplinen zu nennen:

- System-Spezifikation
- Partitionierung
- Co-Simulation
- Co-Synthese (Hardware, Software, Schnittstellen)
- Co-Verifikation

Hardware/Software-Co-Design ist ein sehr weites Feld, so daß es derzeit keine universelle Co-Design Methodik gibt, sondern applikationsspezifische Methoden, die auf eine applikationsspezifische Teilmenge an Einzelkomponenten, Zielarchitekturen und Kommunikationsmodellen beschränkt sind [dMGu97].

5.3.2.1 Hardware/Software-System-Spezifikation

Um den Entwurfsablauf von Hardware und Software zu synchronisieren, und um frühzeitig Ent-

wurfsfehler zu entdecken, ist es zweckmäßig, den Entwurfsprozeß frühzeitig zu formalisieren. Oft wird von einer ausführbaren Spezifikation ausgegangen, so daß kritische Eigenschaften während des Entwurfsablaufes durch Simulation oder formale Verifikation überprüft werden können. Hierfür gibt es eine Vielzahl an Spezifikationssprachen (z.B. SystemC, HardwareC, Statecharts, SDL, Esterel, Spec Charts, etc.) die durch unterschiedliche Sprachelemente und Beschreibungsformen die Spezifikation eines unterschiedlichen Satzes an Eigenschaften ermöglichen (z.B. Hierarchie, Nebenläufigkeit, Ausnahmebehandlung, Sequentielle Hochsprachenkonstrukte, etc.).

Derzeit erweckt die Sprache SystemC [SystemC] große Aufmerksamkeit, da SystemC auf einer vielfach für Software-Entwicklung eingesetzten Programmiersprache aufbaut, und da es von bedeutenden EDA-Firmen unterstützt wird. SystemC basiert auf C++, das durch eine Spracherweiterung in Form einer Bibliothek ergänzt wird. Es unterstützt die Definition von Ports und Modulen, die parallel operieren können, stellt Hardware-Datentypen (Bits, Vektoren, etc.) bereit und gestattet die Verwendung eines Taktes sowie die Modellierung von Synchronisation und Reaktivität. Die Sprachkonstrukte die SystemC zur Verfügung stellt, gestatten die kombinierte Modellierung von Software und Hardware, für die Analyse von Algorithmeneigenschaften im Hinblick auf Hardware-Implementierung stehen jedoch keine speziellen Sprachkonstrukte zur Verfügung. Zur Untersuchung von Architekturen bietet SystemC die Möglichkeit, ein algorithmisches Funktionsmodell in Blöcke aufzuteilen zu einem architekturspezifischen Modell (ohne bzw. mit Zeitverhalten, das bus- bzw. taktzyklusgenau beschrieben werden kann) zu verfeinern. Beim Vergleich von Architekturen verursacht jedoch eine mehrfache Aufteilung und Verfeinerung einen unpraktikabel hohen Aufwand.

5.3.2.2 Hardware/Software-Partitionierung

Methoden zur Hardware/Software-Partitionierung umfassen mehrere Aufgaben:

- Unterteilung der System-Spezifikation
- Allokation von Hardware-Ressourcen bzw. Prozessoren
- Wahl einer Architektur
- Abbildung auf Hardware bzw. Software
- Implementierung der Kommunikation und Scheduling

Während derzeit keine kommerziellen Werkzeuge hierfür existieren, gibt es im Forschungsbereich einige Partitionierungswerkzeuge, wie z.B. das COSYMA System der TU Braunschweig [ÖsBe97] bzw. das an der University of California, Berkeley entwickelte POLIS [ChEn96]. COSYMA übernimmt beispielsweise die Unterteilung der System-Spezifikation ausgehend von einer reinen Software-Implementierung. Mit einem iterativen Verfahren werden solange Hardware-Einheiten ausgelagert bis die geforderten Randbedingungen erfüllt werden. Während COSYMA die Allokation von Hardware-Einheiten selbständig durchführt, ist die Zielarchitektur auf ein System, bestehend aus einem Prozessor mit Speicher und Koprozessor, beschränkt. Eine erweiterte Version gestattet den Entwurf von Multiprozessorsystemen, jedoch ist in beiden Fällen der Einsatz auf Koprozessoren beschränkt, die lediglich kurze Abschnitte („Segmente“) des implementierten Algorithmus übernehmen.

Typischerweise wird bei Partitionierungswerkzeugen die Architektur vorab durch den Designer gewählt [GaVa94] [YeWo96], bzw. in Form von Co-Synthese Templates vorgegeben (z.B. Prozessor und Koprozessor), während die Co-Synthesewerkzeuge die verbleibenden Freiheitsgrade adressieren, wie Abbildung von Prozessen auf Komponenten, Abbildung von komplexen Variablen auf Speicher, Implementierung der Kommunikation und Scheduling [Ern98]. Eine automatische Partitionierung ohne Beschränkung des Einsatzgebietes bzw. der Freiheitsgrade beim Entwurf ist derzeit nicht möglich.

Daneben gibt es auch Werkzeuge, wie z.B. Coware N2C [N2C], die von einem bereits in Blöcke aufgeteilten System ausgehen, und die Abbildung auf Hardware bzw. Software und der Imple-

mentierung der Kommunikation unterstützen. Derartige Werkzeuge zählen jedoch im engeren Sinne nicht zu Partitionierungswerkzeugen, da keine Unterstützung bei der Blockaufteilung geboten wird. Die Wahl der Aufteilung ist jedoch entscheidend, da eine zu grobe Aufteilung die Untersuchungsmöglichkeit von Architekturen beschränkt, während eine zu feine Aufteilung einen hohen Mehraufwand bei der Simulation und beim Entwurf bedingt.

5.3.2.3 Hardware/Software-Co-Simulation

Co-Simulationswerkzeuge können in solche unterteilt werden, bei denen die Simulation auf Ebene einer einheitlichen System-Spezifikation erfolgt, und solche, die auf die gemeinsame Simulationen einer später im Entwurfsablauf vorliegenden unterteilten Beschreibung von Software und Hardware abzielen. In der Literatur ist der Begriff Co-Simulation für beides gebräuchlich, während im Sinne dieser Unterteilung strenggenommen zwischen System-Simulation und Co-Simulation unterschieden werden muß. In Abgrenzung zu Kapitel 5.3.2.1 wird hier auf Werkzeuge zur Co-Simulation eingegangen. Manche dieser Werkzeuge benutzen einen Befehlsatzsimulator zur Prozessorsimulation und ermöglichen so eine taktgenaue Simulation, während andere Prozessoren rein funktional ohne Zeitverhalten simulieren. Co-Simulationswerkzeuge setzen jedoch in beiden Fällen zu einem Zeitpunkt im Entwurfsablauf an, zu dem die Architektur bereits festgelegt und modelliert ist, so daß ein Architekturvergleich nicht mehr sinnvoll ist.

5.3.2.4 Hardware/Software-Co-Synthese

Co-Synthese umfaßt die Teilaufgaben Hardware-Synthese, Software-Synthese bzw. Compilierung und Schnittstellensynthese. In der Bereich der Hardware-Synthese fallen die zuvor genannten Methoden zur Architektursynthese. Wie auch bei Software-Synthesemethoden ist der Entwurfsraum gegenüber manuellem Entwurf stark eingeschränkt. Methoden und Optimierungstechniken für Software-Compilierung sind hingegen sehr weit fortgeschritten, jedoch bestehen Einschränkung hinsichtlich der Codeerzeugung für Signalprozessoren und Spezialprozessoren. Im Bereich der Schnittstellensynthese, der früher vernachlässigt wurde, gibt es inzwischen kommerzielle Werkzeuge, wie z.B. Coware N2C [N2C] bzw. den Synopsys Protokoll Compiler [PC].

Während der Synopsys Protokoll Compiler lediglich Schnittstellen zwischen Hardware-Blöcken erzeugen kann, ist N2C generell für Schnittstellen zwischen Hardware und/oder Software einsetzbar. Ausgangspunkt bei N2C ist eine in Blöcke aufgeteilte Systembeschreibung. Blöcke können auf Hardware oder Software abgebildet werden, die Generierung der Schnittstelle wird von N2C übernommen. Dies ermöglicht eine schnelle Exploration des Designraumes, beschränkt sich jedoch auf die Abbildung der Blöcke auf Software oder Hardware.

Co-Synthesewerkzeuge können beim derzeitigen Entwicklungsstand Architekturentscheidungen nicht bzw. nur für kleine Blöcke und eingeschränktem Anwendungsbereich automatisch treffen. In anderen Fällen, insbesondere auf höheren Ebenen, müssen Entwurfsentscheidungen weiterhin manuell getroffen werden, so daß Analysemethoden zur Bereitstellung von Entscheidungsgrundlagen bzw. zur Kontrolle der Entwurfsentscheidungen unverzichtbar sind.

5.3.2.5 Hardware/Software-Co-Verifikation

Aufgabe der Co-Verifikation ist die Prüfung ob ein entworfenes Hardware/Software-System korrekt funktionsfähig ist, bzw. ob in der Spezifikation geforderte Eigenschaften erfüllt werden. Bei der Entscheidungsfindung im Entwurfsprozeß spielen Co-Verifikationsmethoden jedoch keine Rolle, so daß sie hier nicht weiter betrachtet werden.

5.3.3 Komponentenbasierte Entwurfsmethoden

5.3.3.1 Bibliotheksbasierte Entwurfsmethoden

Bibliotheksbasierte Entwurfswerkzeuge unterstützen den Entwurf von Systemen, die sich aus

einer Menge bekannter Module zusammensetzen lassen. Der Nutzen kommt dadurch zustande, daß für die Bibliothekselemente Modelle bereitstehen, die auf einfache Weise zu einem Systemmodell verschaltet werden können. Bibliotheksbasierte Entwurfswerkzeuge bieten typischerweise die Möglichkeit zur Simulation des erstellten Modells und zur automatischen Erzeugung von Programmen bzw. Hardware-Modellen.

Da der Nutzen bibliotheksbasierter Entwurfswerkzeuge nur gegeben ist, wenn der überwiegende Teil eines Systems aus den vorhandenen Bibliothekselementen zusammengesetzt werden kann, sind bibliotheksbasierte Entwurfswerkzeuge hinsichtlich ihres Einsatzbereiches eingeschränkt. Anwendungsfelder, die große wirtschaftliche Bedeutung haben, treiben die Entwicklung voran, insbesondere im Bereich der Signalverarbeitung haben bibliotheksbasierte Entwurfswerkzeuge, wie SPW/Envision der Firma Cadence [SPW], [Envision] und Cossap der Firma Synopsys [Cossap], Bedeutung erlangt.

Beide Werkzeuge basieren auf einem graphischen Entwurfssystem, mit dem in einer Bibliothek verfügbare Standardkomponenten der Nachrichtentechnik und Videosignalverarbeitung zum Aufbau von Hardware/Software-Systemen verwendet werden können. Sie gestatten Simulation und Visualisierung der Ergebnisse, und unterscheiden sich in der Simulationsart (zyklusbasiert in Falle von SPW bzw. strombasiert im Falle von Cossap). Für DSPs besteht die Möglichkeit der automatischen Programmerzeugung. Für eine Teilmenge elementarer Komponenten (Addition, Multiplikation, u.ä.) sind Hardware-Module verfügbar, während komplexe Komponenten aus elementaren Komponenten aufgebaut werden müssen.

Der Hauptnachteil bibliotheksbasierter Entwurfswerkzeuge besteht in der Einschränkung auf vorhandene Bibliothekselemente. Wenn ein benutzerdefiniertes Modul eingesetzt werden soll, muß hierfür ein neues Bibliothekselement erstellt werden, ohne daß eine spezielle Unterstützung für den Architekturentwurf bereitsteht. Bibliotheksbasierte Entwurfswerkzeuge schränken darüberhinaus den Freiraum bei der Algorithmenentwicklung ein, da die Funktionalität innerhalb der Bibliothekselemente vorgegeben ist, und der Algorithmus nur durch Änderung der Zusammenschaltung optimiert werden kann. Die Änderung der Moduleinteilung, z.B. um eine andere Systemarchitektur zu untersuchen, oder um mehrere Funktionen auf ein architekturelle Module abzubilden, ist bei bibliotheksbasierten Entwurfswerkzeugen sehr aufwendig. Durch die Einschränkung auf Standardkomponenten und die Abbildung auf dedizierte Architekturen sind diese Werkzeuge auch nicht für den Entwurf flexibler Hardware Architekturen geeignet.

5.3.3.2 IP-basierte Entwurfsmethoden

Aufgrund der kontinuierlich steigenden Komplexität von Hardware/Software-Architekturen findet die Wiederverwendbarkeit bereits entworfener Module zunehmend Beachtung. Das Konzept des IP-basierten Entwurfes (= Intellectual Property bzw. geistiges Eigentum) besteht darin, daß Geschäftsmodelle entwickelt werden, bei denen IP-Anbieter lediglich Teile eines komplexen Systems entwerfen, und diese an Systemhersteller verkaufen, die fremde und eigene Module zu einer Architektur integrieren. Da der Hersteller eines Moduls bei IP basiertem Entwurf nicht identisch mit dem Verwerter des Moduls sein muß, haben Fragen, wie z.B. der Schutz geistigen Eigentums, Cosimulation verschiedener Beschreibungsformen und Standardisierung von Modulschnittstellen, große Bedeutung. Neben der VSI-Allianz [VSI] zielt die Standardisierung des Open Models Interface durch den IEEE [OMI] auf die Lösung derartiger Fragen.

Zur Zeit entstehen Entwurfswerkzeuge die auf die Anforderungen des IP-basierten Entwurfes ausgerichtet sind. Wie bei bibliotheksbasierten Entwurfswerkzeugen wird auf vorhandene Module zurückgegriffen, wobei es sich jedoch typischerweise um sehr komplexe Modelle handelt. Ein Werkzeug für den IP-basierten Systementwurf ist Cadence VCC. Es gestattet die Simulation aus Software- und Hardware-Modulen bestehender Systeme. Die Systembeschreibung erfolgt durch Verhaltensmodelle, die mit unterschiedlichen Beschreibungsformen wie C/C++,

SDL, FSMs, SPW dargestellt werden können und durch Architekturmodellen für Software (Mikroprozessor/DSP und Echtzeit-Betriebssystem), Hardware (z.B. OMI) und verbindenden Elemente (Busse und Speicher). Das aus der Felix Initiative hervorgegangene Werkzeug hat zum Ziel das Verhalten eines Gesamtsystems zu simulieren und eine Bewertung der Eignung verfügbarer IP-Module durchzuführen bzw. deren Auswahl zu treffen. Während für die Untersuchung der Implementierung von Softwareblöcken eine Untersuchung der Implementierbarkeit auf unterschiedlichen Prozessoren mittels Annotierung der Ausführungszeit vorgesehen ist, werden keine entsprechenden Möglichkeiten für Hardware-Module bereitgestellt.

Da Entwurfswerkzeuge für IP-basierten Entwurf nicht auf die Erstellung von IP-Blöcken zielen, sondern vielmehr auf die Integration und Simulation von IP-Modulen innerhalb eines Systems, bieten sie keine Unterstützung für die Erstellung bzw. die Optimierung von IP-Modulen, die anderen Werkzeugen überlassen wird.

5.3.4 Simulationsverfahren

5.3.4.1 Beschleunigte Hardware-Simulation

Die am meisten verbreitete Methode, Hardware-Modelle zu simulieren, ist die ereignisgesteuerte Simulation. Hierbei werden alle auftretenden Signaländerungen, sogenannte Ereignisse, zeitlich sortiert in eine Warteschlange eingereiht. Der Simulator holt sich das aktuellste Ereignis aus der Warteschlange und startet einen Auswertungszyklus [Ant96]. Bild 5.3 zeigt diesen Ablauf.

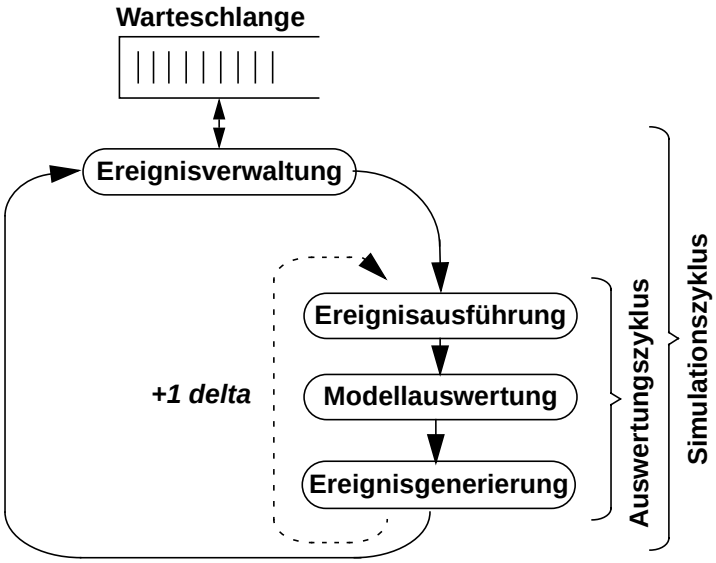


Bild 5.3: Ablauf der ereignisgesteuerten Simulation

Bei der Ereignisausführung werden die Signalwerte im Modell entsprechend dem Ereignis geändert. Die Modellauswertung ermittelt, welche Elemente von diesen Änderungen betroffen sind, was wiederum die Änderungen von Signalen nach sich zieht. Dadurch ergeben sich neue Ereignisse, die im Schritt Ereignisgenerierung erstellt und der Ereignisverwaltung übergeben werden. Treten unverzögerte Ereignisse auf, muß zum gleichen Simulationszeitpunkt ein neuer Auswertungszyklus stattfinden. In der ereignisgesteuerten Simulation wird dafür ein Ereignis in der Warteschlange abgelegt und sofort wieder herausgeholt. Eine Variante der ereignisgesteuerten Simulation durchläuft die Auswertungszyklen („delta-Zyklen“) so oft, bis keine Signaländerungen mehr stattfinden. Dann erst wird von der Ereignisverwaltung das nächste Ereignis aus der Warteschlange geholt. Ein Simulationszeitpunkt kann somit aus vielen unendlich kleinen delta-Schritten bestehen.

Bedingt dadurch, daß bei ereignisgesteuerter Simulation von Hardware-Modellen Funktionalität und Zeitverhalten mit einer hohen zeitlichen Auflösung simuliert werden, ist ein hoher Rechenaufwand erforderlich. Durch Vergröberung der Auflösung wird bei zyklusbasierter Simulation Rechenzeit gespart und die Simulationsgeschwindigkeit erhöht. Dazu werden die Signalwerte nicht bei jeder Änderung, sondern nur bei jeder aktiven Taktflanke berechnet, wie in Bild 5.4 dargestellt. Gegenüber der ereignisgesteuerten Simulation kann die Simulationszeit auf 1/2 bis 1/10 verkürzt werden. Die Beschleunigung hängt davon ab, wie detailliert das Hardware-Modell abgefaßt ist. Ein Modell auf Gatterebene erfährt eine größere Beschleunigung, als ein Modell auf Register-Transfer-Ebene.

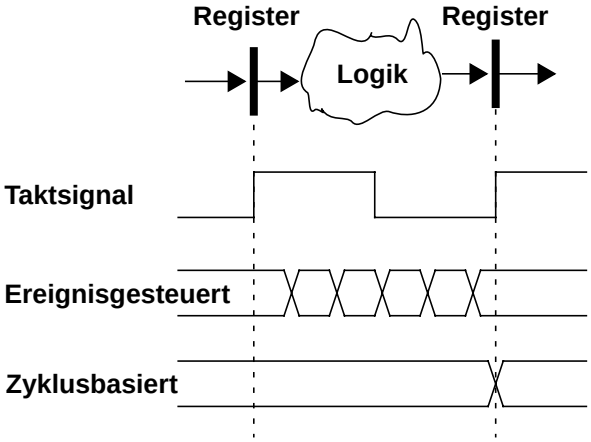


Bild 5.4: Prinzip der zyklusbasierten Simulation

Trotzt der Geschwindigkeitssteigerung ist die Simulationsdauer immer noch zu hoch: In [Jan99] wurde abgeschätzt, daß alleine die Simulation der Steuereinheit für 10 QCIF Bilder auf einer SUN ULTRASPARC-2 Workstation mindestens 6 Stunden dauert. Der entscheidende Nachteil ist jedoch, daß von denselben Hardware-Modellen wie bei der ereignisgesteuerten Simulation ausgegangen wird, und daß deren Erstellung für einen Vergleich von Architekturalternativen einen bei weitem zu hohen Aufwand erfordert. Zusätzlich zur Modellierung der Komponenten des Koprozessors wäre es außerdem notwendig einen Stimulus für das Hardware-Modell zu erzeugen, um eine von den Eingangsdaten abhängige Simulation durchzuführen. Dazu müßte eine Datenverbindung zwischen dem Hardware-Modell und dem Programm des High-Level-Algorithmus geschaffen werden, was ebenfalls sehr aufwendig ist. Um verschiedene Architekturvarianten durch Simulation zu vergleichen, ist die Simulation von RTL-Hardware-Modellen somit ungeeignet.

5.3.4.2 Simulation auf Systemebene

Um komplexe Systeme mit hoher Geschwindigkeit zu simulieren, und um den Aufwand für die Erstellung von Simulationsmodellen zu verringern, kann eine Simulation auf einer höheren Abstraktionsebene erfolgen. Es gibt viele Entwurfswerkzeuge, die zwar auf einer hohen Abstraktionsebene arbeiten, jedoch lediglich eine Simulation der Funktion durchführen. Zu diesen Werkzeugen zählen beispielsweise Aphelion [Aphelion], Micromorph [umorph] und Khoros [Khoros]. Bei letzterem handelt es sich um ein graphisches Werkzeug für den Entwurf von Bild- bzw. Videoverarbeitungsalgorithmen, das die Verschaltung von Funktionsblöcken mittels graphischer Oberfläche gestattet. Allerdings eignen sich derartige Werkzeuge, nicht für Architekturuntersuchungen.

Um komplexe Systeme bzw. Architekturen simulieren und vergleichen zu können ist es jedoch nicht nötig, die Funktion bzw. die durchgeführten Operationen Schritt für Schritt zu simulieren,

vielmehr ist es entscheidend zu simulieren, welche Vorgänge innerhalb einer Architektur stattfinden, wie häufig sie geschehen und wie lange sie dauern.

Das Werkzeug eArchitect der Firma Innoveda [Yoh00], gestattet die Erstellung reiner Architekturmodelle auf Systemebene, mit dem Ziel schneller Simulation und Beurteilung von Architekturalternativen. Modelle werden in Form von Blockdiagrammen erstellt, dessen Elemente Durchsatz und Latenz von Hardware-Einheiten beschreiben. Darüberhinaus stehen Prozessormodelle zur Verfügung, die verfügbare Befehle und deren Ausführungsdauer beschreiben. Softwaremodelle beschreiben die zur Durchführung einer Aufgabe erforderlichen Befehle, und werden Prozessormodellen zugeordnet. Neben mitgelieferten Modellen lassen sich eigene Modelle als Datenflußgraph oder in einer proprietären Sprache beschreiben. Innerhalb benutzerdefinierter Modelle können auch Teile der Funktion modelliert werden.

Ein anderes Werkzeug, das sich (neben der Erstellung funktionaler Modelle) für reine Architektursimulation einsetzen läßt, ist das an der University of California, Berkeley entwickelte Ptolemy [Ptolemy]. Es ist für die Simulation heterogener Systeme (mechanisch/elektronisch, analog/digital, HW/SW) konzipiert, so daß eine Modellierung von Systemen zusammen mit der Umgebung möglich ist. Für die Systembeschreibung können unterschiedliche Typen von Modellen eingesetzt werden, wie z.B. Differentialgleichungen, Differenzengleichungen, endliche Zustandsautomaten, synchrone reaktive Modelle, diskrete Event-Modelle.

Obwohl reine Architekturmodelle zum Architekturvergleich zweckmäßig sind, da sie unter Verzicht auf gemeinsame Modellierung von Architektur und Funktion eine schnelle Simulation gestatten, unter Verzicht auf detaillierte Modellierung gute Abstraktionsmöglichkeiten bieten und geringeren Modellierungsaufwand erfordern, ist der Verzicht auf eine funktionale Simulation nicht in jedem Fall möglich. Ohne Simulation der Funktion lassen sich beispielsweise Architekturen, die datenabhängige Algorithmen implementieren, nicht simulieren, da Abläufe in derartigen Architekturen von Vorgängen im Algorithmus abhängen. Zur Simulation von Ressourcen- und Zugriffskonflikten, schwankenden Füllständen von Puffern und anderen statistischen Effekten für reale Szenarios müssen zumindest die für den Ablauf relevanten Teile der Funktion nachgebildet werden. Dies ist im Rahmen benutzerdefinierter Modelle mit den genannten Werkzeugen möglich, da jedoch eine Strukturierung der Modelle entsprechend der untersuchten Architektur nötig ist, ergibt sich ein sehr hoher Aufwand für den Architekturvergleich. Noch schlechter eignen sich Modelle, die die komplette Funktion und die Architektur gemeinsam beschreiben. Da bei den genannten Werkzeugen keine speziellen Sprachelemente zur Verfügung stehen, um Architekturmodelle einem Funktionsmodell zuzuordnen, muß das Funktionsmodell entsprechend der zu untersuchenden Architektur strukturiert werden. Für einen Architekturvergleich ergibt sich auch hier die Notwendigkeit zu mehrfacher Umstrukturierung, die aufgrund der Gefahr von Fehlern und dem hohen Aufwand für die Einführung von Schnittstellen nicht praktikabel ist.

5.4 Neue Simulationsmethodik

Nachdem sich gezeigt hat, daß sich bekannte Entwurfsmethoden nicht für einen effizienten Entwurf der CONIAN-Architektur einsetzen lassen, ist es das Ziel, eine hierfür geeignete Methodik zu entwickeln. Da beim Entwurf komplexer Systeme bzw. Architekturen sehr viele qualitative Entscheidungen zu treffen sind, ist es beim derzeitigen Forschungsstand nicht möglich, zentrale Architekturentscheidungen zu automatisieren. Um aufwendige Re-Designs zu vermeiden, ist es erforderlich, manuell getroffene Architekturentscheidungen möglichst früh zu verifizieren, ohne daß ein vollständiges bzw. detailliertes Modell der Architektur existiert. Für diese Verifikation ist bereits eine Simulationsmethodik von großem Nutzen, wenn sie Analyse, Vergleich und Optimierung verschiedener Architekturen mit grober Modellierung gestattet. Zur Unterstützung des

Entwurfes der CONIAN-Architektur, muß eine derartige Simulationsmethodik außerdem für Architekturen bzw. Hardware/Software-Systeme einsetzbar sein, die komplexe Algorithmen mit datenabhängiger Operation implementieren.

Ziel ist somit eine Simulationsmethodik, die den Vergleich verschiedener in ein Hardware/Software-System eingebetteter Hardware-Architekturen gestattet und die Untersuchung von Architekturoptimierungen und deren Auswirkung auf die Verarbeitungsleistung ermöglicht, um eine zielgerichtete Architekturentwicklung zu gestatten. Das Ergebnis ist die Architekturentscheidung, nicht die automatische Erzeugung eines synthesefähigen Modells. Zentrale Anforderungen an die Simulationsmethodik sind:

- Einsetzbarkeit zum schnellen Vergleich von Architekturen
- Einsetzbarkeit für Hardware/Software-Systeme
- Möglichkeit zur Simulation von Algorithmen- und Architekturänderungen
- Möglichkeit zur Simulation auf Systemebene
- Keine Notwendigkeit detaillierter Architekturmodellierung
- Möglichkeit zur Verfeinerung von Architekturmodellen
- Möglichkeit zur Simulation datenabhängiger Abläufe
- Möglichkeit zur Simulation von Ressourcennutzung, Ressourcenkonflikten, Speicherzugriffen, Zugriffskonflikten und sonstiger Laufzeiteffekte
- Möglichkeit der Simulation mit realen Eingangsdaten
- Hohe Simulationsgeschwindigkeit

5.4.1 Architekturalternativen

Eine Hardware-Architektur bzw. ein Hardware/Software-System besteht aus Verarbeitungseinheiten, speichernden Elementen, Verbindungsressourcen und einer bzw. mehreren Steuereinheiten. Beim Entwurf einer Architektur werden Art und Anzahl an Verarbeitungseinheiten und speichernden Elementen festgelegt. Die Implementierung eines Algorithmus kann in abstrakter Weise als Abbildung der Operationen und Daten auf Funktionseinheiten und speichernde Elemente gesehen werden, aus deren Gestalt sich die Art der Verbindungen und der Ablauf ergibt. Aus dem Ablauf ergibt sich die Einheit zur Steuerung der Operationsausführung, der Datenzugriffe und der geschalteten Verbindungen.

Bei der Implementierung von Algorithmen bestehende Freiheitsgrade bzw. Architekturalternativen, die sich in ihren Eigenschaften wie Ressourcenbedarf, Verarbeitungsgeschwindigkeit und Leistungsverbrauch unterscheiden, können zur Optimierung eingesetzt werden, so daß entsprechend den gegebenen Randbedingungen eine Architektur mit optimalen Eigenschaften gewählt wird.

In Abhängigkeit der gewählten Art und Anzahl an Verarbeitungseinheiten ändert sich die Reihenfolge der ausgeführten Operationen, die Zuordnung zu Verarbeitungseinheiten und die Datenzufuhr, nicht jedoch die Anzahl und Art der durchgeführten Operationen. Ebenso ändert sich die Lage und Zugreifbarkeit verarbeiteter Daten in Abhängigkeit der gewählten Datenorganisation, nicht jedoch deren Größe oder Wert. Außerdem ist möglich, daß ein Datum bei manchen Datenorganisationsformen mehrfach gespeichert ist, da es z.B. in einem Pufferspeicher lokal abgelegt ist. Um vom Algorithmus zu einer Architektur zu gelangen, ist es notwendig in Abwägung zwischen Kosten und Nutzen die Operationsausführung und Datenorganisation möglichst vorteilhaft zu wählen.

5.4.2 Funktionale und zeitliche Aspekte der Simulation

Die Simulation von Algorithmen und Architekturen unterscheidet sich sowohl hinsichtlich der simulierten Aspekte, als auch hinsichtlich der Art der Simulation. Bei Algorithmen steht die Simulation der Funktion im Vordergrund, so daß das Ergebnis der durchgeführten Berechnungen

bestimmt werden muß, Zeitpunkt und Abfolge der Berechnungen jedoch außer Acht gelassen werden können. Da Zeitpunkt und Abfolge nicht simuliert werden müssen, können sie zur Erzielung einer hohen Simulationsgeschwindigkeit auch verändert werden. Bei der Simulation von Architekturen ist hingegen der Verarbeitungsleistung ein zentraler Aspekt. Die Operationsausführung ist nicht mehr sequentiell, wie bei einer Funktionalen Simulation auf einem Prozessor, vielmehr ergeben sich Nebenläufigkeiten und Zugriffskonflikte entsprechend dem gewählten zeitlichen Ablauf (Schedule), so daß hier die Simulation der Zeitpunkte und Abfolge der Operationen von zentraler Bedeutung ist.

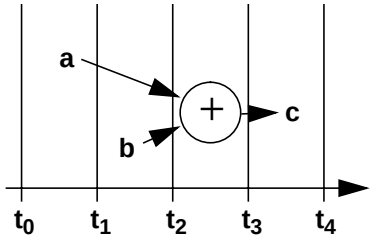


Bild 5.5: Funktionaler und zeitlicher Aspekt von Operationen

In Bild 5.5 ist der funktionale und der zeitliche Aspekt einer Simulation am Beispiel der Addition zweier Zahlen gezeigt: Die Berechnung des Operationsergebnisses c entspricht der durch Gleichung (5.1) beschriebenen Addition, die Berechnung des Zeitpunktes, an denen das Operationsergebnis zur Verfügung steht, ergibt sich aus dem Zeitpunkt zu dem beide Eingangswerte bereitstehen plus der Dauer der Addition. Dies wird durch Gleichung (5.2) beschrieben, wobei t_x den Zeitpunkt repräsentiert, zu dem das Datum x bereitsteht.

$$c = a + b \tag{5.1}$$

$$t_c = \max(t_a,t_b) + t_{add} \tag{5.2}$$

Wie aus obigen Gleichungen ersichtlich ist, kann die Berechnung des Operationszeitpunktes unabhängig von der Berechnung des Operationsergebnisses durchgeführt werden. Die Unabhängigkeit ist nicht nur bei Additionen sondern grundsätzlich bei elementaren Operationen und auch bei vielen komplexeren Operationen gegeben.

5.4.3 Simulation architektureller Aspekte

Neben dem funktionalen und zeitlichen Aspekt spielt auch der architekturelle Aspekt bei der Simulation der Verarbeitungsgeschwindigkeit eine wichtige Rolle. In Bild 5.6 ist der architekturelle Aspekt am Beispiel zweier Additionen gezeigt, für die nur ein Addierer zur Verfügung steht. Die Berechnung des Zeitpunktes, zu dem das Operationsergebnis c zur Verfügung steht, ergibt sich durch Gleichung (5.3) unter der Voraussetzung, daß diese Berechnung Priorität gegenüber der zweiten Addition hat. Bei der Berechnung des Zeitpunktes, zu dem das Operationsergebnis f gültig ist, muß wie in Gleichung (5.4) gezeigt, eine zusätzliche Wartezeit für einen eventuellen Ressourcenkonflikt berücksichtigt werden.

$$t_c = \max(t_a,t_b) + t_{add} \tag{5.3}$$

$$t_f = \max(t_d,t_e) + t_{add} + t_{wait} \tag{5.4}$$

$$t_{wait} = \begin{cases} t_{add} \text{ falls } \max(t_a, t_b) = \max(t_d, t_e) \\ 0 \qquad\qquad\qquad\qquad sonst \end{cases} \tag{5.5}$$

Ein anderes Beispiel für einen architekturellen Vorgang ist z.B. die Operation eines Caches.

Während seine Funktion für den Algorithmus transparent ist, geschehen abhängig von der Abfolge der Datenzugriffe unterschiedliche Vorgänge wie z.B. ein einfacher Zugriff bzw. Nachladen von Cache-Blöcken. Weitere Beispiele für architekturelle Vorgänge sind - gegenüber dem Algorithmus - zusätzliche Operationen zur Berechnung von Speicheradressen für eine gewählte Speicherorganisation bzw. Operationen zur Bestimmung, welche Funktionseinheit für eine Operation selektiert wird. Wenn sich diese Vorgänge z.B. durch Verzögerungen auf den Ablauf auswirken, ist ihre Berücksichtigung im Architekturmodell erforderlich, wenn nicht können sie zugunsten einer schnellen Simulation außer Acht gelassen werden.

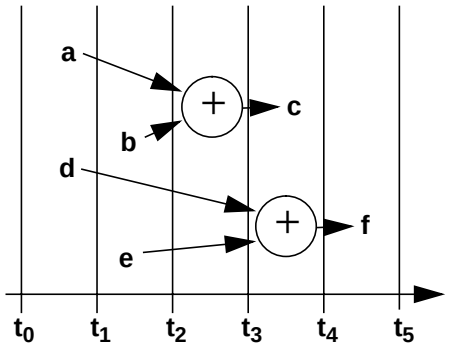


Bild 5.6: Architektureller Aspekt von Operationen

5.4.4 Trennung der Simulationsmodelle

Insgesamt können somit drei Modelltypen unterschieden werden: Das Algorithmenmodell, das Architekturmodell und das Timingmodell. Das Algorithmenmodell umfaßt Berechnungen, deren Durchführung die Aufgabe des Algorithmus ist, das Architekturmodell beschreibt Vorgänge, die in der Architektur nicht jedoch im Algorithmus stattfinden, und das Timingmodell beschreibt die Dauer von Operationen, Zugriffen und sonstigen Abläufen in Abhängigkeit der Art der Operation und vorangegangene Operationen. Die drei Modelle sind komplementär, d.h. sie ergänzen sich zu einem simulierbaren Modell, überdecken sich jedoch nicht bezüglich der modellierten Eigenschaften.

Eine Trennung zwischen Architekturmodell und Timingmodell hat zum Vorteil, daß das Zeitverhalten als Metrik betrachtet werden kann. Die Aufteilbarkeit zwischen Architekturmodell und Timingmodell ist jedoch nur möglich, wenn parallel operierende Module nicht auf den Verarbeitungszustand der jeweils anderen Module zurückgreifen und die Operationsreihenfolge der Architektur sich nicht in Abhängigkeit der zeitlichen Parameter ändert. Da diese Unterteilung keine Voraussetzung für die Simulation einer Architektur ist und sie nicht in jedem Fall möglich ist, wird im Folgenden nur zwischen Algorithmenmodell und Architekturmodell unterschieden, wobei letzteres sowohl architekturelle als auch zeitliche Aspekte beschreibt.

Zur Simulation der Verarbeitungsleistung von Architekturen ist die Simulation architektureller und zeitlicher Vorgänge bereits ausreichend, die Berechnung von Operationsergebnissen ist nicht erforderlich. Sie ist sogar nachteilig, da sie unnötigen Aufwand bedeutet, wenn die Funktion bereits in einer algorithmischen Simulation verifiziert wurde. Aus diesem Grund ist die Trennung von Algorithmenmodell und Architekturmodell sinnvoll. Die Aufteilbarkeit beider Modelle ist in jedem Fall gegeben, da eine Architektur definitionsgemäß den zugrundeliegenden Algorithmus implementiert. Durch Verzicht auf eine Modellierung funktionaler Aspekte ist die Simulation schneller und außerdem ist für die Erstellung eines solchen Modells weniger Aufwand nötig.

Der Verzicht auf funktionale Modellierung ist jedoch auf Algorithmen beschränkt, deren Operation unabhängig von den verarbeiteten Daten immer in der gleichen Weise geschieht. Bei Algo-

rithmen mit datenabhängigem Ablauf fehlt in einem Modell, das funktionale Aspekte nicht beschreibt, die Information über Art, Anzahl und Abfolge an Operationen und Datenzugriffen, die von der Architektur durchzuführen sind. Da diese durch den verwendeten Algorithmus bestimmt sind, müssen entweder Teile des Algorithmus innerhalb des Architekturmodells nachgebildet werden oder es werden Simulationsmodelle erstellt, die sowohl die Funktion als auch das architekturelle und zeitliche Verhalten der Architektur beschreiben.

In beiden Fällen muß der Algorithmus bzw. Teile davon wiederholt implementiert bzw. angepaßt werden. Dies bedeutet zusätzlichen Aufwand, der für jede zu simulierende Architektur erforderlich ist, insbesondere wenn der Algorithmus sehr komplex und das Architekturmodell vergleichsweise einfach ist. Um eine korrekte Simulation sicherzustellen, ist es außerdem notwendig mehrfach vorhandene funktionale Modelle gegeneinander zu verifizieren. Auch wenn ein neues Modell durch Kopie und Modifikation eines existierenden erstellt wird, ist der dabei entstehende Aufwand nicht zu unterschätzten, da eine aufwendige Umstrukturierung des Modells erforderlich sein kann. Außerdem ist es problematisch, eventuell erforderliche nachträgliche Algorithmenänderungen zu berücksichtigen, da diese Modifikationen in allen Architekturmodellen zur Folge haben.

5.4.5 Kopplung Architekturmodell - Software-Algorithmenmodell

Um die Einschränkungen reiner Architektursimulation zu vermeiden und um einen Vergleich von Architekturen mit geringem Modellierungsaufwand zu ermöglichen, werden Algorithmus und Architektur in getrennten komplementären Modellen beschrieben, die so miteinander gekoppelt werden, daß sich ein simulierbares Gesamtmodell ergibt. Komplementarität der Modelle bedeutet, daß sich Algorithmen- und Architekturmodell zu einem Gesamtmodell ergänzen, das alle relevanten Aspekte beschreibt, und daß jeder Aspekt nur genau einmal beschrieben ist. Die Komplementarität ist wichtig, da hierdurch sichergestellt wird, daß keine unnötige Modellierung bzw. Verifikation erforderlich ist.

In [LivdW99] ist eine trace-gesteuerte Methodik zur Architekturexploration beschrieben, die ebenfalls auf der Trennung zwischen Algorithmus und Architektur beruht, wobei das Algorithmenmodell dort als Applikation bezeichnet wird. Für die Modellierung der Applikation wird ein Kahn-Prozeß-Netz (KPN) eingesetzt, die Simulation erfolgt durch Abbildung von Prozessen auf sogenannte trace-gesteuerte Ausführungseinheiten. Da der Algorithmus bei diesem Ansatz in Prozesse eines KPN partitioniert wird, müssen Schnittstellen in das Algorithmenmodell eingeführt werden, so daß bereits bei dessen Erstellung Überlegungen hinsichtlich der für die zu untersuchenden Architekturen erforderlichen Granularität eine Rolle spielen. Bei der hier vorgeschlagenen Methodik, wird stattdessen von einer Implementierung des Algorithmus in Software mit der Programmiersprache C ausgegangen. Dabei ist keine Einführung von Schnittstellen beim Algorithmenentwurf erforderlich, und es können Standardwerkzeuge für die Algorithmenentwicklung eingesetzt werden. Anstelle einer festen Abbildung der Teile eines partitionierten Algorithmenmodells auf Architekturelemente erfolgt eine flexible Kopplung von Architekturelementen an ein nicht partitioniertes Algorithmenmodell. Die Kopplung ermöglicht einen weiten Bereich virtueller Unterteilungen und Zuordnung zu Architekturelementen, insbesondere auch solche, die von der Struktur des Algorithmus abweichen.

Bei der Simulation von Architekturen ist die Operationsausführung ein wichtiger Gesichtspunkt. Die Art und Anzahl von Operationen wird durch das Algorithmenmodell beschrieben, die Zuordnung der Operationen zu Funktionseinheiten und deren Verhalten jedoch durch das Architekturmodell. Damit im Architekturmodell die Operationsausführung simuliert werden kann, muß die Informationen über das Pensum durchgeführter Operationen vom Algorithmenmodell an dieses weiter gegeben werden. Aus diesem Grund findet eine Operationskopplung statt, bei der das Algorithmenmodell das Architekturmodell steuert.

Neben der Operationsausführung spielen auch die Datenzugriffe eine zentrale Rolle. Die Art, Anzahl und Abfolge von Zugriffen werden durch das Algorithmenmodell beschrieben, die Datenorganisation, d.h. die Zuordnung zu speichernden Elementen und deren Verhalten durch das Architekturmodell. Daher erfolgt in analoger Weise eine Kopplung von Datenzugriffen, d.h. das Architekturmodell wird durch das Algorithmenmodell gesteuert. Bei der Modellierung von Pufferspeichern, die nur einen Teil der über sie zugegriffenen Daten tatsächlich enthalten, muß z.B. im Architekturmodell der Belegungszustand modelliert werden, um bei einem Zugriff bestimmen zu können, ob der Pufferspeicher das zugegriffene Datum enthält. Für die stattfindenden Zugriffe, kann der Inhalt der zugegriffenen Daten außer Acht gelassen werden, es ist jedoch nötig, daß die Adressen der Zugriffe im Architekturmodell bereitstehen, d.h. nicht nur die Art und Anzahl an Zugriffen, sondern auch die Adressen müssen gekoppelt werden.

Zur Implementierung des Architekturmodells wird ebenfalls die Programmiersprache C verwendet. Das Architekturmodell wird durch eine Verschaltung von Architekturelementen gebildet, wobei in einer Konfigurationsdatei definiert wird, welche Architekturelemente benutzt werden und wie diese miteinander verbunden sind. Die Operation von Architekturelementen wird durch Funktionsaufrufe angestoßen. Die Verschaltung der Modelle erfolgt dadurch, daß der Simulator die Architekturbeschreibung in einer Konfigurationsdatei liest, die Architekturelemente instanziert, und für jede Instanz die aufzurufenden Architekturelemente durch das Setzen von Zeigern auf die korrespondierenden Funktionen der verbundenen Architekturelemente konfiguriert. Neben der Verschaltung werden zur Konfigurationszeit auch die Parameter der Architekturelemente gesetzt. Die anschließende Simulation erfolgt dadurch, daß der Simulator für jeden aktivierten Koppelpunkt (Ereignis) entsprechend der Festlegung in der Konfigurationsdatei ein bestimmtes Architekturelement aufruft.

Neben der Unterteilung des Algorithmenmodells, der Modellierung der Architektur und der Modellkopplung ist auch die Festlegung des zeitlichen Ablaufes (Schedule) erforderlich, um ein simulierbares Gesamtmodell zu erhalten. Da jedoch das Ziel dieser Arbeit der Verglich von Architekturvarianten für das CONIAN Architekturkonzept ist, ist es nicht erforderlich eine Methodik zu entwickeln, die für beliebige Schedules einsetzbar ist. Vielmehr ist es ausreichend, wenn sich der Ablauf durch die Verschaltung von Architekturelementen ergibt. Darauf wird in Kapitel 5.4.10 näher eingegangen.

5.4.6 Operationskopplung und Granularität

Um eine Operationskopplung zu ermöglichen, muß das Pensum durchgeführter Operationen bestimmt werden, d.h. es muß festgelegt werden, welche Teile des Algorithmus einer Operation entsprechen. Bekannte Verfahren wie [LivdW99] beruhen auf einer Aufteilung eines Algorithmenmodells in Prozesse eines KPNs bzw. allgemein auf einer Partitionierung des Algorithmenmodells durch Einführung von Schnittstellen, wie in Bild 5.7 (a) gezeigt.

Da bei verschiedenen Architekturvarianten unterschiedliche Algorithmenteile einer Operation entsprechen können, ist es für den Vergleich von Architekturen wichtig, daß die Unterteilung leicht änderbar ist. Auch weil bei der Erstellung des Algorithmenmodells nicht grundsätzlich absehbar ist, welche Algorithmenteile einer Operationen entsprechen können, darf die Unterteilung nicht bereits im Algorithmus enthalten sein. Vielmehr müssen je nach betrachteter Architekturvariante unterschiedliche Unterteilungen und Zuordnungen zu Operationen von Architekturelementen möglich sein.

Diese Überlegungen sprechen gegen die Partitionierung durch Einführung von Schnittstellen, d.h. im Algorithmenmodell tatsächlich vorhandene Unterteilungen, die durch Sprachelemente wie z.B. Prozesse (in KPNs) oder Funktionsaufrufe (in C-Programmen) dargestellt werden. Zweckmäßiger ist hingegen, eine virtuelle Unterteilung in logische Abschnitte des Algorithmenmodells vorzunehmen, die nicht durch Sprachelementen beschrieben wird.

Bei einer Unterteilung in logische Abschnitte gibt es jedoch keine durch Sprachelemente gekennzeichneten Algorithmenteile, die der Operationen eines Architekturelements direkt zugeordnet werden könnten. Um trotzdem eine Verbindung zwischen dem Algorithmenmodell und Architekturmodell zu erreichen, bei der das Architekturmodell durch das Algorithmenmodell gesteuert wird, erfolgt eine Kopplung, wie in Bild 5.7 (b) gezeigt. Dabei werden Koppelpunkte im Algorithmenmodell eingeführt, bei deren Erreichung die Operation von Architekturelementen angestoßen wird. Da die Kopplung so erfolgen muß, daß die Ausführung eines logischen Abschnittes zur Operation eines Architekturelements führt, müssen die Koppelpunkte so gewählt werden, daß diese genau einmal erreicht werden, wenn ein logischer Abschnitt ausgeführt wird. Bevor auf die Wahl der Koppelpunkte eingegangen wird, muß jedoch diskutiert werden, wie detailliert Vorgänge im Algorithmus betrachtet werden müssen.

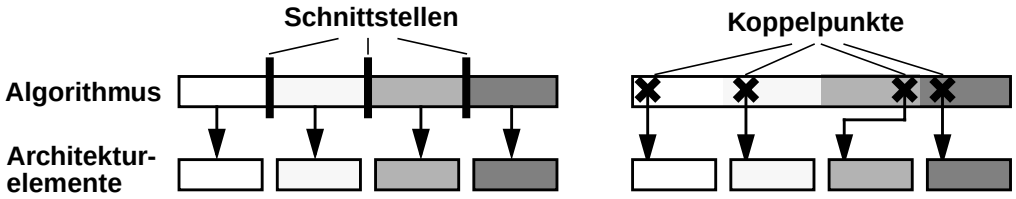


Bild 5.7: (a) Zuordnung part. Algorithmenteile (b) Kopplung logischer Abschnitte

Für eine grobe Simulation der Verarbeitungsgeschwindigkeit müssen architekturelle Vorgänge auf einer abstrakten Ebene modelliert werden, d.h. komplexe Vorgänge im Algorithmus werden als Einheit betrachtet, die der Operation eines Architekturelements entspricht. Für detailliertere Simulationen kritischer bzw. häufiger Vorgänge ist es erforderlich das Architekturmodell zu verfeinern, so daß ggf. auch die Vorgänge im Algorithmus detaillierter betrachtet werden müssen. Da nur für das Simulationsergebnis relevante Vorgänge verfeinert werden sollen, ist eine Möglichkeit zur Simulation mit ungleichem Detailierungsgrad erforderlich.

Der Detailierungsgrad, mit dem Abläufe in einem Algorithmus betrachtet werden, wird als Granularität bezeichnet. Dabei muß zwischen der Modellgranularität und der Simulationsgranularität unterschieden werden. Die Modellgranularität entspricht der im Algorithmenmodell enthaltenen feinstmögliche Auflösung, mit der Abläufe in einem Algorithmus betrachtet werden können. Sie kann somit auch als potentielle Granularität gesehen werden. Die Simulationsgranularität stellt hingegen die Auflösung dar, mit der die Vorgänge im Algorithmus zur Architektursimulation betrachtet werden, und kann somit als tatsächliche genutzte Granularität gesehen werden.

Da sich Abläufe nur an den Stellen unterscheiden können, an denen sich in Programm Verzweigungen (bedingte Sprünge bzw. bedingte Unterprogrammaufrufe) bzw. Verzweigungsziele befinden, stellt die Basic-Block-Struktur eines Algorithmus den höchsten Detailierungsgrad dar, mit dem dieser betrachtet werden kann. Als Basic-Block wird dabei ein Algorithmenabschnitt bezeichnet, der zwischen Verzweigungen bzw. Verzweigungszielen liegt, und (abgesehen von Interrupts) immer als Ganzes durchlaufen werden muß. Auch wenn ein Basic-Block der Operation mehrerer Architekturelemente entspricht, ist keine feinere Auflösung erforderlich, da sich die Operationsreihenfolge der Architekturelemente nicht ändert.

Bekannte Verfahren wie [LivdW99] benutzen zur Definition der Simulationsgranularität einen Top-Down-Ansatz, bei dem ein Modell durch Einführung von Schnittstellen strukturiert werden muß, um eine Partitionierung zu definieren und eine Abbildung zu Architekturelementen zu ermöglichen. In diesem Sinne wird von maximal grober Granularität ausgegangen und durch die Strukturierung eine feiere Simulationsgranularität festgelegt. Im Unterschied dazu handelt es sich bei der vorgeschlagenen Methodik um einen Bottom-Up-Ansatz, bei dem von der feinstmöglichen Unterteilung des Algorithmenmodells, der Unterteilung in Basic-Blocks, ausgegan-

gen wird, und durch Zusammenfassung dieser Abschnitte die Simulationsgranularität definiert wird. Daraus ergibt sich der Vorteil, daß volle Freiheit bezüglich der Kopplung der Architekturelemente besteht ohne den Algorithmus umzustrukturieren.

In Bild 5.8 ist exemplarisch der Unterschied zwischen Modellgranularität und der Simulationsgranularität anhand eines Struktogramms gezeigt, das einen Programmabschnitt mit den Sprachelementen Verzweigung, Schleife, Funktionsaufruf und Operationsausführung enthält. Die fett dargestellten Linien kennzeichnen die Simulationsgranularität, während die dünn dargestellten Linien die Modellgranularität repräsentieren. Bei der Kopplung des Algorithmenmodells an das Architekturmodell werden nur Operationen auf Ebene der Simulationsgranularität betrachtet. Für eine solche Operation wird entsprechend der festgelegten Kopplung ein Architekturelement aufgerufen, Operationen unterhalb der Granularitätsebene, wie beispielsweise Aufrufe der Funktion Fkt.1, werden nicht weiter beachtet. Die Wahl der Simulationsgranularität ist von den zu untersuchenden Architekturen abhängig. Für das CONIAN-Architekturkonzept wird darauf in Kapitel 5.5 näher eingegangen.

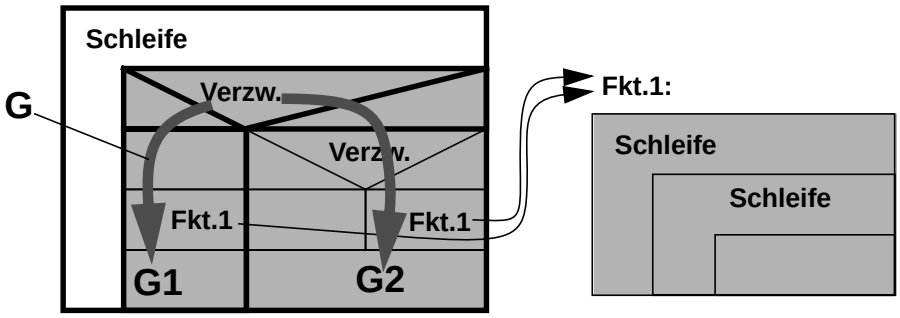


Bild 5.8: Granularität und Operationsgruppierung

Um eine Änderung der Simulationsgranularität für unterschiedliche Architekturen zu vermeiden, können mehrere Operationen zu einer Operationsgruppe zusammengefaßt werden, die mit einem Architekturelement korrespondiert. Außerdem kann innerhalb einer Operationsgruppe unterschieden werden, welcher der möglichen Zweige auf Ebene der Simulationsgranularität beim Ablauf des Algorithmus ausgeführt wird. In Bild 5.8 ist die Wahl einer Operationsgruppe exemplarisch gezeigt. Der grau markierte Bereich G stellt eine Operationsgruppe dar, innerhalb der die Zweige G1 oder G2 durchlaufen werden können.

Ein Koppelpunkt kann innerhalb eines Basic-Blockes beliebig plaziert werden, der Basic-Block muß jedoch so gewählt werden, daß er bei Ausführung einer Operationsgruppe nur einmal erreicht wird. Die Kopplung bei Eintritt oder Austritt aus einer Operationsgruppe kann durch Wahl des Koppelpunktes direkt vor bzw. nach der Operationsgruppe implementiert werden. Auch die Unterscheidung der Zweige kann auf vergleichbare Weise durchgeführt werden. Die Einfügung der Kopplungsfunktionen kann entweder manuell erfolgen oder sie kann automatisiert werden, indem die Markierungen der Simulationsgranularität und der Operationsgruppen z.B. mit einem Struktogrammeditor erfolgt, und das Einlesen der Programmstruktur und das Einfügen der Kopplungsfunktionen mit einem Parser und einem Codegenerator erfolgt. Durch die Festlegung der Simulationsgranularität und der Operationsgruppen ist außerdem die Möglichkeit gegeben, Ergebnisse eines Instruktions-Profilings zu Architekturelementen zuzuordnen, so daß ein Vergleich zur Komplexität der Software-Implementierung möglich ist.

5.4.7 Datenkopplung

Zur Simulation von Datenzugriffen in Architekturen ist es erforderlich, Variablenzugriffe im

Algorithmenmodell für die Steuerung von Architekturmodellen zu benutzen. Da sich die Operation der Architektur je nach Art des Zugriffs unterscheiden kann, wird im Algorithmus zwischen Lese- und Schreibzugriffen unterschieden. Neben der Kopplung von Variablenzugriffen ist auch die Kopplung von Array-Zugriffen möglich, da Arrays Speicherblöcken in Architekturen entsprechen können. Die Indizes bei Array-Zugriffen haben dann im Architekturmodell die Rolle von Speicheradressen, so daß ihr Wert ebenfalls für die Kopplung benutzt wird. Anders als bei Operationskopplung ist zusätzlich zur Zuordnung der Variablen bzw. Arrays zu Architekturelementen auch eine Umrechnung von Indizes in Speicheradressen entsprechend der simulierten Speicherorganisation möglich.

Bei der Spezifikation der Kopplung werden die Variablen bzw. Arrays im Algorithmus festgelegt, deren Datenorganisation im Architekturmodell simuliert wird. Da Datenzugriffe je nach Position im Algorithmus, an der sie auftreten, Zugriffen auf unterschiedliche Speicher in der Architektur entsprechen können bzw. in manchen Fällen nur Teile des Algorithmus auf eine Architektur abgebildet werden, und daher Zugriffe nur innerhalb bestimmter Abschnitte simuliert werden dürfen, ist eine Spezifikation der relevanten Abschnitte des Algorithmus erforderlich. Bild 5.9 zeigt ein Beispiel für die Kopplung aller Array-Zugriffe im grau markierten Bereich.

Die Kopplung von Datenzugriffen kann sowohl durch manuelle Einfügung von Kopplungsfunktionen erfolgen, als auch automatisch basierend auf der Festlegung relevanter Variablen und Bereiche. Die automatische Kopplung von Datenzugriffen kann durch Einfügung von Syntaxelementen in den Algorithmus erfolgen, die außerhalb des normalen Sprachumfanges liegen, und die in einem Vorverarbeitungsschritt in zusätzlichen Programmcode für die Kopplung umgewandelt werden. Da bei der Übersetzung von mit der Programmiersprache C/C++ erstellten Programmen bereits im Rahmen des normalen Übersetzungsprozesses ein Vorverarbeitungsschritt erfolgt, läßt sich auch durch Syntaxelemente innerhalb des normalen Sprachumfanges wie z.B. Makros oder Bibliotheksfunktionen, die automatische Erzeugung von Programmcode für die Kopplung zu veranlassen. Für die Datenkopplung wurde eine C++ Bibliothek erstellt, die mit speziellen Makros und Klassen die Kopplung von Zugriffen und Adressen lokaler und globaler Variablen bzw. Arrays implementiert und mit der in Bild 5.9 gezeigten Syntax benutzt werden kann.

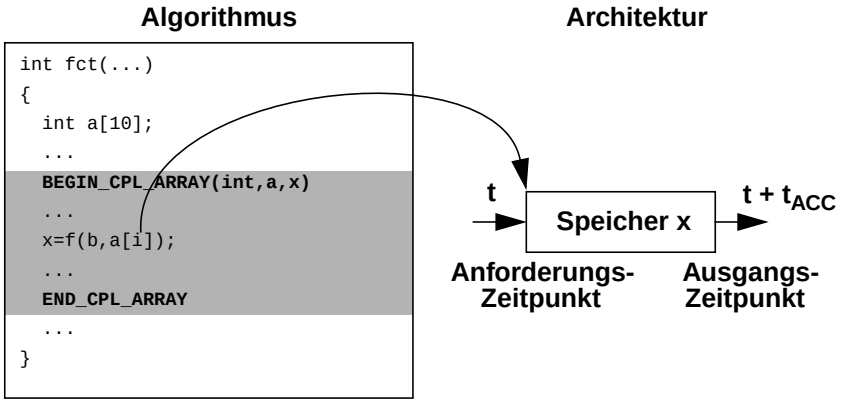


Bild 5.9: Kopplung von Array-Zugriffen in einem selektierten Bereich

5.4.8 Kopplung von Betriebsarten

Zur Vereinfachung der Kopplung bzw. der Architekturmodellierung kann es zweckmäßig sein, bereits im Algorithmenmodell definierte Betriebszustände zur Steuerung des Architekturmodells zu verwenden. Auf diese Weise kann das Architekturmodell gesteuert werden, ohne daß die vom Betriebsmodus abhängigen Details der Operationsausführung bzw. Datenzugriffe aufgezeichnet werden müssen. Im Unterschied zur Kopplung der Operationsausführung und von Datenzugrif-

fen erfolgt hier die explizite Nutzung von Daten des Algorithmus, wodurch eine Reduzierung der entstehenden Datenmenge bei getrennter Simulation (siehe Kapitel 5.4.9) erreicht werden kann. Es ist jedoch nicht erforderlich, hierfür das Algorithmenmodell zu modifizieren, da alternativ auch eine Operations- und Datenkopplung erfolgen kann.

5.4.9 Getrennte vs. integrierte Simulation

Die Kopplung zwischen Algorithmenmodell und Architekturmodell kann durch Aufzeichnung der relevanten Operationen, Datenzugriffe und Operationsmodi während der Algorithmensimulation erfolgen. Algorithmen- und Architektursimulation erfolgen in diesem Falle getrennt, wie in Bild 5.10 gezeigt. Während der Algorithmensimulation werden die relevanten Vorgänge in einer Datei aufgezeichnet. Wenn deren Anzahl klein ist, ist eine direkte Aufzeichnung zweckmäßig, andernfalls ist eine Kompression bei der Aufzeichnung erforderlich bzw. eine Akkumulation bei Vorgängen, bei denen die Abfolge keine Rolle spielt. Die Architektursimulation wird von den in der Datei aufgezeichneten Vorgängen gesteuert. Neben der Architektursimulation ist eine statistische Auswertung des Algorithmus im Sinne eines Profiling ohne Berücksichtigung von Architekturmodellen möglich.

Vorteil der Kopplung über eine Datei ist die Entkopplung von Algorithmus und Architektur hinsichtlich der Simulation. Dadurch ist eine mehrfache Simulation des Algorithmus beim Vergleich mehrerer Architekturen unnötig, und es lassen sich auch für Algorithmen, deren Verhalten nicht reproduzierbar ist, verschiedene Architekturen für denselben Algorithmenlauf vergleichen, während ansonsten das Algorithmenverhalten nicht reproduzierbar wäre. Auch wenn das Verhalten des Algorithmus deterministisch ist, bietet die Kopplung über eine Datei den Vorteil, daß der komplette Satz an Eingangsdaten plus zugehörige Algorithmenversion in der Aufzeichnung festgehalten wird, und daß sich kritische Stellen des Algorithmenlaufes ausschneiden und gezielt simulieren lassen.

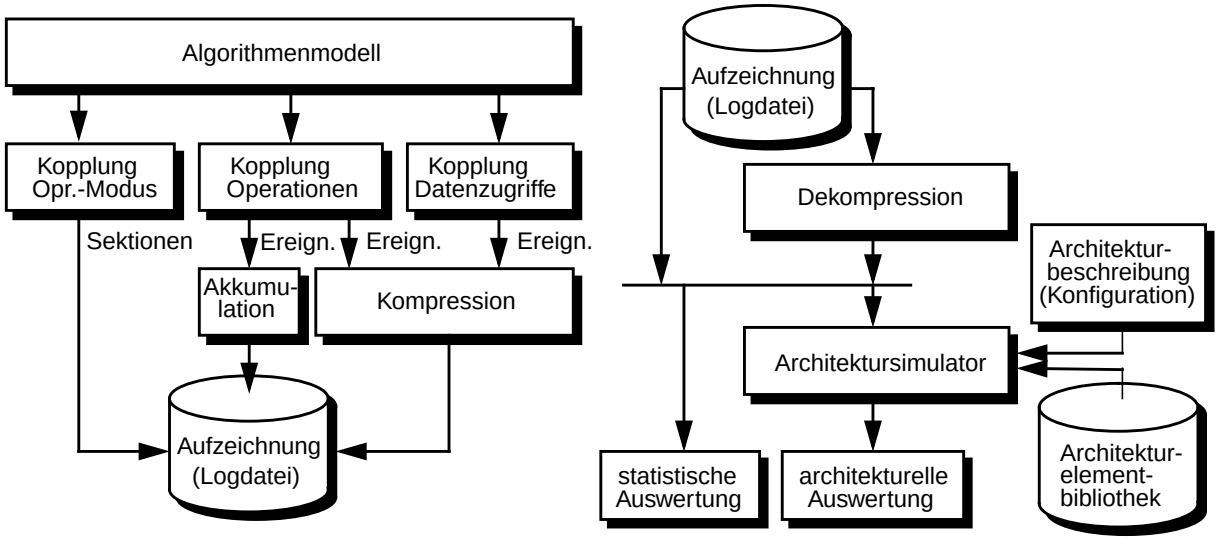


Bild 5.10: Algorithmen- und Architekturmodell bei getrennter Simulation

Nachteilig ist jedoch die Größe der entstehenden Datei, die zur Simulation längerer Vorgänge eine Kompression vor der Aufzeichnung bei der Algorithmensimulation und eine Dekompression während der Architektursimulation erforderlich macht, und die maximale Simulationsdauer beschränkt. Durch integrierte Simulation von Algorithmenmodell und Architekturmodell kann dieser Nachteile vermieden werden. In diesem Fall steuern die relevanten Operationen und Zugriffe das Architekturmodell direkt, Algorithmen- und Architektursimulation erfolgen gleichzeitig, wie in Bild 5.11 gezeigt. Durch das Wegfallen der Aufzeichnung wird die Simulation län-

gerer Vorgänge möglich.

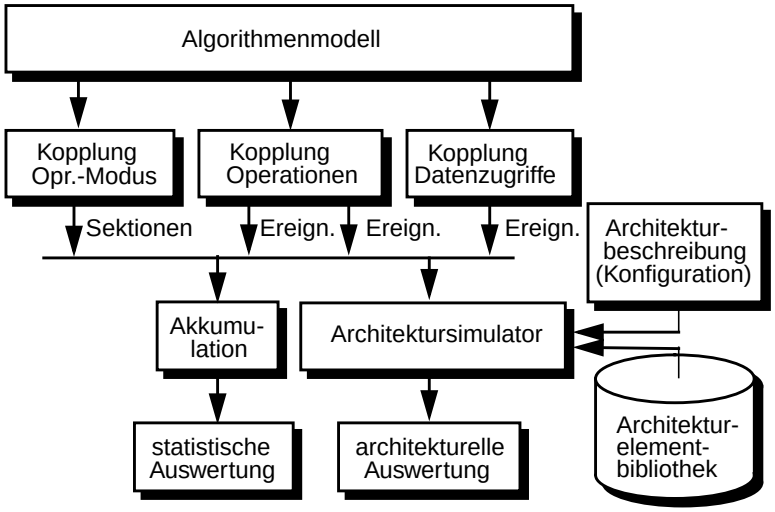


Bild 5.11: Algorithmen- und Architekturmodell bei integrierter Simulation

5.4.10 Architekturelemente und Verschaltung

Die Grundelemente, durch deren Verschaltung Architekturmodelle aufgebaut werden, werden als Architekturelemente bezeichnet. Die Architekturelemente besitzen Ein- und Ausgänge, über die sie untereinander in Wechselwirkung stehen bzw. durch das Architekturmodell gesteuert werden. Die Steuerung erfolgt über Eingänge zur Signalisierung gekoppelter Aktionen des Algorithmus, d.h. Operationen, Datenzugriffe, Adressen und Betriebsmodi. Die Wechselwirkung untereinander erfolgt ebenfalls durch Signalisierung von Aktionen, zusätzlich können auch Zeitpunkte, zu denen Daten bzw. Ressourcen operieren bzw. bereitstehen, und Häufigkeiten von Vorgängen signalisiert werden. Im Gegensatz zu Hardware-Modellen, die auch die Funktion beschreiben, sind jedoch keine Verbindungen für den Austausch von Daten erforderlich.

Die Architekturelemente stellen Modelle dar, die das nach außen sichtbare Verhalten von Teilen einer Architektur beschreiben. Neben dem Beginn und dem Ende einer Operation sind dies Zeitpunkte, zu denen das Architekturelement mit anderen Architekturelementen interagiert, und Aktionen anderer Architekturelemente und Häufigkeiten von Vorgängen, die das Architekturelement steuert. Da der interne Ablauf nicht modelliert wird, lassen sich viele Architekturelemente auf Modelle generischer Grundelemente mit architekturspezifischen Parametern zurückführen, so daß eine Bibliothek dieser Elementen erstellt werden kann.

Architekturelemente zur Modellierung von Funktionseinheiten unterscheiden sich von Architekturelementen zur Modellierung von Speicherelementen durch die Art der Aktionen, durch die sie gesteuert werden. Während ein Architekturelement zur Modellierung von Funktionseinheiten durch Operationen gesteuert wird, geschieht dies bei einem Architekturelement zur Modellierung von Speicherelementen durch Datenzugriffe.

In Bild 5.12 (a) und (b) sind Beispiele für generische Architekturelemente zur Modellierung von Funktionseinheiten gezeigt. (a) zeigt eine Funktionseinheit, die eine feste Operation durchführt, während (b) eine Funktionseinheit zeigt, bei der die Auswahl der Operation durch den Algorithmus oder ein anderes Architekturelement gesteuert wird. Während es sich bei (a) um eine einfache Funktionseinheit handelt, ist (b) eine Pipeline-Funktionseinheit, die Operationen überlappend ausführen kann. (c) zeigt exemplarisch Parameter für das Architekturelement (c).

In Bild 5.12 (d) und (e) sind Beispiele für generische Architekturelemente zur Modellierung von

Speicherelementen gezeigt. Bild 5.12 (d) zeigt eine Architekturelemente zur Modellierung einer FIFO bzw. eines Pufferspeichers. Das Architekturelement wird durch die Aktionen Schreiben und Lesen gesteuert. Das Modell in Bild 5.12 (e) beschreibt einen Speicher der wahlfrei gelesen und beschrieben werden kann. Je nachdem ob es sich um ein Modell für SRAM oder DRAM handelt, wird die Adresse entweder ignoriert oder dazu benutzt, um zu bestimmen, ob ein Pagewechsel erfolgen muß.

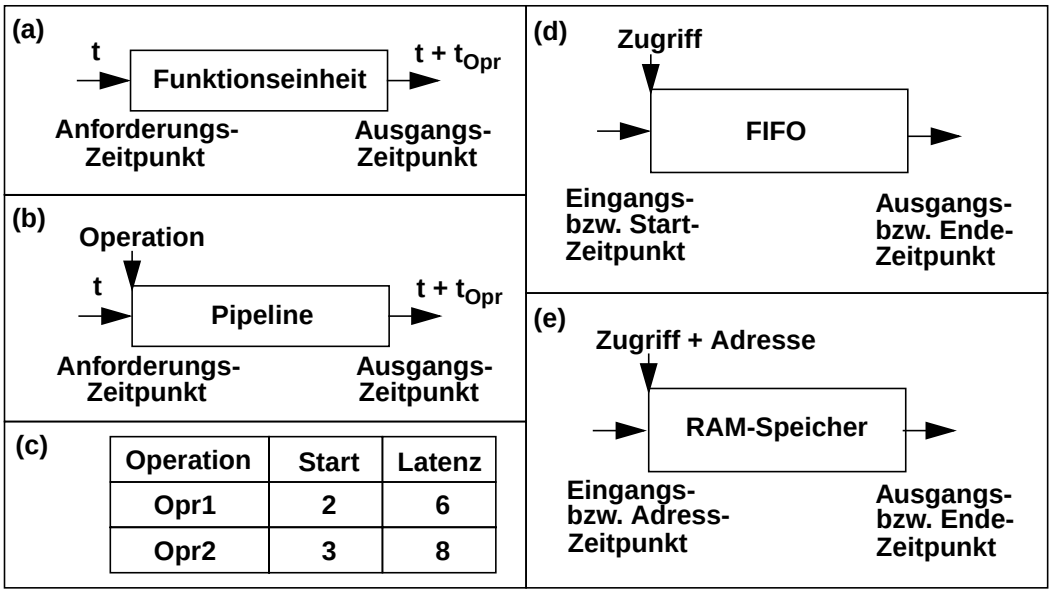


Bild 5.12: Beispiele generischer Architekturelemente für Funktionseinheiten

Die Architekturelemente werden durch Funktionen modelliert, die mit der Programmiersprache C implementiert sind. Bei dieser Art der Modellierung ist der Ablauf von Vorgängen im Architekturmodell (Schedule) durch die Reihenfolge der Funktionsaufrufe im Algorithmenmodell festgelegt. Parallele Vorgänge lassen sich modellieren, da die Operationszeitpunkt der Architekturelemente unabhängig voneinander verwaltet werden können. Da jedoch der Schedule bereits bei der Modellierung festgelegt wird, ist der Vergleich unterschiedlicher Schedules eingeschränkt. Im Rahmen dieser Arbeit wird diese Einschränkung in Kauf genommen, da dies zur Simulation der CONIAN-Architekturvarianten bereits ausreichend ist.

Die Struktur des Modells aus Architekturelementen und die Struktur der Architektur müssen nicht identisch sein, wenn sich eine Architektur durch ein anders strukturiertes Modell besser darstellen läßt. In Bild 5.13 (a) ist eine aus Verarbeitungseinheit, Cache-Speicher und weiteren Blöcken bestehende Architektur gezeigt. Das Laden der Daten aus dem Cache, die Verarbeitung und das Speichern des Verarbeitungsergebnis erfolgen zeitlich überlappend. In Bild 5.13 (b) ist das korrespondierende Modell aus Architekturelementen gezeigt. Das Cache-Modell bestimmt, für welche Zugriffe Blocktransfers zum Hintergrundspeicher nötig sind, und steuert entsprechend die Aktionen der Pipeline. Der zeitlichen Ablauf in der Architektur wird vom Architekturelement Pipeline bestimmt. Dementsprechend modelliert dieses Architekturelement das Verhalten mehrerer Hardware-Module. Um die Parameter des externen Speichers von denen der Pipeline zu trennen, wird die Dauer von Blocktransfers durch eine eingenständiges Modell beschrieben.

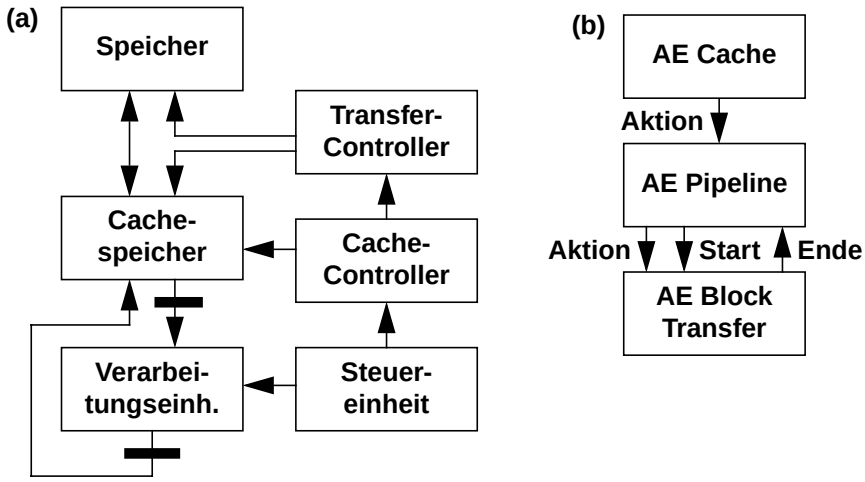


Bild 5.13: Gegenüberstellung von (a) Architekturaufbau und (b) Architekturmodell

5.5 Simulationsmodell für CONIAN

Zum Einsatz der Simulationsmethodik für die in Kapitel 4 vorgeschlagene Architektur wurde ein Modell für den CONIAN-Koprozessor erstellt, das zur Algorithmensimulation und zur Simulation der Verarbeitungsgeschwindigkeit von Architekturalternativen und deren Vergleich eingesetzt werden kann. Um einzelne Operationen bzw. Algorithmenteile gut simulieren zu können, und wegen der Entkopplung von Algorithmenmodell und Architekturmodellen wurde eine getrennte Simulation implementiert.

5.5.1 Algorithmenmodell und AddressLib

Das Algorithmenmodell umfaßt den kompletten Satz der von CONIAN unterstützten Operationen. Es erlaubt die Simulation des in Kapitel 2.6 untersuchten Farbsegmentierungsalgorithmus, sowie die Simulation anderer Algorithmen, die von der CONIAN-Architektur unterstützt werden. Das Algorithmenmodell besteht aus einem Teil für Low-Level-Operationen und einem Teil für High-Level-Operationen. Die Umsetzung der von der CONIAN-Architektur unterstützten Low-Level-Operationen im Rahmen des Algorithmenmodells erfolgt in Form einer Bibliothek, die als *AddressLib* bezeichnet wird [HeMo97b], [HeMo99a].

Wie der CONIAN-Architektur auch, liegt der AddressLib die Trennung von Adressierung und Verarbeitung zugrunde. Die AddressLib stellt den Satz generischer Adressierungsfunktionen bereit, der in Kapitel 2.5 aufgezeigt wurde. Zusammen mit operationsspezifischen Verarbeitungsfunktionen erlaubt die AddressLib die Implementierung von Low-Level-Operationen.

Durch die Ähnlichkeit zwischen der Struktur der AddressLib und der Struktur der vorgeschlagenen Architektur können die Programmabschnitte leicht Hardware-Operationen zugeordnet werden. Das Algorithmenmodell wurde durch Aufzeichnung von Operationen, Datenzugriffen und Betriebsmodi erweitert, so daß es die Informationen zur Steuerung der Architektursimulation bereitstellt. Bild 5.14 zeigt die Struktur der AddressLib. Zu jeder generischen Adressierungsart enthält die AddressLib eine Adressierungsfunktion, die Laden und Speichern der Bilddaten entsprechend der beim Aufruf übergebenen Parameter durchführt. Die Adressierungsfunktionen werden durch den High-Level-Algorithmus aufgerufen.

Zur Verarbeitung der Bilddaten rufen die Adressierungsfunktionen Verarbeitungsfunktionen auf, die durch Funktionszeiger spezifiziert werden. Bei Inter- und bei Intra-Adressierung übernimmt die Verarbeitungsfunktion bei jedem Aufruf die Berechnung des Ergebnisses für einen Bild-

punkt. Bei Segment-Adressierung gibt es drei Verarbeitungsfunktionen, die für die Berechnung von Startkriterium, Nachbarschaftskriterium und Verarbeitungsergebnis für eine Position bzw. für einen bestimmten Nachbarn zuständig sind. High-Level-Algorithmus und Verarbeitungsfunktionen sind nicht Teil der AddressLib.

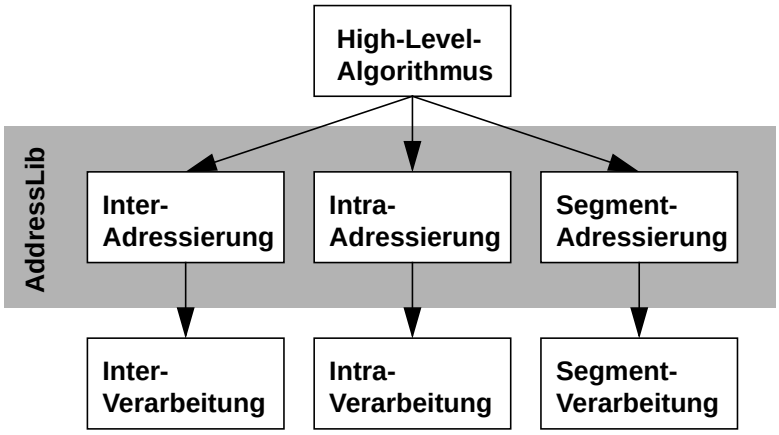


Bild 5.14: Aufbau des Algorithmenmodells und der AddressLib

Um Low-Level-Operationen effizient zu implementieren, muß Adressierung und Datenzugriff effizient erfolgen. Aus diesem Grund wird in der AddressLib Zeigerarithmetik benutzt, die zwar höheren Implementierungsaufwand erfordert, dafür jedoch zu einer effizienten Implementierung führt. Gegenüber einer operationsspezifisch optimierten Implementierung ergibt sich aufgrund des generischen Charakters der AddressLib zusätzlicher Rechenaufwand, der im ungünstigsten Fall bei Inter-Adressierung 90%, bei Intra-Adressierung 75% und bei Segment-Adressierung unter 50% des gesamten Rechenaufwandes beträgt. Zeigerarithmetik führt grundsätzlich zu einer größeren Gefahr von Implementierungsfehlern. Da die AddressLib die Adressierung von Bilddaten an einer zentralen Stelle durchführt, und da sie intensiv getestet wurde, reduziert ihr Einsatz die Fehlerwahrscheinlichkeit gegenüber operationsspezifisch implementierten Low-Level-Operationen. Die AddressLib bietet auch bei der Algorithmenentwicklung Vorteile, da lediglich der applikationsspezifische High-Level-Algorithmus und die Verarbeitungsfunktionen implementiert werden müssen, und so eine schnelle Implementierung von Videosegmentierungsverfahren durch Zusammensetzung aus Low-Level-Operationen ermöglicht wird. Die AddressLib stellt damit zum einen eine Plattform für Algorithmenentwicklung und –optimierung dar, zum anderen jedoch auch eine Plattform zur Evaluierung von Segmentierungsalgorithmen im Hinblick auf Implementierungseigenschaften und zur Simulation von Architekturalternativen für das in Kapitel 4 vorgestellte Architekturkonzept.

5.5.2 Kopplung von Algorithmus und Architektur

5.5.2.1 Kopplung von Betriebsmodi

Im Rahmen der Kopplung von Betriebsmodi wird vor dem Aufruf einer Adressierungsfunktion der Beginn eines Abschnittes und danach das Ende des Abschnittes inklusive zugehöriger Betriebsmodi aufgezeichnet. Im einzelnen sind das:

- Adressierungsart (Inter-/Intra-/Segment-Adressierung)
- Benutzte Komponenten der Eingangs- und Ausgangsbilder
- Größe der Eingangs- und Ausgangsbilder
- Scan-Modus
- Position und Größe des Scan-Fensters
- Nachbarschaftssystem

- Art der Verarbeitungsoperation

Bei Segmentadressierung werden zusätzlich folgende Modi aufgezeichnet:

- Nachbarschaftssystem für den Ausbreitungsprozeß
- Verarbeitungsmodus (PQI/PQO)
- Art des Startkriteriums
- Art des Nachbarschaftskriteriums

5.5.2.2 Kopplung der Operationsausführung

Im Falle regulärer Adressierungsarten kann aus der Bildgröße und dem Scan-Modus die Operationshäufigkeit und -abfolge abgeleitet werden, so daß die Operationsausführung aus Gründen der Datenkompression nicht aufgezeichnet wird. Bei der Segment-Adressierung ist die Operationshäufigkeit und -abfolge von den verarbeiteten Daten abhängig, so daß eine Aufzeichnung zur Kopplung von Algorithmenmodell und Architekturmodell erforderlich ist. Während in der Software zwischen

- Überprüfung des Startkriteriums
- Überprüfung des Nachbarschaftskriteriums
- Durchführung der Verarbeitungsoperation

unterschieden wird und diese sequentiell ausgeführt werden, erfolgt in der Hardware die Durchführung der Verarbeitungsoperation und die Überprüfung des Nachbarschaftskriteriums für alle Nachbarn parallel. Bei der Architektursimulation wird daher nicht zwischen Nachbarschaftskriteriumsüberprüfung und Verarbeitungsoperation unterschieden.

Beide Hardware-Operationen können je nach Erfüllung der Kriterien unterschiedlich lange dauern, so daß eine Aufzeichnung der jeweils eintretenden Fälle erfolgt. In Bild 5.15 ist die Kopplung bei Segment-Adressierung im Fall des Non-Collected-Modus mit PQO dargestellt. Nicht abgebildet ist der Collected-Modus, bei dem der Ausbreitungsprozeß (Schleife „solange Koordinatenspeicher nicht leer“) erst nach dem Ende der Startpunktsuche erfolgt, die Kopplung jedoch auf dieselbe Weise geschieht. Wird die Segment-Adressierung im PQI-Modus durchgeführt, wird die Verarbeitungsoperation bei erfülltem Nachbarschaftskriterium für einen Bildpunkt vor dem Speichern seiner Position im Koordinatenspeicher ausgeführt. Auch in diesem Fall geschieht die Kopplung auf dieselbe Weise.

Im Struktogramm ist der Bereich, der der Startkriteriumsüberprüfung entspricht, und der Bereich, der der Nachbarschaftskriteriumsüberprüfung entspricht, grau schattiert. Die Schattierungen entsprechen jeweils einer Operationsgruppe mit den Zweigen S0 bis S2 bzw. N0 bis N3. Je nachdem welcher Zweig im Start- oder Nachbarschaftskriteriums durchlaufen wird, wird ein entsprechendes Ereignis aufgezeichnet. Folgende Fälle sind möglich:

S0: Das Markerkriterium ist nicht erfüllt. Die Startkriteriumsüberprüfung kann abgebrochen werden, die Koordinaten des Bildpunkts werden nicht im Koordinatenspeicher abgelegt.

S1: Das Markerkriterium ist erfüllt, das Startkriterium jedoch nicht. Die Koordinaten des Bildpunktes werden nicht im Koordinatenspeicher abgelegt.

S2: Das Markerkriterium und das Startkriterium sind erfüllt. Die Koordinaten des Bildpunktes werden im Koordinatenspeicher abgelegt.

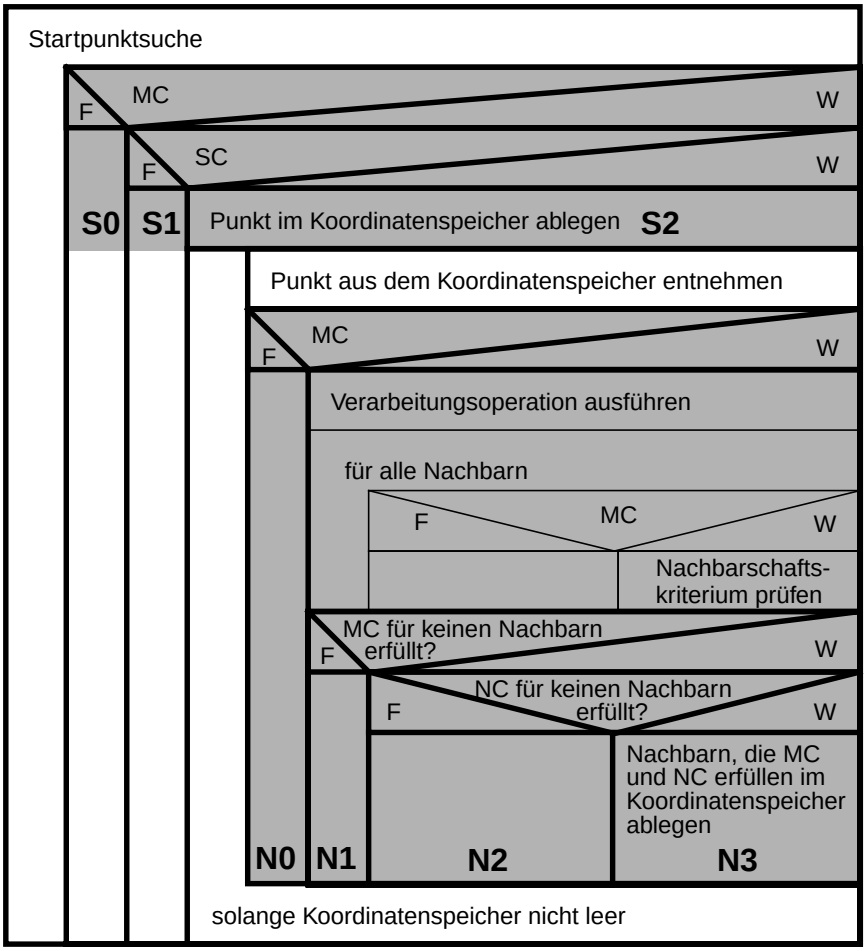
N0: Das Markerkriterium für den Zentralpunkt ist nicht erfüllt. Die Nachbarschaftskriteriumsüberprüfung kann abgebrochen werden. Es werden keine Punkte im Koordinatenspeicher abgelegt.

N1: Das Markerkriterium ist für den Zentralpunkt erfüllt, für die Nachbarpunkte jedoch nicht. Es werden keine Punkte im Koordinatenspeicher abgelegt.

N2: Das Markerkriterium ist für den Zentralpunkt und mindestens einen Nachbarpunkt

erfüllt. Allerdings ist das Nachbarschaftskriterium für keinen der Nachbarpunkte erfüllt. Es werden keine Punkte im Koordinatenspeicher abgelegt.

N3: Das Markerkriterium ist für den Zentralpunkt und mindestens einen Nachbarpunkt erfüllt. Für einen oder mehrere der Nachbarpunkte ist auch das Nachbarschaftskriterium erfüllt. Die Koordinaten dieser Punkte werden im Koordinatenspeicher abgelegt.



Abkürzungen: **MC: Marker Prüfung** **W: Wahr**
NC: Nachbarschaftskriterium **F: Falsch**
SC: Startkriterium **S0..S2, N0..N3 siehe Text**

Bild 5.15: Kopplung der Operationsausführung bei Segment-Adressierung im Fall des Non-Collected-Modus mit PQO-Verarbeitungsmodus

Neben den Bildpunktoperationen nehmen auch andere Ereignisse Einfluß auf die Verarbeitungsgeschwindigkeit. Wird z.B. der letzte Bildpunkt aus dem Koordinatenspeicher entnommen, können während dessen Verarbeitung nicht schon die Daten für den nächsten Punkt geladen werden. Die laufende Verarbeitung kann jedoch zum Ergebnis haben, daß ein gültiger Nachbar gefunden wird, dessen Koordinaten im Koordinatenspeicher abgelegt werden. Wenn dieser weiter verarbeitet wird, entsteht dadurch eine zusätzliche Verzögerung, da keine überlappende Operation möglich ist. Um diesen Effekt simulieren zu können, ist es notwendig die Anzahl der gültigen Nachbarpunkte in der aktuelle Hierarchiestufe aufzuzeichnen.

Oftmals ist das Nachbarschaftskriterium für mehrere Punkte erfüllt. In diesen Fällen entsteht eine Anforderung zum gleichzeitigen Speichern mehrerer Punkte, was bei langsamer sequentieller Operation des Koordinatenspeichers zu unnötigen Verzögerungen führen kann. Um diesen Effekt

zu simulieren, muß zusätzlich zur Anzahl der gültigen Nachbarpunkte in der aktuellen Hierarchiestufe, die Anzahl alle gültigen Nachbarpunkte unabhängig von deren Hierarchiestufe aufgezeichnet werden.

5.5.2.3 Kopplung von Datenzugriffen

Bei Segment Adressierung erfolgen irreguläre Datenzugriffe aufgrund der irregulären datenabhängigen Ausbreitung. Um zu simulieren, ob mehrfache Zugriffe auf dieselben Eingangsdaten durch einen Pufferspeicher reduziert werden, ist es erforderlich beim Lese- und Schreibzugriffen auf Bilddaten die Adresse des Zugriffes zu kennen. Um diese in kompakter Form zu speichern werden in der Logdatei

- die Koordinate des Zentralpunktes und
- die Maske der zugegriffenen Nachbarpunkte

abgespeichert. Zusammen mit dem aufgezeichneten Modus (Nachbarschaftssystem, Bildgröße, verwendete Kanäle) lassen sich daraus die Anzahl und Adressen der durchgeführten Zugriffe ableiten.

Bei segmentindizierter Adressierung erfolgen zusätzliche Datenzugriffe zum Lesen bzw. Schreiben von Segmenteigenschaften bzw. Histogrammdaten. Von der durchgeführten Bildpunktoperation ist abhängig, welche Segmenteigenschaften zugegriffen werden, und ob sie gelesen und/oder beschrieben werden. Da auch für den Fall segmentindizierter Zugriffe untersucht werden soll, welche Möglichkeiten zur Datenpufferung bestehen, muß die jeweilige Zugriffsadresse ebenfalls festgehalten werden.

5.5.2.4 Referenzereignisse und Sektionen

Da für die statistische Auswertung eine Zuordnung der Häufigkeit von Vorgängen bzw. der Komplexität zu Teilen des High-Level-Algorithmus möglich sein soll, werden auch Ereignisse aufgezeichnet, die keine Korrespondenz zu Architekturvorgängen haben, sondern vielmehr als Referenz dienen, auf die andere Ereignisse bezogen werden können:

`NEW_FRAME_EVENT`	Beginn des nächsten Bildes
`ADDR_START_EVENT`	Beginn des nächsten Aufrufes einer Adressierungsfunktion
`START_SCAN_EVENT`	Beginn des Scan-Vorganges
`START_FLOOD_EVENT`	Beginn des Ausbreitungsprozesses
`START_HRCH_EVENT`	Beginn der Verarbeitung innerhalb einer Hierarchiestufe
`SKIP_HRCH_EVENT`	Überspringen einer Hierarchiestufe
`START_DIST_EVENT`	Beginn einer neuen Wellenfront beim Ausbreitungsprozeß

Die Referenzereignisse bilden in der gezeigten Reihenfolge eine Hierarchie, bei der `START_SCAN_EVENT` und `START_FLOOD_EVENT` bzw. `START_HRCH_EVENT` und `SKIP_HRCH_EVENT` zur selben Hierarchieebene gehören und `NEW_FRAME_EVENT` der obersten und `START_DIST_EVENT` der untersten Hierarchiestufe entsprechen. Je nach gewünschter Auflösung können andere Vorgänge auf die entsprechenden Referenzereignisse bezogen werden.

5.5.2.5 Format der Aufzeichnung

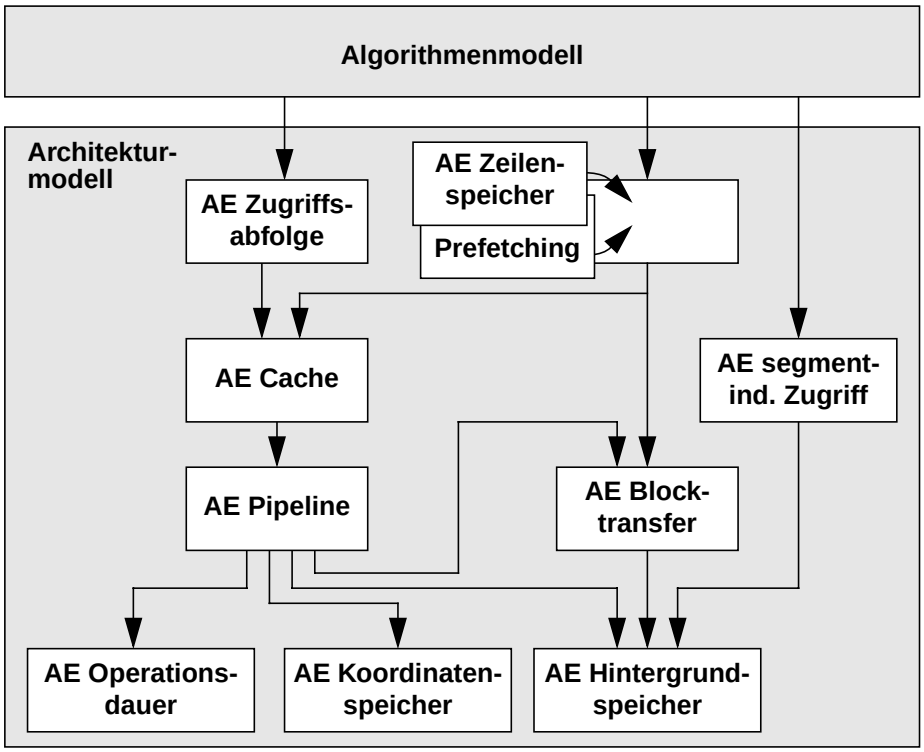
Zur statistischen Auswertung wurde die eval-Bibliothek implementiert. Sie erlaubt Ereignisse und Sektionen in einem kompakten Binärformat in Logdateien abzuspeichern, und umfaßt Funktionen für das Auslesen von Logdateien und für die automatische statistische Auswertung (Häufigkeit, Schwankung, Minimum, Maximum) im Bezug auf wählbare Referenzereignisse. Da die Aufzeichnung der Operationsausführung und der Datenzugriffe mit der eval-Bibliothek zu einer zu großen Datenmenge führen würde, erfolgt deren Aufzeichnung mit einem komprimierten Datenformat, das im Mittel 2-3 Byte pro Bildpunktoperation benötigt.

5.5.3 Architekturmodelle

Da die untersuchten Architekturvarianten ausführlich in Kapitel 6 behandelt werden, erfolgt an dieser Stelle lediglich ein kurzer Überblick über den Aufbau der Architekturmodelle. **I**n Bild 5.16 ist die Zusammenschaltung der Architekturelemente für die komplexeste Architekturvariante gezeigt, weitere Varianten werden entweder durch eine andere Wahl der Parameter oder durch eine andere Zusammensetzung der Architekturelemente (AE) erzeugt. So kann z.B. alternativ das AE Zeilenspeicher oder das AE Prefetching verwendet werden. Letzteres setzt den Einsatz des AE Cache voraus, das zur Simulation einer Architektur ohne Cache entfällt. Werden weder Cache, Prefetching noch Zeilenspeicher verwendet, entfällt das AE Blocktransfer. Das AE segmentindizierter Zugriff entfällt, wenn die Daten direkt aus dem externen Speicher gelesen werden. Insgesamt ergeben sich damit acht strukturell verschiedene Architekturmodelle. Die Simulation einer Verarbeitungseinheit mit Pipeline-Aufbau bzw. die Simulation von Abbruchkriterien bedeutet hingegen lediglich eine Änderung der Parameter des AE Pipeline.

5.5.3.1 Eingesetzte Architekturelemente

Das AE Zugriffsabfolge modelliert die Umsetzung von Zugriffen auf eine Nachbarschaft von Bildpunkten durch jeweils einen oder mehrere Speicherzugriffe mit unterschiedlichem Maß an Parallelität. Auf diese Weise wird die Speicherorganisation modelliert, die bestimmt, wieviele Bildpunkte gleichzeitig zugegriffen werden können.



AE = Architekturelement

Bild 5.16: Aufbau eines der Architekturmodelle für CONIAN

Das AE Cache modelliert einen zweidimensionalen Cache für Bilddaten. Es simuliert Cache-Zustand und ob ein Nachladen bzw. Zurückschreiben von Daten vom/in den Hintergrundspeicher erfolgt. Modellparameter sind die Anzahl der Cache-Blöcke, deren Abmessungen, die Assoziativität, das Plazierungsschema, die Art der Verdrängung und das Ersetzungsschema. Das AE segmentindizierter Zugriff beschreibt die Pufferung segmentindizierter Zugriffe. Modellparameter

sind die Puffergröße und die Einstellung, ob ein Schreibpuffer verwendet wird. Das AE Zeilenspeicher steuert bei regulärer Adressierung die Initialisierung des Zeilenspeichers sowie das Nachladen der Lesepuffer und das Zurückschreiben des Schreibpuffers entsprechend dem Fortschritt der Verarbeitung. Modellparameter sind die Puffergrößen. Das AE Prefetching modelliert das Vorladen des Caches bei regulärer Adressierung. Da es parallel zu Zugriffen auf den Cache operiert, hängt der Ablauf vom Verhältnis der Dauer der Verarbeitung zur Dauer des Ladens von Blöcken aus dem Hintergrundspeicher ab.

Diese Wechselwirkung wird aufgehoben, indem die Reihenfolge des Vorladens der Blöcke fest definiert wird und im AE Prefetching sowohl der frühestmögliche Zeitpunkt für das Vorladen (so daß der Cache-Speicher optimal genutzt wird), als auch der spätestmögliche Zeitpunkt (so daß der Block spätestens bei Benutzung zur Verfügung steht) bestimmt wird. Das AE Blocktransfer simuliert schließlich die Durchführung möglicher Transfers bei Ausnutzung der Zeit, während der der Hintergrundspeicher verfügbar ist. Wenn mehr Zeit für Zugriffe auf den Hintergrundspeicher zur Verfügung steht als durch den frühestmöglichen Zeitpunkt definiert ist, bleibt die Schnittstelle zum Hintergrundspeicher unausgelastet, wenn weniger Zeit zur Verfügung steht als durch den spätestmöglichen Zeitpunkt definiert ist, wird dem AE Pipeline mitgeteilt, daß Wartezyklen eingefügt werden müssen. Weiterhin greift auch das AE Zeilenspeicher auf das AE Blocktransfer zurück.

Das AE Pipeline beschreibt die Dauer einer Bildpunktoperation in Abhängigkeit der Adressierungsart und der Anzahl an Zugriffen, ohne den detaillierten Ablauf der überlappend ausgeführten Schritte zu simulieren. Zur Berücksichtigung der Verarbeitungsdauer, der Zugriffszeit des Hintergrundspeichers und der Dauer für Ablegen und Entnehmen von Koordinaten aus dem Koordinatenspeicher wird auf drei untergeordnete Architekturelemente zurückgegriffen.

Das AE Verarbeitungsdauer beinhaltet eine Tabelle, in der für jeden Operationstyp die Verarbeitungsdauer gespeichert ist. Das AE Hintergrundspeicher bestimmt für eine zu transferierende Datenmenge die Dauer des Zugriffes auf den Hintergrundspeicher, und berücksichtigt die Dauer zum Öffnen einer Speicherseite, falls dynamischer Speicher modelliert wird. Das AE Koordinatenspeicher bestimmt anhand der Anzahl der abzuspeichernden und zu entnehmenden Punkte die jeweilige Dauer. Es simuliert den Füllstand der aktuellen Hierarchiestufe, und gestattet so die Unterscheidung zwischen Entnahme aus dem Speicher und dem Durchreichen von Koordinaten, wenn der Koordinatenspeicher leer ist.

5.5.3.2 Auswertung der Architektursimulation

Als Ergebnis der Architektursimulation wird die Nutzung der Architekturelemente pro Bildpunktoperation ausgegeben, d.h. die Dauer in Takten, und die Art und Anzahl an Speicherzugriffen aufgeschlüsselt nach regulärer und irregulärer Adressierung. Mit einem Programm zur Auswertung lassen sich diese Daten für einen Satz bestimmter Bildpunktoperationen aufsummieren, außerdem werden dabei Standardabweichung, Minimum und Maximum bestimmt. Mit anderen Auswertewerkzeugen läßt sich für eine bestimmte Verarbeitungsoperation bzw. für einen bestimmten Bereich verarbeiteter Bilder die Liste der betroffenen Bildpunktoperationen ermitteln, so daß die Aufsummierung auf diese Bildpunktoperationen eingegrenzt werden kann. Weitere übergeordnete Auswertewerkzeuge gestatten die automatische Aufbereitung der Simulationsergebnisse wie z.B. die Zusammenstellung der Operationsdauer bzw. Hitrate aufgeschlüsselt nach Adressierungsart und Art der Verarbeitungsoperation. Mit einem anderen Programm, lassen sich schließlich die Architekturparameter verändern und Simulationen starten, so daß der Vergleich von Architekturmodellen und Architekturparametern automatisiert werden kann. Die Ergebnisse der Simulationen finden sich im folgenden Kapitel.

6. Architekturentwurf

Das vorliegende Kapitel gliedert sich in zwei Teile: In Kapitel 6.1 werden die Ergebnisse von mit der Methodik aus Kapitel 5 angestellten Architekturuntersuchungen dargelegt. In Kapitel 6.1.1 wird auf den Operationsablauf, und auf die Balancierung von Verarbeitungsdauer zu Dauer der Speicherzugriffe eingegangen. Die Untersuchung der Speicheranbindung in Kapitel 6.1.2, die sich als essentiell für eine effiziente Architektur erwiesen hat, stellt den Kern der Untersuchungen dar. Für Bilddaten und für mit Segmenten assoziierte Daten wird neben der Speicherorganisation insbesondere der Einsatz von Pufferspeichern diskutiert, die durch den Aufbau einer Speicherhierarchie für eine effiziente Datenzufuhr sorgen. Weiterhin werden die Anforderungen hinsichtlich des Zeitverhaltens und der Größe des für Ausbreitungsprozesse erforderlichen Koordinatenspeichers dargestellt, und abschließend wird ein Überblick über die mit verschiedenen Speicherarchitekturen erzielbare Verarbeitungsleistung beim Einsatz statischer bzw. dynamischer Speichertechnologie gegeben.

Aufgrund der Ergebnisse der angestellten Untersuchungen wurde das in Kapitel 4 erarbeitete Architekturkonzept in eine implementierbare Architektur umgesetzt. Der Aufbau dieser Architektur und deren Implementierung ist in Kapitel 6.2 beschrieben. Im Kapitel 6.2.1 wird kurz auf die Steuereinheit eingegangen, in Kapitel 6.2.2 wird die Verarbeitungseinheit und in Kapitel 6.2.3 die Speicheranbindung dargestellt.

6.1 Ergebnisse der Architekturuntersuchung

Zur Beurteilung verschiedener Architekturalternativen wurden deren Auswirkungen für den in [HeMo99b] beschriebenen Farbsegmentierungsalgorithmus simuliert. Als Referenz wird eine Architektur mit einer einfachen Speicherschnittstelle verwendet, bei der sich Bilddaten und segmentindizierte Daten in einem gemeinsamen externen Speicher befinden und direkt zugegriffen werden. Aus diesem statischen Speicher können mit einem Zugriff alle Komponenten eines Bildpunktes, ein segmentindizierter Datensatz bzw. ein Histogrammwert geladen werden. Der Koordinatenspeicher ist in der Referenzarchitektur als separater Speicher ausgeführt, und kann parallel zum anderen Speicher zugegriffen werden. Für die Untersuchung wird eine Systemtaktfrequenz von 200 MHz zugrundegelegt, so daß für statischen Speicher die typische Zugriffsdauer von 10ns [Sam99] zwei Takte erfordert. Im Folgenden werden die Architekturalternativen mit der Referenzarchitektur verglichen, und es wird aufgezeigt, wie - unter Berücksichtigung des nötigen Aufwandes und der erzielbaren Verarbeitungsleistung - die Merkmale einer optimierten Architektur herausgearbeitet wurden.

6.1.1 Operationsablauf

Eine Bildpunktoperation setzt sich aus mehreren Teiloperationen zusammen. Je nach Art der Operationen unterscheidet sich die Zusammensetzung. Insgesamt gibt es folgende Teiloperationen:

- Adressbestimmung
- Laden von Bildpunkten
- Laden von segmentindizierten Daten
- Verarbeitung
- Speichern des Verarbeitungsergebnisses für den Zentralpunkt
- Speichern segmentindizierter Daten
- Speichern von Nachbarpunkten

Bild 6.1 zeigt den Ablauf eines Verarbeitungszyklus für einen Bildpunkt, beispielhaft für Intra-Adressierung mit CON8 Nachbarschaft, d.h. pro Bildpunktoperation erfolgen 3 Lesezugriffe und 1 Schreibzugriff, bei einer Verarbeitungsdauer von 5 Takten. Bild B.1 im Anhang zeigt einen Verarbeitungszyklus bei Segment-Adressierung, der weitere Teiloperationen umfaßt.

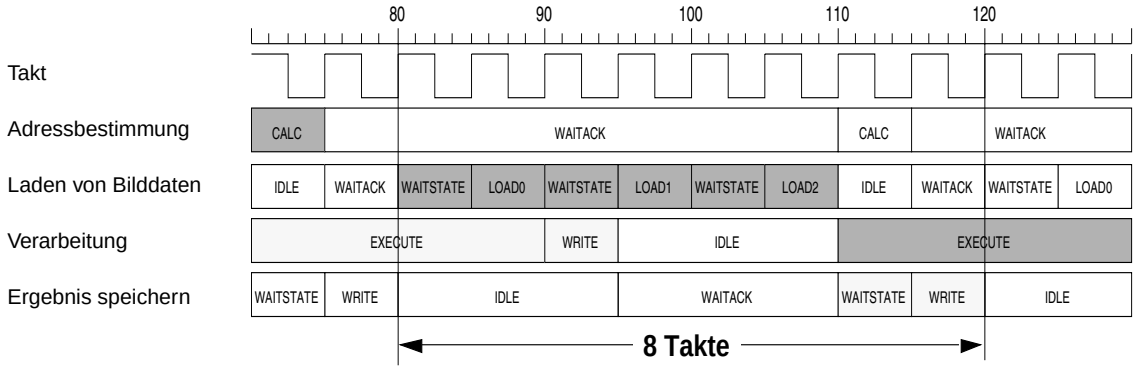


Bild 6.1: Überlappender Ablauf der Teiloperationen einer Bildpunktoperation

Die Steuerung erfolgt so, daß die Teiloperationen soweit wie möglich überlappend ausgeführt werden. Die Teiloperationen benutzen mehrere Ressourcen gemeinsam, wie z.B. die Registermatrix, der Koordinatenspeicher bzw. den Speicher in dem Bilddaten und segmentindizierte Daten abgelegt sind. Eine überlappende Durchführung von Teiloperationen ist möglich, solange unterschiedliche Ressourcen benutzt werden, bei einem Ressourcenkonflikt muß eine der Teiloperationen bevorzugt werden, während die andere verzögert wird. Für eine effiziente Operation des Gesamtsystems ist eine zeitliche Balancierung der Teiloperationen, die den Stufen einer Pipeline entsprechen, erforderlich. Da einerseits die Teiloperation „Verarbeitung“ relativ lange dauert und da sich andererseits die Teiloperationen, die auf den Speicher zugreifen, wegen gemeinsamer Ressourcennutzung gegenseitig ausschließen und insgesamt ebenfalls lange dauern, entspricht dies der Balancierung zwischen Adressierung und Verarbeitung.

Um zu untersuchen, ob ein ausgeglichenes Verhältnis von Adressierung und Verarbeitung besteht, ist es notwendig die Parameter der Bildpunktoperationen, wie Anzahl der zu ladenden Bildpunkte, Verarbeitungsdauer, Verwendung segmentindizierter Daten, etc. zu kennen. Während sich die meisten Parameter direkt aufgrund der jeweiligen Funktion angeben lassen, ist dies bei der Verarbeitungsdauer nicht möglich. Daher wurde in [Jan99] die Dauer verschiedener Bildpunktoperationen abgeschätzt, beruhend auf der Bestimmung des kritischen Pfades bei dedizierter Implementierung. Um die Operationsdauer für eine konfigurierbare Verarbeitungseinheit zu erhalten, wurde für die Funktionseinheiten eine prozentual höhere Durchlaufzeit angenommen, deren Verhältnis zur Durchlaufzeit bei dedizierter Implementierung anhand von [Thi98] bestimmt wurde. Die in dieser Arbeit angegebenen Zahlen beruhen hingegen auf den Ergebnissen einer statischen Analyse des Zeitverhaltens des VHDL-Modells der Verarbeitungseinheit [Lip00].

Da die Verarbeitung durch Multizyklus-Operationen erfolgt (siehe Kapitel 4.4), ergibt sich je nach Durchlaufzeit eine unterschiedliche Anzahl an Takten für die Verarbeitung. Die Werte hierfür und die Anzahl an Takten für eine Bildpunktoperation sind in Tabelle C.1 im Anhang gezeigt. In dem in Bild 6.1 gezeigten Fall ist die Verarbeitungseinheit zu 62,5% ausgelastet, während die Speicherschnittstelle durchgehend ausgelastet ist. Die Verarbeitungsdauer ist mit typischerweise 2 bis 5 Takten oftmals kürzer, während in der Regel gleich viele oder mehr Speicherzugriffe nötig sind. In Bild 6.2 ist die Auslastung der Verarbeitungseinheit bei verschiedenen Operationen gezeigt. Da die Auslastung der Speicherschnittstelle dabei jeweils 100% beträgt, stellt diese die kritische Ressource dar.

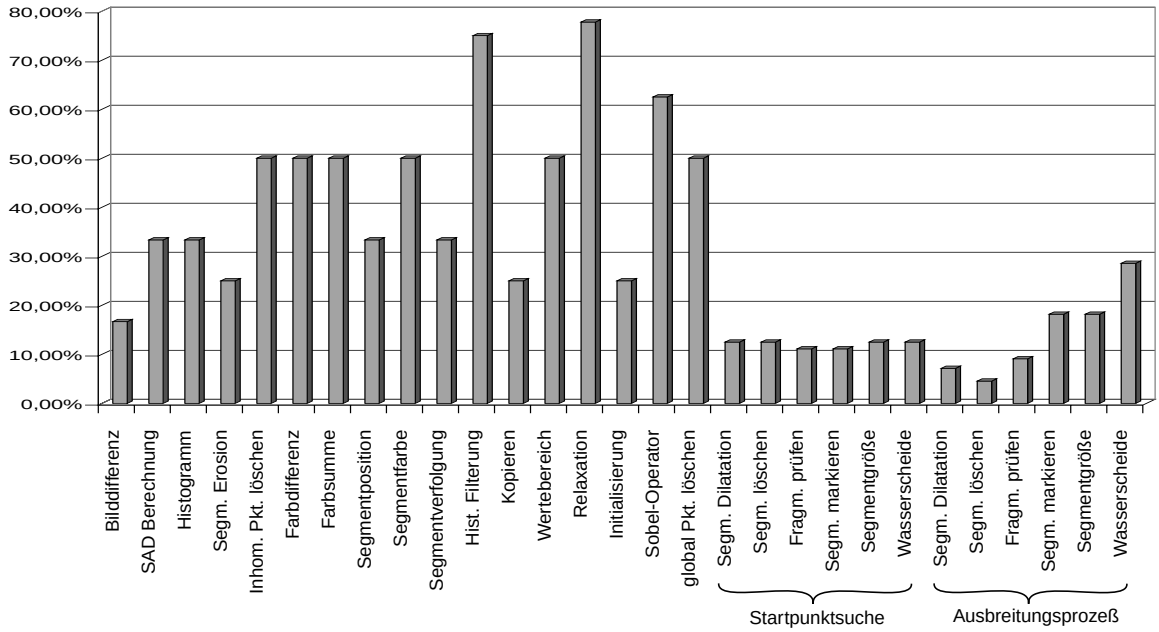


Bild 6.2: Auslastung der Verarbeitungseinheit

6.1.1.1 Pipelining der Verarbeitung

Da bei der Referenzarchitektur die Verarbeitungseinheit nicht die kritische Ressource darstellt, bringt deren Aufteilung in Pipelinestufen keine Verbesserung. Ein Vorteil durch Pipelining läßt sich jedoch erzielen, wenn die Speicherschnittstelle schnell ist. Zur Abschätzung der maximal möglichen Geschwindigkeitssteigerung wurde die Verarbeitungsdauer bei schnellstmöglicher Speicherschnittstelle verglichen (d.h. alle Zugriffe auf Bilddaten und segmentindizierte Daten dauern einen Takt). Wenn ein Pipelining in dem Maße erfolgt, daß wiederum die Speicherschnittstelle die Geschwindigkeit bestimmt, ergab sich dadurch ein Geschwindigkeitszuwachs von maximal 12,6%. Da Register verhältnismäßig flächenaufwendig sind und eine Aufteilung in Pipelinestufen die Flexibilität der Verarbeitungseinheit einschränkt, stehen dem erzielbaren Geschwindigkeitsvorteil jedoch Nachteile gegenüber. Außerdem ist Pipelining der Verarbeitungseinheit auf nicht-rekursive Operationen beschränkt. Bei rekursiver Verarbeitung, wie sie typischerweise bei Segment-Adressierung und bei segmentindizierter Adressierung stattfindet, wird eine überlappende Operation durch die sofortige Nutzung des Verarbeitungsergebnisses verhindert.

6.1.1.2 Segment-Adressierung

In Abhängigkeit der Eingangsdaten haben Operationen mit Segment-Adressierung einen unterschiedlichen Ablauf (siehe Kapitel 5.5.2). Wenn die Steuerung unabhängig davon erfolgt, muß sie sich am langsamsten möglichen Ablauf orientieren. Bei Nichterfüllung des Markerkriteriums ist der tatsächliche Ablauf jedoch kürzer, da für dessen Überprüfung nur wenig Zeit benötigt und kein Ergebnis gespeichert wird. Tabelle 6.1 zeigt am Beispiel der Operation „Segment markieren“, welche Teiloperationen in Abhängigkeit des durchlaufenen Zweiges ausgeführt werden müssen. Die Differenzierung nach Zweigen bewirkt eine Geschwindigkeitssteigerung von 10,4% bei Segment-Adressierung, das entspricht einer Gesamtbeschleunigung von 3,5%.

Bei der Prüfung von Start- und Nachbarschaftskriterium wird der Wert der Marker-Komponente des Zentralpunktes geprüft. Nur wenn dieses Markerkriterium erfüllt ist, kann auch das Start- bzw. Nachbarschaftskriterium erfüllt sein, andernfalls kann die Überprüfung des Start- bzw. Nachbarschaftskriteriums abgebrochen werden, so daß weder Nachbarschaft noch Zentralpunkt

geladen zu werden brauchen. Das Markerkriterium kann innerhalb kurzer Zeit geprüft werden, jedoch muß dies direkt nach Vorliegen des Zentralpunkts in der Registermatrix geschehen und außerdem muß die Steuereinheit die Möglichkeit bieten, den Ablauf einer Bildpunktoperation abzubrechen. Die Anzahl der Takte für Segment-Adressierung sinkt durch die vorgezogene Überprüfung des Markerkriteriums um 42,9%, insgesamt nimmt die Geschwindigkeit um 11,9% zu.

Zweig[a]	Bilddaten laden	SI-Daten laden	Dauer der Verarb.	SI-Daten speichern	Bilddaten speichern	Dauer der Bildpunktopr.
S0	4		1 Takt			18 Takte
S1	4	4	1 Takt	4		20 Takte
S2	4	4	1 Takt	4	4	22 Takte
N0	4		1 Takt			18 Takte
N1	4		4 Takte		4	20 Takte
N2	4		4 Takte		4	20 Takte
N3	4		4 Takte		4	22..38 Takte

a. siehe Bild 5.15

Tabelle 6.1: Dauer der Operation „Segment markieren“ je nach durchlaufenem Zweig

Die Segment-Adressierung unterteilt sich in Startpunktsuche und Ausbreitungsprozeß. Während für den Ausbreitungsprozeß eine Überprüfung der Nachbarschaft notwendig ist und daher typischerweise der Nachbarschaftsmodus CON4 oder CON8 verwendet wird, wird bei der Startpunktsuche in der Regel nur der Zentralpunkt, d.h. Nachbarschaftsmodus CON0 benötigt. Aus diesem Grund ist es zweckmäßig getrennte Nachbarschaftsmodi für Startpunktsuche und Ausbreitungsprozeß zu verwenden, so daß bei der Startpunktsuche weniger Ladeoperationen erforderlich sind. Dies gilt jedoch nur für Bildpunktoperationen, die im PQO-Modus ausgeführt werden, andernfalls ist für die Startkriteriumsprüfung in der Regel auch der Nachbarschaftsmodus CON4 oder CON8 erforderlich. Bezogen auf Operationen mit Segment-Adressierung beträgt die Beschleunigung 84,1%, bezogen auf alle Operationen 19,3%.

Durch Kombination von gesonderter Überprüfung des Markerkriteriums mit getrennten Nachbarschaftssysteme für Startpunktsuche und Ausbreitungsprozeß läßt sich lediglich eine geringfügig größere Geschwindigkeitssteigerung erreichen. Bezogen auf alle Operationen mit Segment-Adressierung beträgt diese 84,6%, bezogen auf alle Operationen 19,4%. Dies liegt daran, daß beide Optimierungen auf die Reduzierung der zu ladenden Bildpunkte während der Startpunktsuche zielen. Der Einsatz unterschiedlicher Nachbarschaftssysteme ist gegenüber dem Abbruch bei Nichterfüllung des Markerkriteriums vorzuziehen, da weder die Auslagerung der Markerprüfung aus dem Verarbeitungsteil noch eine Steuerung mit Abbruchmöglichkeit erforderlich sind.

6.1.2 Speicherschnittstelle

Die Speicherschnittstelle stellt nicht nur die kritische Ressource dar, die zu verarbeitenden Datenmengen sind auch extrem groß, so daß die Datenorganisation eine wichtige Rolle spielt. Die notwendige Speichergröße für Bilddaten hängt vom Bildformat ab und von der Anzahl gleichzeitig zu speichernder Bilder, die sich aus dem Verarbeitungsablauf im High-Level-Algorithmus ergibt. Speicher wird zur Verwaltung von Eingangsdaten, Zwischenergebnissen und Verarbeitungsresultat gebraucht, und um überlappende Operation zu ermöglichen (d.h. während der Verarbeitung werden bereits Bilder für den nächsten Verarbeitungszyklus im Hintergrund gela-

den bzw. Ergebnisse der vorherigen Verarbeitung zurückgeschrieben). Beim Bildformat 4CIF ist pro Bild ein Speicherbedarf von 2,9 MByte erforderlich, außerdem müssen segmentindizierte Daten für bis zu 2^{14} Segmente verwaltet werden. Pro Segment wird ein Datensatz zu 409 bit und ein Histogramm mit 256 Einträgen zu je 16 bit gespeichert. Weiterer Speicherbedarf entsteht durch den Koordinatenspeicher.

Die Größe des chipinternen Speicher ist zum einen durch die maximal fertigbare Chipfläche abzüglich der erforderlichen Fläche für Logik und Verdrahtung begrenzt, zum anderen ergeben sich bei großen Chipflächen hohe Kosten, bedingt durch sinkende Ausbeute. Da bereits für die Speicherung eines 4CIF Bildes ein sehr hoher Flächenbedarf von ca. 450 mm^2 für chipinternen SRAM Speicher erforderlich wäre, ist der Speicherbedarf für eine chipinterne Speicherung zu groß. Daher wurden die Möglichkeiten zur Speicherorganisation und zur Datenpufferung eingehend untersucht.

6.1.2.1 Bildspeicher

6.1.2.1.1 Konzepte für die Organisation der Bilddaten

Bei einer Bildpunktoperation wird je nach Nachbarschaftsmodus auf eine bestimmte Anzahl an Bildpunkten und je nach Operationstyp auf eine bestimmte Auswahl der Komponenten zugegriffen. Da typischerweise sowohl auf mehrere Bildpunkte als auch auf mehrere Komponenten zugegriffen wird, gibt es zwei Alternativen, die Bilddaten zu organisieren: Der Zugriff auf einen oder mehrere Bildpunkte gleichzeitig und der Zugriff auf eine oder mehrere Komponenten gleichzeitig. Im Folgenden werden beide Alternativen verglichen.

6.1.2.1.1.1 Bildpunktorientierte Speicherorganisation

Die erste Alternativen ist es, die zu einem Bildpunkt gehörigen Komponenten gemeinsam zu speichern. Bei dieser Organisation ist nur ein Zugriff zum Laden bzw. Speichern eines Bildpunktes notwendig. Die meisten Operationen benötigen hingegen mehrere Bildpunkte als Eingangsdaten, so daß bei bildpunktorientierter Speicherorganisation die Anzahl der Speicherzugriffe mit der Anzahl der als Eingangsdaten benötigten Bildpunkte steigt. Für eine schnelle Operation der Adreßeinheit muß auf mehrere Bildpunkte parallel zugegriffen werden. Da, je nach Position der Nachbarschaft, auf unterschiedliche Bildpunkte gleichzeitig zugegriffen wird, und da sich eine konfliktfreie Mengenzuordnung finden läßt, ist gemäß Kapitel 3.2.2.4 eine Verteilung der Bilddaten auf verschiedene Speicherbänke am günstigsten. Bild 6.3 zeigt eine entsprechende Anordnung.

Bei Intra-Adressierung mit CON8 Nachbarschaft müssen bei horizontalem Scan-Modus drei vertikal übereinanderliegende Bildpunkte gelesen werden. Wenn die Daten dieser Bildpunkte in drei verschiedenen Speicherbänken abgelegt sind, ist ein paralleler Zugriff möglich. Bild 6.4 (a) zeigt eine Bankzuordnung, die diese Bedingung erfüllt.

Da sich die Bankzuordnung alle 3 Zeilen wiederholt, muß der Adreßgenerator in diesem Fall modulo-3-Operationen beherrschen, die einen entsprechenden zeitlichen Aufwand bei der Adreßrechnung erfordern. Aufgrund des binären Rechensystems sind modulo-Operationen zu einer ganzzahligen Potenz der Basis zwei wesentlich schneller und einfacher zu implementieren. Wird die Anzahl der Speicherbänke soweit erhöht, daß sich eine Zweierpotenz ergibt, wiederholt sich auch die Bankzuordnung entsprechend, so daß auf diese Weise die Adreßrechnung vereinfacht werden kann. Eine auf vier Bänke erweiterte Zuordnung ist in Bild 6.4 (b) gezeigt, eine entsprechende Bankzuordnung für vertikalen Scan-Modus in Bild 6.4 (c). Bei der in Bild 6.4 (b) bzw. (c) gezeigten Bankzuordnung ist ein paralleler Zugriff nur bei horizontalem oder nur bei vertikalem Scan-Modus möglich. Werden hingegen diagonal liegende Bildpunkte den gleichen Bänken zugeordnet, wie in Bild 6.4 (d) gezeigt, können sowohl drei horizontal als auch drei vertikal benachbarte Bildpunkte gleichzeitig zugegriffen werden.

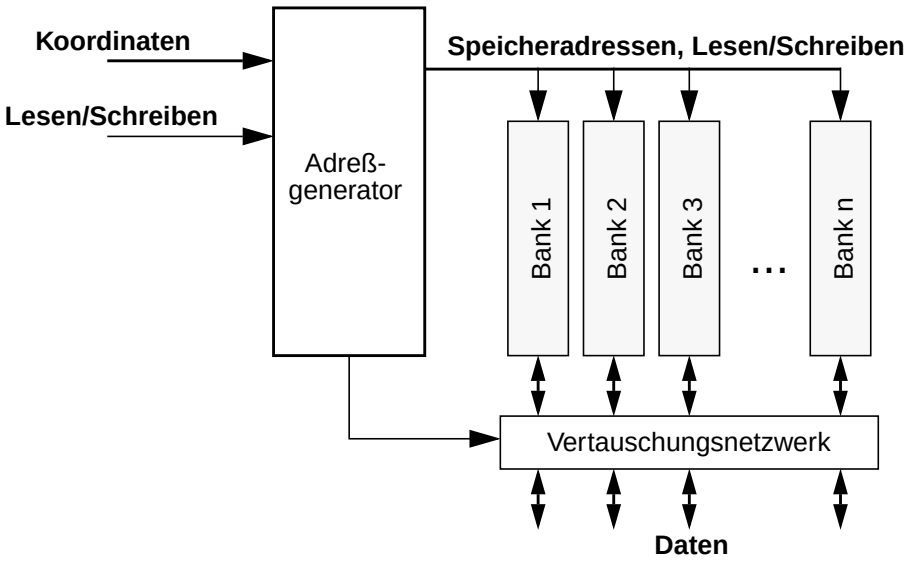


Bild 6.3: Speicheranordnung für parallelen Zugriff auf mehrere Bildpunkte

Bei Segment-Adressierung werden Nachbarschaften mit mehr als drei Bildpunkten auf einmal geladen, z.B. die CON4 Nachbarschaft bestehend aus einem Bildpunkt und den vier nächsten Nachbarn. Bild 6.4 (e) und (f) zeigen zwei mögliche Zuordnungen für die minimale Anzahl von fünf Speicherbänken, mit denen auf eine beliebig lokalisierte CON4-Nachbarschaft parallel zugegriffen werden kann.

Auch bei dieser Bankzuordnung ist die Adreßrechnung aufwendig zu implementieren und langsam, da modulo-5-Rechnung benötigt wird. Um die Adreßrechnung zu vereinfachen, muß die Anzahl der Bänke mindestens gleich der nächstgrößeren Zweierpotenz, d.h. gleich acht sein. Ein Rechteck, das aus acht Punkten besteht kann die Abmessungen 1x8, 2x4, 8x1 oder 4x2 besitzen. Eine CON4-Nachbarschaft würde jedoch immer über ein solches Rechteck hinausragen und so würden mehrere Punkte innerhalb einer CON4 Nachbarschaft zur selben Bank zugeordnet. Das kleinste Rechteck, das eine CON4-Nachbarschaft aufnehmen kann, und dessen Höhe und Breite Zweierpotenzen darstellen, hat die Abmessungen 4x4. Innerhalb dieses Rechtecks können, wie in Bild 6.4 (g) gezeigt, die Bildpunkte den acht Speicherbänken so zugeordnet werden, daß bei einem Zugriff mit beliebiger Position eine CON4-Nachbarschaft parallel geladen werden kann.

Bei der Bankzuordnung (g) können jedoch aus der Banknummer des Zentralpunkts die Banknummern der Nachbarpunkte innerhalb der CON4 nicht ermittelt werden, da keine eindeutige Zuordnung gegeben ist. Dadurch wird die Operation des Vertauschungsnetzwerks unnötig kompliziert, denn durch die in Bild 6.4 (h) gezeigte Bankzuordnung kann die Zahl möglicher Verbindungsarten des Vertauschungsnetzwerks halbiert werden. Wie in Bild 6.4 (h) gezeigt, gibt es bei dieser Bankzuordnung Blöcke vom „linkem“ und „rechtem“ Typ.

Die gleiche Problematik, die bei den bisher vorgestellten Zuordnungsschemata aufgetreten ist, entsteht auch bei parallelem Zugriff auf eine CON8-Nachbarschaft. Um alle neun Punkte innerhalb der 3x3 Nachbarschaft parallel laden zu können, müssen sie auf neun Speicherbänke verteilt werden. Für die in Bild 6.4 (i) gezeigte Bankzuordnung reichen 9 Speicherbänke aus, jedoch ist zur Berechnung der Bankzugehörigkeit eine modulo-3-Rechnung notwendig.

Zur Vereinfachung der Adreßrechnung muß, wie bei der CON4-Nachbarschaft, das Rechteck, das der Bankzuordnung zugrundeliegt, ein 4x4 Quadrat sein, allerdings ist hier eine Verteilung der Bildpunkte auf 16 Speicherbänke erforderlich (siehe Bild 6.4 (j)). Die Bankzugehörigkeit braucht in diesem Fall nicht errechnet werden, sondern ergibt sich direkt aus den zwei niederwertigsten Bits der x- und der y-Koordinaten.

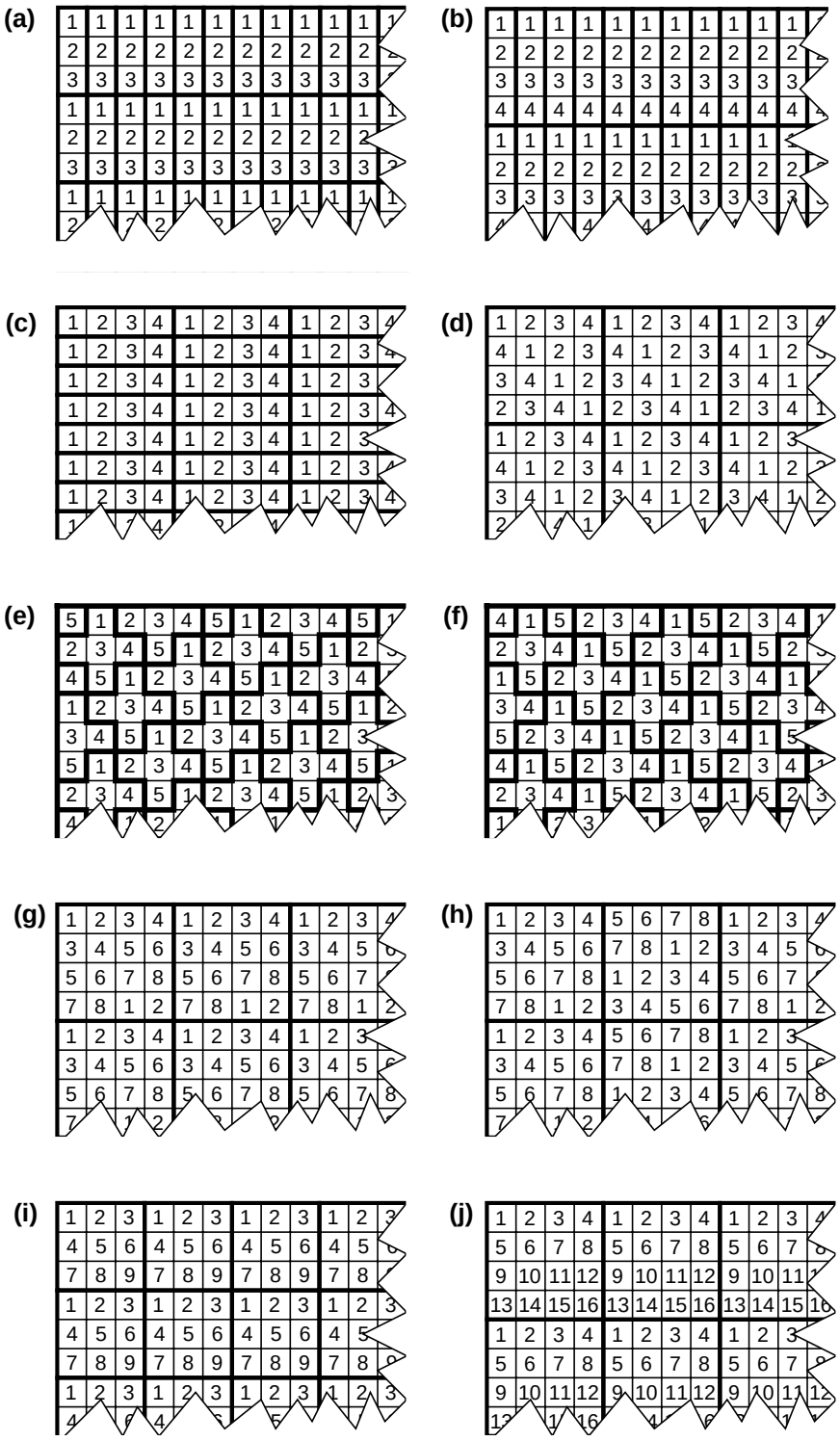


Bild 6.4: Bankzugehörigkeit für parallelen Zugriff auf
(a) drei vertikal benachbarte Bildpunkte (b) vier vertikal benachbarte Bildpunkte
(c) vier horizontal benachbarte Bildpunkte (d) vier horiz. oder vert. benachbarte Bildpunkte
(e,f) CON4 Nachbarschaft (g,h) CON4 Nachbarschaft mit vereinfachter Adressierung
(i) CON8 Nachbarschaft (j) CON8 Nachbarschaft mit vereinfachter Adressierung

6.1.2.1.1.2 Komponentenorientiere Speicherorganisation

Beim parallelen Laden mehrerer Bildpunkte ergibt sich eine große Datenbusbreite, nur wenige der Operationen benötigen jedoch alle sechs Komponenten als Eingangsdaten. Daher wurde untersucht, ob die Datenbusbreite durch gezieltes Laden ausschließlich der benötigten Komponenten gesenkt werden kann.

Die Alternative zur bildpunktorientierten Speicherorganisation ist, jede Komponente einzeln zu speichern und zuzugreifen. Neben beiden Extremfällen besteht auch die Möglichkeit zur Bündelung von Komponenten. Dabei wird nicht auf eine einzelne Komponente zugegriffen, sondern auf das entsprechende Bündel. Aufgrund der häufig zusammen auftretenden Zugriffe auf die Komponenten Y, U und V, bietet sich an, diese zu einem Bündel zusammenzufassen. Gleiches gilt für die Komponenten A, AX und M.

Werden die Komponentenbündel alternierend im Speicher abgelegt, wird dieser nicht optimal genutzt, da zwischen den Bitbreiten der Komponentenbündel eine Differenz besteht. In Bild 6.5 (a) ist dieser ungenutzte Speicherplatz grau hinterlegt, der bei obiger Aufteilung 10 bit pro Bildpunkt umfaßt. Ungenutzter Speicher wird vermieden, wenn die Kanalbündel, wie in Bild 6.5 (b) gezeigt, in verschiedenen Speichern abgelegt werden.

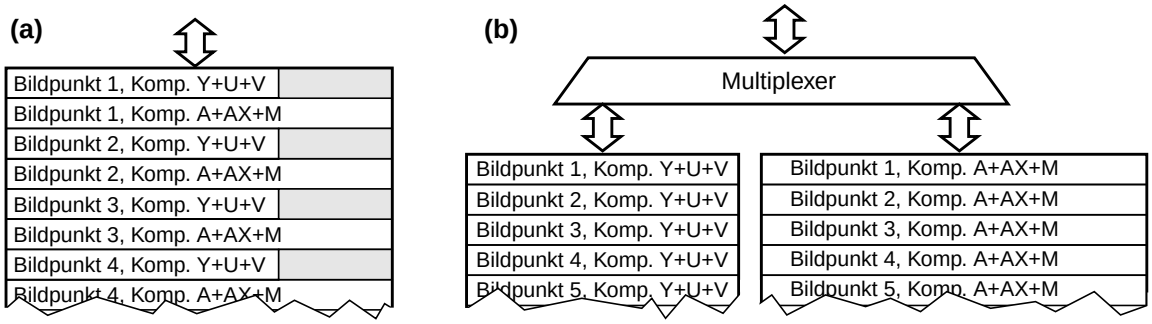


Bild 6.5: Anordnung der Komponenten im Speicher
(a) Alternierende Anordnung (b) Verteilung auf zwei Speicher

6.1.2.1.1.3 Vergleich der Speicherorganisationsformen

Je nach Anzahl parallel ladbarer Bildpunkte und Komponenten steigt oder sinkt die Daten- und Adreßbusbreite. Die möglichen Kombinationen beider Zugriffskonzepte sind in Tabelle 6.2 dargestellt. Im Interesse kostengünstiger Herstellbarkeit werden nur Varianten betrachtet, bei denen maximal 200 Pins für die Speicheranbindung erforderlich sind. Daher können im Fall eines externen Speichers nur die oberhalb der doppelten Linie gezeigten Kombinationen verwendet werden, chipintern hingegen können alle Kombinationen verwendet werden.

Um zu ermitteln welche der Kombinationen am günstigsten ist, wird die Anzahl der Speicherzugriffe bei einer Architektur mit entsprechender Speicherschnittstelle betrachtet. In Tabelle D.1 und Tabelle D.2 im Anhang sind die Bildpunktoperationen nach Art und Anzahl der zu ladenden Komponenten und Bildpunkte differenziert. Entsprechend ergibt sich eine unterschiedliche Anzahl an Speicherzugriffen pro Bildpunktoperationen, die mit deren jeweiliger Häufigkeit gewichtet wird. Tabelle 6.3 zeigt die relative Häufigkeit der Speicherzugriffe. Im Falle der schnellsten Schnittstelle (am weitesten rechts) erfolgen pro Bildpunktoperation maximal zwei Speicherzugriffe, da in einem Zugriff alle nötigen Daten geladen und in einem zweiten Zugriff das Ergebnis gespeichert wird. Da manche Operationen keine Bilddaten als Ergebnis liefern, ergibt sich eine gewichtete Häufigkeit von 190 (anstelle von 200).

parallel ladbar	1 Komponente (Y / U / V / A / AX / M)	3 Komponenten (Y+U+V / A+AX+M)	6 Komponenten (Y+U+V+A+AX+M)
1 Bildpunkt (CON0)	43 (16 / 27)	59 (34 / 25)	82 (58 / 24)
3 Bildpunkte (CONH2/V2)	123 (48 / 75)	175 (102 / 73)	246 (174 / 72)
5 Bildpunkte (CON4)	203 (80 / 123)	291 (170 / 121)	410 (290 / 120)
9 Bildpunkte (CON8)	363 (144 / 219)	523 (306 / 217)	738 (522 / 216)

Tabelle 6.2: Busbreite für bildpunkt- bzw. komponentenorientierten Zugriff Gesamtbusbreite (Daten- / Adreßbusbreite)

Die grau hinterlegten Spalten markieren Varianten, bei denen trotz größerer Busbreite mehr Speicherzugriffe notwendig sind, so daß sie nicht weiter betrachtet werden brauchen. Unter diesen Varianten befinden sich in erster Linie solche, die Bilddaten über mehrere Kanäle laden. Es zeigte sich, daß die komponentenorientierte Speicherorganisation keinen wesentlichen Vorteil bietet. Lediglich die Bündelung dreier Komponenten bei parallelem Zugriff auf drei Bildpunkte stellt eine mögliche Alternative dar. Angesichts der Tatsache, daß bei einer leichten Verringerung der Zugriffe die Busbreite gegenüber bildpunktorientierter Speicherorganisation mehr als verdoppelt wird und eine größere Anzahl an Speicherbänken erforderlich ist, wird im Folgenden die bildpunktorientierte Speicherorganisation weiterverfolgt.

Komponenten	1	3	6	1	3	6	1	3	6	1	3	6
Bildpunkte	1	1	1	3	3	3	5	5	5	9	9	9
Lesezugriffe	791	396	311	413	211	149	357	183	122	310	160	100
Schreibzugriffe	278	137	90	278	137	90	278	137	90	278	137	90
Summe	**1069**	**533**	**401**	**691**	**348**	**239**	**635**	**320**	**212**	**588**	**297**	**190**
Busbreite	43	59	82	123	175	246	203	291	410	363	523	738

Tabelle 6.3: Relative Häufigkeit von Lese- und Schreibzugriffen

6.1.2.1.2 Konzepte zur Pufferung von Bilddaten

In Kapitel 6.1.1 wurde dargelegt, daß die Speicherschnittstelle die kritische Ressource im Bezug auf das Gesamtsystem ist. Da der parallele Zugriff auf alle Komponenten und Bildpunkte im Falle externer Speicher eine zu hohe Pinzahl erfordert, und die Zugriffsgeschwindigkeit interner Speicher höher als die externer Speicher ist, wäre die interne Speicherung der Bilddaten ideal. Wie zu Beginn von Kapitel 6.1.2 dargelegt, reicht dazu der Platz auf dem Chip jedoch nicht aus.

Da die meisten Operationen mehrere Bildpunkte als Eingangsdaten benutzen, werden die Bildpunkte im Eingangsbild mehrfach gelesen, meist fünf mal (CON4) bzw. neun mal (CON8). Im Fall der Segment-Adressierung werden Nachbarpunkte ebenfalls mehrfach geschrieben (die

Marker Komponente im PQI-Modus bzw. weitere Komponenten im PQO-Modus). Wenn Lokalität gegeben ist, können mehrfache Zugriffe durch eine Speicherhierarchie reduziert werden. D.h. es wird ein Pufferspeicher eingesetzt, der die meisten Zugriffe übernimmt, so daß nur für Daten, die sich nicht im Pufferspeicher befinden, auf den Hintergrundspeicher zugegriffen werden braucht. Da hierfür schnelle interne Speicher eingesetzt werden können und auf diese mit großer Busbreite zugegriffen werden kann, stellt sich dieser Ansatz als sehr vielversprechend dar. Daher wurde die Eignung unterschiedlicher Pufferspeicher für reguläre und irreguläre Zugriffe untersucht, und nach einer Lösung gesucht, die sich für beide Arten von Zugriffen einsetzen läßt.

6.1.2.1.2.1 Pufferspeicher für reguläre Zugriffe: Zeilenspeicher

Bei Inter- und Intra-Adressierung sowie bei der Startpunktsuche für Segment-Adressierung ist die Reihenfolge, in der die Bilddaten zugegriffen werden, durch den Scan-Modus festgelegt. Wie in Kapitel 3.2.2.3 dargestellt, können bei lokalen Bildoperationen Zeilenspeicher als Pufferspeicher eingesetzt werden.

Durch zusätzliche Lese- und Schreibpuffer können pro Speicherzugriff mehrere Bildpunkte aus dem Hintergrundspeicher geladen bzw. gespeichert werden, so daß Burst-Zugriffe unterstützt werden können. Im Fall dynamischer Speicher, die im Burst Modus schneller gelesen und beschrieben werden können als mit einzelnen Zugriffen, wird dadurch die Busbreite zum Hintergrundspeicher effizienter genutzt. Um bei schneller Operation zu ermöglichen, daß die Verarbeitung parallel zum Laden bzw. Speichern der Bilddaten stattfindet, müssen beide FIFO-Speicher, der Lese- und der Schreibpuffer zwei Ports besitzen, auf die parallel und unabhängig voneinander zugegriffen werden kann. Damit ergibt sich die in Bild 6.6 gezeigte Anordnung.

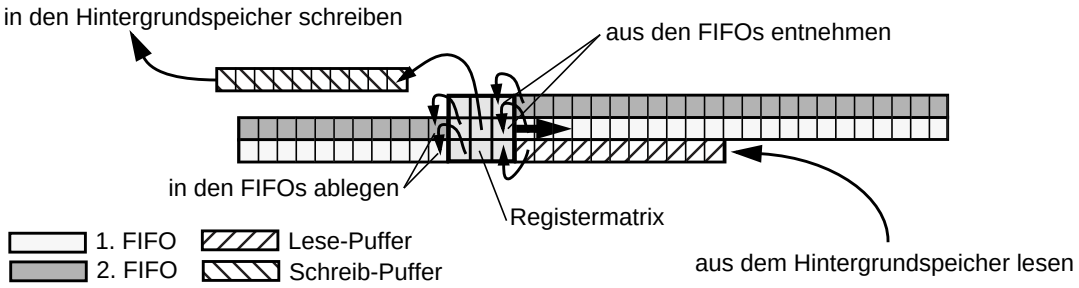


Bild 6.6: Puffer für reguläre Zugriffe: Zeilenspeicher mit Lese- und Schreibpuffer

Im Folgenden wird die Leistungsfähigkeit der Pufferung mittels Zeilenspeicher betrachtet. Aufgrund überlappender Operation sind während der Verarbeitung keine zusätzlichen Takte erforderlich. Bevor die Verarbeitung beginnen kann, muß jedoch zuerst der Lesepuffer gefüllt werden. Bei Operation im CON0-, CONHx- oder CONVx-Modus ist dies bereits ausreichend, für Operationen im CON4- oder CON8-Modus müssen zusätzlich die ersten zwei Zeilen (bzw. Spalten) des Bildes geladen werden. Nach Ende der Verarbeitung muß außerdem der Schreibpuffer geleert werden.

Bei dynamischem Speicher setzt sich die zusätzlich erforderliche Zeit t_{OH}, aus der Zeit zum Öffnen einer Speicherseite t_{page}, der Pipeline-Verzögerung t_{pipe} bis die Daten am Ausgang des Speichers anliegen, der Übertragungszeit für das Laden des Lesepuffers sowie der Übertragungszeit für das Leeren des Schreibpuffers entsprechend Gleichung (6.1) zusammen. Für statische Speicher gelten dieselben Überlegungen, wenn t_{page} und t_{pipe} gleich Null gesetzt werden. t_{acc} bezeichnet die Übertragungszeit für einen Zugriff und $n_{parallel}$ gibt an, wieviele Bildpunkte gleichzeitig zugegriffen werden, n_{load} bzw. n_{store} bezeichnet die Anzahl der Punkte im Lese- bzw. Schreibpuffer.

$$t_{OH,Intra1D} = (t_{page}+t_{pipe}) + (n_{load}/n_{parallel}) \cdot t_{acc} + (n_{store}/n_{parallel}) \cdot t_{acc} \quad (6.1)$$

Bei Inter-Verarbeitung müssen zu Beginn zwei Pufferspeicher gefüllt werden, d.h. die Initialisierungszeit vor Beginn der Verarbeitung ist entsprechend Gleichung (6.2) doppelt anzusetzen.

$$t_{OH,Inter} = 2 \cdot ((t_{page}+t_{pipe}) + (n_{load}/n_{parallel}) \cdot t_{acc}) + (n_{store}/n_{parallel}) \cdot t_{acc} \quad (6.2)$$

Bei Verwendung einer CON8 bzw. einer CON4 Nachbarschaft bei Intra-Verarbeitung müssen vor Verarbeitungsbeginn neben dem Lesepuffer auch zwei Zeilen in den Zeilenspeicher geladen werden. Bei günstiger Speicherorganisation (siehe Kapitel 6.2.3.1) treten dabei je Bildzeile 6 Seitenwechsel auf, für die jeweils $t_{page} + t_{pipe}$ veranschlagt werden müssen. Damit ergibt sich:

$$t_{OH,Intra2D} = 2 \cdot ((t_{page}+t_{pipe}) + (n_{row}/n_{parallel}) \cdot t_{acc} + 6 \cdot (t_{page}+t_{pipe})) + t_{offset,Intra1D} \quad (6.3)$$

Für die Operationen „Kopieren“ und „inhomogene Punkte löschen“, die Bildausschnitte verarbeiten, und Operationen, die Histogramme filtern, ist t_{OH} gleich $t_{OH,Intra1D}$, da diese Operationen mit einer CON0 bzw. eindimensionalen Nachbarschaften arbeiten. Wenn zwei Bildpunkte gleichzeitig übertragen werden können, ergeben sich für typische Speicher [Inf99], [Sam99] die in Tabelle 6.4 angegebenen Zeiten bei QCIF-Bildformat (n_{row}=176). Dabei wurden folgende Parameter zugrundegelegt:

$$t_{page}=4,\ t_{pipe}=4,\ n_{load}=16,\ n_{store}=16,\ t_{acc}=2,\ n_{parallel}=2 \quad (6.4)$$

Ein Zeilenspeicher läßt sich jedoch nur für reguläre Adressierung nutzen, bei der die Abfolge der Zugriffe zeilen- bzw. spaltenweise erfolgt. Da auch bei Segment-Adressierung viele Speicherzugriffe erfolgen, ist auch hierfür eine Pufferung der Bilddaten erforderlich.

	Intra1D (CON0, CONHx/Vx)	**Inter (CON0)**	**Intra2D (CON4, CON8)**
SRAM	32 Takte	48 Takte	384 Takte
SDRAM	40 Takte	64 Takte	504 Takte

Tabelle 6.4: Zusätzlicher Zeitbedarf beim Einsatz eines Zeilenspeichers

6.1.2.1.2.2 Pufferspeicher für irreguläre Zugriffe: Zweidimensionaler Cache

Bei Segment-Adressierung wird auf zweidimensional organisierte Daten irregulär in datenabhängiger Abfolge zugegriffen. Da der Segmentadressierung ein Ausbreitungsprozeß zugrunde liegt, ist innerhalb des zweidimensionalen Adreßraumes ein zeitliche und örtliche Lokalität der Zugriffe gegeben, d.h. nach einem Zugriff wird häufig mit kurzem zeitlichem Abstand auf denselben Bildpunkt zugegriffen bzw. es wird ebenfalls häufig mit kurzem zeitlichem Abstand auf Nachbarpunkte zugegriffen.

Caches werden eingesetzt, um zeitliche und räumliche Lokalität zu nutzen und die Anzahl an Zugriffen auf den Hintergrundspeicher zu reduzieren. Im Falle der Segment-Adressierung kann jedoch kein konventioneller Cache eingesetzt werden, da dessen interne Organisation nicht der räumlichen Lokalität in einem zweidimensionalem Adreßraum entspricht. Ein konventioneller Cache gestattet außerdem lediglich Zugriff auf ein Datum gleichzeitig. In [Kne97] wurde das Konzept eines objektorientierten Caches entwickelt, bei dem der Cache zur Pufferung bestimmter Datenstrukturen ausgelegt, und mit lokalen Adressen angesprochen wird. Während in [Kne97] bereits auf zweidimensionale Datenstrukturen und parallelen Vektor- und Matrixzugriff eingegangen wird, wurde im Rahmen der vorliegenden Arbeit darüberhinaus untersucht, wie ein

Cache organisiert sein muß, um in Verbindung mit Segment-Adressierung effizient eingesetzt werden zu können. Das Ziel der im Folgenden dargestellten Ergebnisse ist es, durch eine geeignete Cache-Architektur die Leistungsfähigkeit der Speicherschnittstelle zu maximieren.

Blockorganisation

Um die zweidimensionale örtliche Lokalität von Zugriffen zu nutzen und um auf den Hintergrundspeicher effizient im Burst-Modus zugreifen werden Caches blockweise organisiert. Bei zweidimensionaler Lokalität der Zugriffe, muß auch die Blockorganisation zweidimensional sein. Da es bei Ausbreitungsprozessen keine Vorzugsrichtung gibt, ist eine quadratische Blockaufteilung zweckmäßig. Im Interesse schneller Zugriffe muß die Adreßrechnung einfach gehalten sein, insbesondere muß die Kantenlänge der Blöcke eine Zweierpotenz ergeben, so daß 2x2, 4x4, 8x8 und 16x16 mögliche Blockgrößen sind. Sehr große Blöcke bewirken eine lange Unterbrechung für Laden bzw. Speichern vom/zum Hintergrundspeicher und erhöhen die Wahrscheinlichkeit, daß Daten geladen werden, die später nicht gebraucht werden. Sehr kleine Blöcke nutzen dagegen die Lokalität und den Vorteil von Burst-Zugriffen nicht in ausreichendem Maß. Die Bestimmung der optimalen Blockgröße erfolgt später in diesem Kapitel durch Simulation.

Wenn zur Adressierung der Cache-Daten direkt der logische Adreßraum (d.h. die Koordinaten der gepufferten Bilder) anstelle des physikalischen Adreßraumes (d.h. die Adressen des Hintergrundspeichers) verwendet werden, ist es möglich, die für die Adreßrechnung nötige Multiplikation zu verschieben. Dadurch kann die bei jeden Zugriff durchzuführende Adressierung schnell vonstatten gehen, während die aufwendigere Multiplikation nur beim seltener stattfindenden Zugriff auf den Hintergrundspeicher durchgeführt werden muß.

Zugriffsarten

Neben der Reduzierung der Zugriffe auf den Hintergrundspeicher ist ein weiterer Grund für den Einsatz eines Caches parallelen Zugriff auf mehrere Bildpunkte zu ermöglichen. Für den parallelen Zugriff gelten die gleichen Überlegungen, die bereits in Kapitel 6.1.2.1.1.1 angestellt wurden. Die dort gezeigten Bankzuordnungen sind in x- und y-Richtung periodisch, was einer Blockeinteilung des zweidimensionalen Adreßraumes entspricht. Wenn die Adressierung der Cache-Blöcke und die Bankauswahl voneinander entkoppelt sind, müssen zwei voneinander unabhängige Adreßrechnungen durchgeführt werden. Um eine dadurch bedingte höhere Komplexität bzw. niedrigere Geschwindigkeit zu vermeiden, sollten Cache-Blockgröße und die Blockeinteilung aufgrund der Bankzuordnung miteinander verträglich d.h. ganzzahlige Vielfache sein, so daß es nur einen oder wenige Typen von Cache-Blöcken gibt.

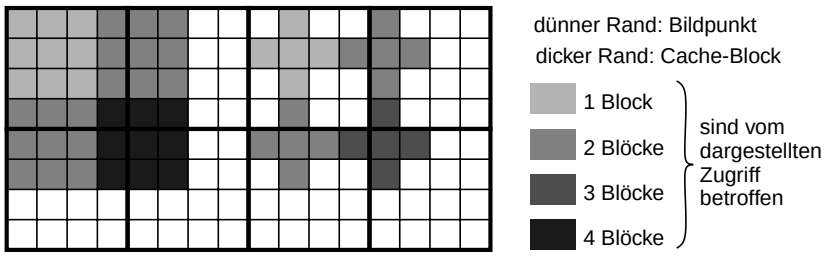


Bild 6.7: Anzahl gleichzeitig zugegriffener Cache-Blöcke bei CON4 und CON8 Nachbarschaft und bei je vier verschieden positionierten Zugriffen

Parallele Zugreifbarkeit kann entweder parallele Zugriffe innerhalb eines Blockes oder parallele Zugriffe auf Bildpunkte verschiedener Blöcke bedeuten. Wie in Bild 6.7 gezeigt, können durch den Zugriff auf eine Nachbarschaft Zugriffe auf bis zu vier Cache-Blöcke entstehen, sofern die Cache-Blockgröße nicht kleiner als 2x2 und die Größe der zugegriffenen Nachbarschaft nicht

größer als 3x3 ist. Die Anzahl der zugegriffenen Blöcke schwankt je nach Position des Zugriffes und der Form der Nachbarschaft zwischen 1 und 4.

Assoziativität

Wenn ein Cache-Block jeden beliebigen Block des Hintergrundspeichers aufnehmen kann, müssen bei jedem Zugriff alle Cache-Blöcke geprüft werden, ob sie das gewünschte Datum enthalten. Da diese Überprüfung aufwendig ist, ist es zweckmäßig einzuschränken, an welchen Stellen im Cache ein Block aus dem Hintergrundspeicher abgelegt werden kann. Bei konventionellen Caches unterscheidet man [HePa90]

- vollassoziative Caches
- direktabgebildete Caches
- Satz-assoziative Caches

Diese Konzepte lassen sich auch für zweidimensionale Daten einsetzen. Da bei vollassoziativen Caches eine sehr große Anzahl von Vergleichern oder ein Inhaltsadressierbarer Speicher benötigt werden, kommt im Interesse der Implementierbarkeit nur ein direktabgebildeter oder ein Satz-assoziativer Cache in Betracht, wobei bei letzterem zu fordern ist, daß der höhere Implementierungsaufwand einer entsprechenden Verbesserung der Hitrate gegenübersteht. Beide Varianten sind in Bild 6.8 für einen zweidimensionalen Cache illustriert.

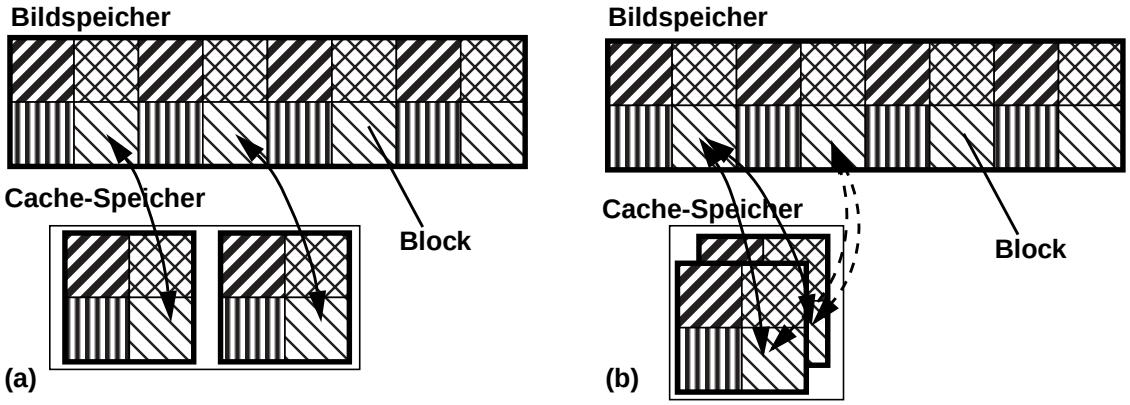


Bild 6.8: (a) Direktabgebildeter und (b) 2-fach assoziativer Cache

Plazierungsschema

Während bei konventionellen direktabgebildeten bzw. Satz-assoziativen Caches die Blöcke im Hintergrundspeichers periodisch den Cache-Blöcken zugeordnet werden, muß beim einem Satz-assoziativen Cache für Bilddaten für jeden Block im Hintergrundspeichers festgelegt werden, in welchem Cache-Block er abgelegt werden kann. Dies wird im Folgenden als Plazierungsschema bezeichnet.

Um parallele Zugreifbarkeit zu gewährleisten, müssen vier Blöcke, die alle an einem Punkt zusammenstoßen gleichzeitig im Cache gehalten werden können. Bei Ausbreitungsprozessen ist außerdem anzustreben, daß beliebig positionierte kleine bis mittlere zusammenhängende Flächen gleichzeitig im Cache gespeichert werden können. Das Plazierungsschema muß so gewählt werden, daß im Falle eines direktabgebildeten Caches alle Blöcke, die gleichzeitig im Cache gehalten werden müssen, auf verschiedene Orte im Cache abgebildet werden. Im Falle eines Satz-assoziativen Cache sind die Anforderungen schwächer, da in jedem Fall n beliebige Blöcke im Cache abgelegt werden können.

Verdrängungsstrategie

Da der Cache nur einen Teil des Bildes aufnehmen kann, müssen im Laufe der Zeit Daten wieder aus dem Cache entfernt werden, um das Speichern neuer Blöcke zu ermöglichen. Während bei direktabgebildeten Caches aufgrund des Plazierungsschemas der zu verdrängende Block festgelegt ist, muß dieser bei Satz-assoziativen und bei vollassoziativen Caches aus mehreren Möglichkeiten gewählt werden. Die bei konventionellen Caches gebräuchlichen Konzepte LRU-Verdrängung, FIFO-Verdrängung und zufällige Verdrängung [HePa90] lassen sich auf einen Cache für Bilddaten übertragen.

Bei LRU-Verdrängung (Least Recently Used) wird der Cache-Block verdrängt, auf den am längsten nicht mehr zugegriffen wurde. Die zugrundeliegende Annahme, daß häufig gefragte Daten auch in Zukunft häufig benötigt werden, trifft jedoch nicht die Charakteristik der Segment-Adressierung. Außerdem erfordert die Implementierung dieses Konzepts einen sehr hohen Verwaltungsaufwand. Bei FIFO-Verdrängung werden die Cache-Blöcke in der gleichen Reihenfolge aus dem Cache entfernt, in der sie zuvor gelesen wurden. Dieses Konzept läßt sich einfacher implementieren als LRU-Verdrängung, es erfordert jedoch - im Gegensatz zu zufälligen Verdrängung - für jeden Cache-Block einen Zähler, der anzeigt, welcher Cache-Block als nächstes verdrängt wird. Im Interesse einfacher Implementierbarkeit wird im Weiteren bei Satz-Assoziativität nur zufällige Verdrängung betrachtet.

Schreibstrategie

Um Verarbeitungsergebnisse zu speichern gibt es drei Möglichkeiten, die schematisch in Bild 6.9 gezeigt sind. Bei der „write around“-Strategie wird der Cache beim Schreiben ignoriert und die Daten unter Umgehung des Cache direkt in den Hintergrundspeicher geschrieben. Für die Unterstützung von Burst-Schreibzugriffen auf den Hintergrundspeicher, kann zusätzlich ein Schreibpuffer integriert werden. Diese Funktionalität entspricht der nicht-rekursiven Verarbeitung. Da die Daten im Cache nicht verändert werden, läßt sich rekursive Verarbeitung mit dieser Schreibstrategie nicht implementieren.

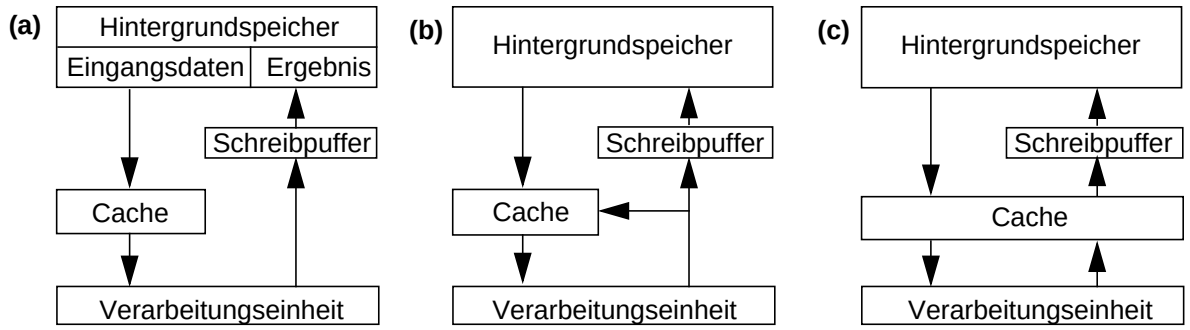


Bild 6.9: Schreibstrategien „write around“ (a) „write through“ (b) „write back“ (c)

Bei der „write through“-Strategie wird bei einem Schreibzugriff sowohl der Cache als auch der Hintergrundspeicher beschrieben damit Cache- und Hintergrundspeicher stets auf dem gleichen Stand bleiben. Somit ist es nicht erforderlich Cache-Blöcke zurückzuschreiben, falls sie verdrängt werden. Diese Strategie gestattet es Datenkonsistenz im Fall rekursive Verarbeitung zu gewährleisten, und erlaubt auch nicht-rekursive Verarbeitung durch Speicherung des Verarbeitungsergebnisses in einem anderen Adreßbereich. Um die einzelnen Schreibzugriffe zu bündeln, kann wiederum ein Schreibpuffer eingeführt werden. Nachteilig ist jedoch, daß vor dem Laden eines Cache-Blockes abgewartet werden muß, bis der Schreibpuffer vollständig geleert wurde, da bei Segment-Adressierung ansonsten nicht sichergestellt werden kann, daß der Schreibpuffer weder Daten enthält, die den zu verdrängenden Block betreffen, noch solche, die den zu ladenden

Block betreffen. Außerdem ist nachteilig, daß mehrfache Schreibzugriffe auf dasselbe Datum, wie sie für das Schreiben der Markerkomponente auftreten, nicht gepuffert werden können.

Die „write back"-Strategie, bei der Schreibzugriffe lediglich auf den Cache erfolgen, und der Inhalt eines geänderten Cache-Blockes erst dann zurückgeschrieben wird, wenn dieser verdrängt wird, eignet sich für rekursive und nicht-rekursive Verarbeitung. Sie behebt die Nachteile der „write through"-Strategie, da auch die Anzahl der Schreibzugriffe auf den Hintergrundspeicher minimiert wird. Der Einsatz eines Schreibpuffers ermöglicht es sofort mit dem Laden eines Blokkes zu beginnen, während der zu verdrängende Block zunächst parallel dazu im Schreibpuffer abgelegt wird. Auf diese Weise wird die Zeit bis zum Bereitstehen der neuen Daten verkürzt.

Cache-Simulationen

Aufgrund der vorangegangenen Überlegungen wurde die Menge sinnvoller Cache-Parameter bereits eingegrenzt. Zur Bestimmung der übrigen Parameter wurden Cache-Simulationen der Segment-Adressierung mit realen Bildsequenzen durchgeführt. Tabelle 6.5 und Tabelle 6.6 zeigen die Simulationsergebnisse für die „Foreman"-Sequenz, Bild 0 bis 20, QCIF-Format mit folgenden Parametern:

- Cache-Größe: 1k/2k/4k/8k/16k Bildpunkte (= 7,25 / 14,5 / 29 / 59 / 116 kByte)
- Blockgröße: 2x2, 4x4, 8x8, 16x16 Bildpunkte
- Assoziativität: direktabgebildet / 4-fach-assoziativ
- Platzierungsschema: Quadratischer bzw. rechteckiger (v:h=1:2) Bereich gleichzeitig
- Verdrängung: implizit / zufällig
- Schreibstrategie: write back

Cache-Größe / Blockgröße	1k Bildpunkte	2k Bildpunkte	4k Bildpunkte	8k Bildpunkte	16k Bildpunkte
2x2	64,16%	68,50%	69,43%	72,77%	74,68%
4x4	84,33%	88,09%	88,56%	91,19%	91,97%
8x8	91,64%	94,43%	94,71%	96,58%	96,92%
16x16	94,33%	96,72%	97,01%	98,29%	98,48%

Tabelle 6.5: Hit-Raten für einen direktabgebildeten Cache (implizite Verdrängung)

Cache-Größe / Blockgröße	1k Bildpunkte	2k Bildpunkte	4k Bildpunkte	8k Bildpunkte	16k Bildpunkte
2x2	64,52%	67,69%	69,70%	73,88%	76,30%
4x4	86,18%	88,40%	89,52%	92,14%	93,42%
8x8	93,07%	95,21%	95,88%	97,36%	98,01%
16x16	---	96,98%	97,83%	98,99%	99,35%

Tabelle 6.6: Hit-Raten für einen 4-fach assoziativen Cache (zufällige Verdrängung)

Wie aus Tabelle 6.5 und 6.6 ersichtlich, besitzt ein 4-fach assoziativer Cache geringfügig bessere Hit-Raten als ein ansonsten gleicher direktabgebildeter Cache. Da aber die Einflüsse der Cache-Größe und der Blockgröße deutlich stärker sind, wird der 4-fach assoziative Cache zugunsten einfacherer Implementierbarkeit des direktabgebildeten Cache nicht weiterverfolgt.

Die Trefferrate von Cache-Zugriffen hängt stark von der Cache-Größe ab. Die Cache-Größe ist allerdings mit der zu verarbeitenden Bildgröße in Relation zu setzen. In Tabelle 6.7 werden die Hit-Raten der „Foreman“-Sequenz im QCIF- und im CIF-Format bei jeweils gleicher Cache-Größe verglichen. Die Cache-Simulation wurde mit 4x4-Blöcken und direktabgebildetem Cache durchgeführt. Aus Tabelle 6.7 ist zu erkennen, daß bei Vervierfachung der Bildgröße die Größe des Caches verdoppelt bis vervierfacht werden muß, um eine vergleichbare Hit-Rate zu erzielen.

Cache-Größe **Bildgröße**	**1k Bildpunkte**	**2k Bildpunkte**	**4k Bildpunkte**	**8k Bildpunkte**	**16k Bildpunkte**
QCIF	84,33%	88,09%	88,56%	91,19%	91,97%
CIF	82,90%	86,39%	87,23%	90,63%	91,11%

Tabelle 6.7: Vergleich der Hit-Raten von QCIF- und CIF-Bildsequenzen

Da die Blockgröße die Dauer für das Nachladen bei einem Cache-Miss bestimmt, ist mit der Hit-Rate alleine noch keine Aussage über die Effizienz eines Cache möglich. Ein besseres Maß ist die mittlere Anzahl an Taktzyklen für den Lese- und Schreibzugriff während einer Bildpunktoperation, die eine Aussage über die Zugriffsgeschwindigkeit erlaubt. Im Folgenden wird die Dauer einer Bildpunktoperation mit CON4 Nachbarschaft bei Segment-Adressierung betrachtet. Die mittlere Anzahl an Takten $\bar{t}_Z$ errechnet sich zu

$$\bar{t}_Z = (1 - r_H) \cdot (t_R + t_W) + t_{CR} + t_{CW} \tag{6.5}$$

mit

- r_H: Hit-Rate
- t_R: Dauer für das Laden eines Blocks aus dem Hintergrundspeicher (in Takten)
- t_W: Dauer für das Schreiben eines Blocks in den Hintergrundspeicher (in Takten)
- t_{CR}: Dauer für das Laden einer CON4 Nachbarschaft aus dem Cache (in Takten)
- t_{CW}: Dauer für das Schreiben einer CON4 Nachbarschaft in den Cache (in Takten)

Da beim Laden eines Blocks typischerweise ein anderer Cache-Block verdrängt werden muß, ist sowohl t_R als auch t_W bei einem Cache-Miss zu berücksichtigen. Beim Laden von Daten in einen internen Cache wird angenommen, daß entsprechend der maximalen Busbreite pro Zugriff zwei Bildpunkte vom Hintergrundspeicher in den Cache übertragen werden. Im Fall eines externen Caches wird angenommen, daß pro Zugriff vier Bildpunkte übertragen werden. Bei Verwendung von dynamischen Speicher als Hintergrundspeicher ergeben sich die in Tabelle 6.8 aufgeführten Werte für t_R. Der Wert für t_W ist mit t_R identisch. Mit den Zugriffszeiten $t_{CR}=t_{CW}=1$ für internen Cache und $t_{CR}=t_{CW}=6$[1] für externen Cache errechnen sich für die mittlere Anzahl an Taktzyklen die in Tabelle 6.9 bzw. 6.10 gezeigten Werte.

1. Es wird davon ausgegangen, daß sich die 5 Bildpunkte der CON4 Nachbarschaft über einen 2 Bildpunkte breiten Bus mit 3 Zugriffen lesen bzw. schreiben lassen.

Blockgröße	interner Cache	externer Cache
2x2	40 ns + 2510 ns = 60ns 12 Takte	40 ns + 1510 ns = 50 ns 10 Takte
4x4	40 ns + 8510 ns = 120 ns 24 Takte	40 ns + 4510 ns = 80 ns 16 Takte
8x8	40 ns + 32510 ns = 360 ns 72 Takte	40 ns + 16510 ns = 200 ns 40 Takte
16x16	40 ns + 128510 ns = 1320 ns 264 Takte	40 ns + 64510 ns = 680 ns 136 Takte

Tabelle 6.8: Dauer t_R eines Blockzugriffes auf SDRAM Hintergrundspeicher

Wie aus Tabelle 6.9 und 6.10 ersichtlich steigt mit zunehmender Cache-Größe naturgemäß die Zugriffsgeschwindigkeit, der Einfluß der Cache-Größe ist jedoch geringer als der Einfluß der Blockgröße. Für die Blockgrößen 4x4 und 8x8 ergibt sich eine hohe Zugriffsgeschwindigkeit, im Falle der Blockgröße 2x2 müssen dagegen häufig Blöcke nachgeladen werden, wodurch der Zeitbedarf zum Öffnen einer Speicherseite und die Ausgangsverzögerung steigt, und mehr Wartezyklen entstehen. Die Blockgröße 16x16 erweist sich ebenfalls als langsamer.

Ohne Cache dauern die Zugriffe auf dynamischen Bildspeicher 18 Takte pro Bildpunktoperation[1], werden die Daten ohne Cache in einem externen statischen Bildspeicher verwaltet, sind 12 Takte erforderlich. Sowohl im Fall des internen Caches als auch im Fall des externen Caches läßt sich der Zugriff auf externe dynamische Speicher beschleunigen. Der externe Cache bietet allerdings gegenüber einem statischen Hintergrundspeicher keinen Geschwindigkeitsvorteil, da sowohl bei einem externen Speicher als auch bei einem externen Cache pro Zugriff maximal zwei Bildpunkte zur Registermatrix übertragen werden können. Mit internem Cache und externem SRAM-Speicher kann ein Geschwindigkeitsvorteil erzielt werden, da hier auf eine komplette Nachbarschaft parallel zugegriffen werden kann.

Bei einem Cache-Miss, muß abgewartet werden, bis ein zu verdrängender Block zurückgeschrieben wurde, bevor der angeforderte Block geladen werden kann. Durch Einsatz eines Schreibpuffers, läßt sich - wie zuvor dargelegt - das Schreiben verzögern, so daß bei einem Cache-Miss lediglich t_R abgewartet werden muß, bis mit der Verarbeitung fortgefahren werden kann. Das Zurückschreiben in den Hintergrundspeicher erfolgt im Anschluß, parallel zur Verarbeitung. Da jedoch bei kurz aufeinanderfolgenden Cache-Misses nicht sichergestellt ist, daß das Zurückschreiben bereits beendet ist, wenn ein weiterer Cache-Miss auftritt, tritt in diesen Fällen eine Verzögerung für das Zurückschreiben auf. Aufgrund der Operationshäufigkeit und der jeweiligen Verarbeitungsdauer wurde die mittleren Verarbeitungsdauer bei Segment-Adressierung zu 3,87 Takten abgeschätzt.

Unter Berücksichtigung der Hit-Rate ergeben sich die Tabelle 6.11 gezeigten Werte für die Dauer, die beim Einsatz eines Schreibpuffers für das Zurückschreiben eines Blocks in den Hintergrundspeicher zu berücksichtigen ist. Wie in Tabelle 6.12 ersichtlich, kann durch den Einsatz eines Schreibpuffers die mittlere Zugriffsdauer wesentlich verbessert werden.

1. Entsprechend [Inf99] wird angenommen, daß zwischen Lesen und Schreiben eines Datums 3 Speichertakte liegen müssen. Dies entspricht 6 Systemtakten. Weiterhin wird angenommen, daß keine Seitenwechsel auftreten.

Cache-Größe Blockgröße	1k Bildpunkte	2k Bildpunkte	4k Bildpunkte	8k Bildpunkte	16k Bildpunkte
2x2	19,17	14,52	14,45	14,18	14,03
4x4	15,13	14,38	14,29	13,76	13,61
8x8	17,68	15,79	15,60	14,33	14,09
16x16	26,74	20,53	19,77	16,45	15,95

Tabelle 6.9: Zugriffsdauer pro Bildpunktoperation für externen Cache, ohne Schreibpuffer

Cache-Größe Blockgröße	1k Bildpunkte	2k Bildpunkte	4k Bildpunkte	8k Bildpunkte	16k Bildpunkte
2x2	10,60	9,56	9,34	8,54	8,08
4x4	9,52	7,72	7,49	6,23	5,85
8x8	14,04	10,02	9,62	6,92	6,44
16x16	31,94	19,32	17,79	11,03	10,03

Tabelle 6.10: Zugriffsdauer pro Bildpunktoperation für internen Cache, ohne Schreibpuffer

Cache-Größe Blockgröße	ohne Schreib-puffer	mit Schreibpuffer				
		1k Bildpunkte	2k Bildpunkte	4k Bildpunkte	8k Bildpunkte	16k Bildpunkte
2x2	11	1,20	0,00	0,00	0,00	0,00
4x4	23	0,00	0,00	0,00	0,00	0,00
8x8	71	25,71	2,52	0,00	0,00	0,00
16x16	263	195,75	146,01	134,57	37,68	9,39

Tabelle 6.11: Mittlere Anzahl an Takten für den Schreibzugriff auf den Hintergrundspeicher

Cache-Größe Blockgröße	1k Bildpunkte	2k Bildpunkte	4k Bildpunkte	8k Bildpunkte	16k Bildpunkte
2x2	6,73	5,78	5,67	5,27	5,04
4x4	5,76	4,86	4,75	4,11	3,93
8x8	10,17	6,15	5,81	4,46	4,22
16x16	28,07	15,45	13,92	7,16	6,16

Tabelle 6.12: Zugriffsdauer pro Bildpunktoperation für internen Cache, mit Schreibpuffer

6.1.2.1.2.3 Pufferspeicher für reguläre und irreguläre Zugriffe

Das in Kapitel 6.1.2.1.2.1 vorgestellte Konzept der Datenpufferung mittels Zeilenspeicher eignet sich gut zum Einsatz bei regulärer Adressierung, d.h. Intra- und Inter-Adressierung und Startpunktsuche bei Segment-Adressierung, ist jedoch ungeeignet für die Datenpufferung bei irregulärer Adressierung, d.h. dem Ausbreitungsprozeß bei Segment-Adressierung. Andererseits eignet sich das in Kapitel 6.1.2.1.2.2 vorgestellte Konzept der Datenpufferung mittels Cache gut für die Pufferung irregulärer Zugriffe. Es kann auch für die Pufferung regulärer Zugriffe eingesetzt werden, ist dabei jedoch nicht so performant wie ein Zeilenspeicher, da Daten nicht im Hintergrund geladen werden.

Da jedoch sowohl reguläre als auch irreguläre Adressierung unterstützt werden müssen, ist ein Pufferspeicher erforderlich, der in beiden Fällen effizient arbeitet. Der gleichzeitige Einsatz von Zeilenspeicher und Cache eignet sich, wenn reguläre und irreguläre Adressierung getrennt stattfinden, nicht jedoch für Segment-Adressierung im „non-collected“ Modus, bei dem die Startpunktsuche häufig durch einen Ausbreitungsprozeß unterbrochen wird, d.h. reguläre und irreguläre Adressierung alternieren. Der Datenbestand von Zeilenspeicher und Cache müßten bei gemeinsamem Einsatz gleichzeitig auf aktuellem Stand gehalten werden, was sich jedoch nur dadurch realisieren ließe, daß bei den Unterbrechungen der Startpunktsuche der geänderte Inhalt des Zeilenspeichers in den Hintergrundspeicher zurückgeschrieben wird, bevor der Cache auf den Hintergrundspeicher zugreifen kann. Nach dem Ausbreitungsprozeß müßte wiederum der geänderte Teil des Cache in den Hintergrundspeicher zurückgeschrieben werden, bevor die Startpunktsuche fortgesetzt werden könnte.

Nachteilig für die Cache-Effizienz bei regulärer Adressierung sind die Wartezyklen, die dadurch entstehen, daß Bilddaten erst dann aus dem Hintergrundspeicher geladen werden, wenn ein Zugriff erfolgt. Da bei regulärer Verarbeitung jedoch bereits vorab bekannt ist, welche Daten benötigt werden, können Cache-Misses vermieden werden, wenn die Daten bereits geladen werden kurz bevor sie gebraucht werden. Dadurch kann das Laden im Hintergrund erfolgen, so daß keine Verzögerung der Verarbeitung auftritt. Neben dem vorausschauenden Laden kann auch vorausschauendes Zurückschreiben eingesetzt werden, so daß Laden und Speichern von Cache-Blöcken im Hintergrund erfolgen (Prefetching). Im Folgenden wird untersucht, ob sich bei regulärer Adressierung mittels Prefetching eine vergleichbare Performance wie beim Einsatz eines Zeilenspeichers erzielen läßt.

Cache mit Prefetching

Um eine gute Performance für die Pufferung irregulärer Zugriffe zu erreichen wird von dem in Kapitel 6.1.2.1.2.2 vorgestellten Cache-Konzept ausgegangen. Während bei irregulärer Adressierung von unveränderter Operation ausgegangen wird, wird im Folgenden untersucht, wie bei regulärer Adressierung durch Prefetching die Effizienz verbessert werden kann.

Während des Verarbeitungsablaufes werden die Bildpunkte in einer vorgegebenen Reihenfolge adressiert. Neben den Bildpunkten, die aktuell für die Verarbeitung benötigt werden, muß zu jedem Zeitpunkt zwischen solchen Bildpunkten unterschieden werden, die bisher noch nicht für die Verarbeitung benötigt wurden, solchen, die bereits für die Verarbeitung benutzt wurden und anschließend nochmals benötigt werden, und solchen, die nicht mehr benutzt werden. Blöcke müssen spätestens zu dem Zeitpunkt im Pufferspeicher bereitstehen, zu dem enthaltene Bildpunkte erstmals benötigt werden, um keine Verzögerungen zu verursachen. Sie müssen außerdem so lange im Cache bleiben, bis sie nicht mehr benötigt werden, da sie ansonsten unnötigerweise mehrfach geladen werden müssen.

Im Falle horizontaler zeilenweiser Verarbeitungsabfolge mit CON8 Nachbarschaft ist in Bild 6.10 die Menge der Bildpunkte, die bereits für die Verarbeitung benutzt wurden und im Folgenden nochmal benötigt werden, für zwei verschiedene Zeitpunkte grau markiert. Unterhalb der

grauen Markierung liegende Bildpunkte zählen zur Klasse der bisher noch nicht benötigten Bildpunkte, während oberhalb liegende Bildpunkte zu der Klasse der nicht mehr benötigten Bildpunkte zählen. Hieraus ergeben sich die im Bild durch Pfeile symbolisierten Zeitpunkte zu denen Blöcke erstmals (unten) bzw. nicht mehr (oben) benötigt werden. Der Pfeilanfang bezeichnet den zu diesem Zeitpunkt bearbeiteten Bildpunkt, die Pfeilspitze den jeweiligen Block.

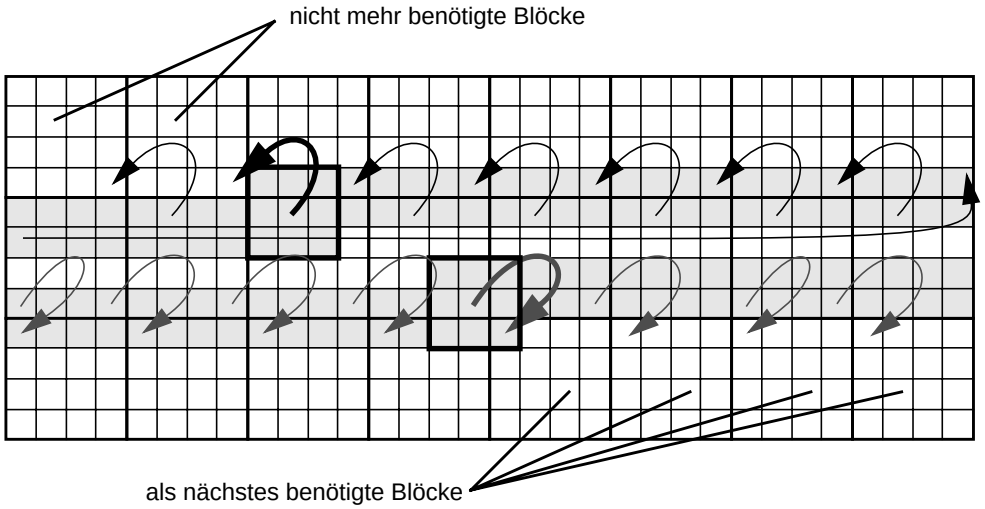


Bild 6.10: Zeitpunkte bei horizontaler zeilenweiser Verarbeitungsabfolge, zu denen Blöcke nicht mehr benötigt (oben) bzw. erstmals benötigt werden (unten)

Aus diesen Zeitpunkten läßt sich die Mindestzahl im Cache zu speichernder Blöcke ermitteln, für den Fall, daß das Nachladen bzw. Verdrängen der Blöcke aus dem Cache sofort zu diesen Zeitpunkten passiert. Wie in Bild 6.11 gezeigt, schwankt die Anzahl zwischen 1 und 2 Blockzeilen, d.h. es sind zwei Blockzeilen erforderlich, damit der Pufferspeicher während dem gesamten Ablauf ausreicht. Daraus ergibt sich als Anforderung an das Plazierungsschema, daß zwei aufeinanderfolgende Zeilen bzw. Spalten gleichzeitig komplett im Cache abgespeichert werden können.

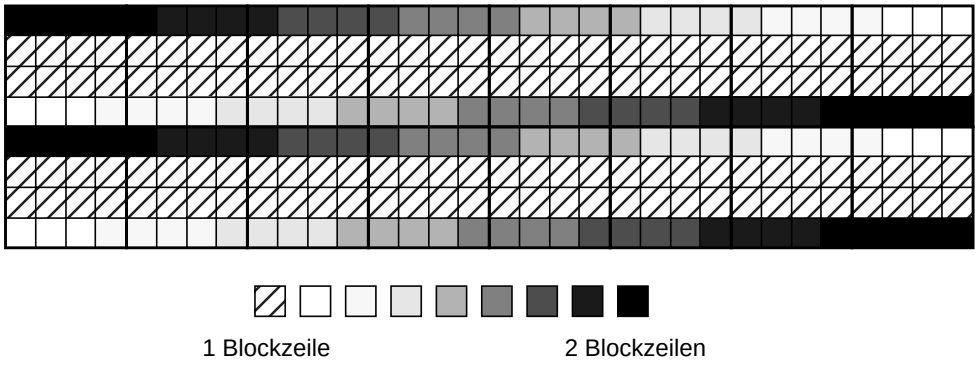


Bild 6.11: Bei horizontaler Verarbeitungsabfolge benötigte Anzahl an Blöcken zu verschiedenen Zeitpunkten

Wenn das Laden zu den in Bild 6.10 gezeigten spätestmöglichen Zeitpunkten erfolgt, ergibt sich keine gleichmäßige Auslastung des Busses zum Hintergrundspeicher, da während der Verarbeitung einer Zeile alle Transfers erfolgen, und während der drei folgenden Zeilen keine. Wird jedoch mit dem Laden bereits früher begonnen, steht mehr Zeit für die Transfers bereit, so daß die Buslast gleichmäßig verteilt werden kann. Da, wie in Bild 6.11 gezeigt, die meist Zeit weni-

ger als die erforderlichen zwei Blockzeilen benötigt werden, können die Ladezeitpunkte zur gleichmäßigen Verteilung der Transfers vorgezogen werden. Die frühestmöglichen Zeitpunkte, zu denen ein Block nachgeladen werden kann, ergeben sich aus den Zeitpunkten, an denen der zuvor an derselben Cache-Position gespeicherte Block nicht mehr benötigt wird.

Neben dem vorausschauenden Laden von Bilddaten kann auch ein vorausschauendes Zurückschreiben erfolgen. Dessen Einsatz erlaubt eine effiziente Pufferung der Ergebnisdaten bzw. den Einsatz bei rekursiver Verarbeitung. Da beim Speichern von Daten ebenfalls eine gleichmäßige Busauslastung angestrebt wird, kann auch das Abspeichern von Blöcken verzögert durchgeführt werden. Hierbei ist der Zeitpunkt, an dem ein Cache-Block erstmals anderweitig benötigt wird, gleich dem Zeitpunkt, an dem Daten spätestens zurückgeschrieben sein müssen.

Für vertikale, umgekehrte horizontale bzw. umgekehrte vertikale Scanrichtung ergeben sich die Zeitpunkte zu denen Blöcke erstmals bzw. letztmals benutzt werden durch Spiegelung. Für eine eindimensionale Nachbarschaft gelten dieselben Überlegungen, wobei maximal eine Blockzeile an Cache-Speicher (minimal: ein Block) benötigt wird und somit mehr Zeit für die Verteilung der Transfers zur gleichmäßigen Busauslastung zur Verfügung steht.

Vor Beginn der Verarbeitung ist das Laden von ein oder zwei Cache-Blockzeilen zur Initialisierung erforderlich, abhängig davon, ob mit einer zweidimensionalen Nachbarschaft an einer Blockgrenze gestartet wird. Nach dem Laden des ersten Blockes (bzw. der ersten beiden Blöcke) kann zwar mit der Verarbeitung begonnen werden, da die Cache-Blockzeilen jedoch während der Verarbeitung nur einer Bildpunktzeile geladen werden müssen, kommt es während diese Initialisierungsphase zu Wartezyklen.

In Tabelle 6.13 sind die Ergebnisse der in Anhang E gezeigten Abschätzung gegenübergestellt. Bei einer Blockgröße von 2x2 Bildpunkten ist ein Cache mit vorausschauendem Laden in etwa genauso performant wie ein Zeilenspeicher. Bei steigenden Blockgrößen steigt auch der Aufwand, und ist bereits bei den Blockgrößen 8x8 und 16x16 unverhältnismäßig hoch. Bei den Blockgrößen 2x2 und 4x4 ist der Aufwand verhältnismäßig klein, so daß sich ein Cache mit Prefetching als effizienter Pufferspeicher für alle Adressierungsarten einsetzen läßt.

	Zeilen-speicher	Prefetching			
		2x2	4x4	8x8	16x16
zusätzliche Dauer	1367400	2041480	3978176	7361184	22281504
rel. zur Gesamtdauer	2,1%	3,2%	6,2%	11,5%	34,9%

Tabelle 6.13: Vergleich des Overheads von Zeilenspeicher und Cache mit Prefetching

Optimale Cache- und Blockgröße

Die optimale Blockgröße muß anhand der vorherigen Untersuchungen so gewählt werden, daß sowohl für reguläre als auch für irreguläre Adressierung eine effiziente Pufferung möglich ist. Während bei Segment-Adressierung eine Blockgröße von 4x4 bzw. 8x8 am besten geeignet sind, ergibt sich für reguläre Adressierung, daß eine möglichst kleine Blockgröße gewählt werden muß. Die Blockgröße 4x4 erfüllt beide Anforderungen am besten. Prefetching erfordert für eine effiziente Operation eine Mindestgröße des Cache von 2 Cache-Blockzeilen pro Bild. Um zwei Eingangs- und ein Ausgangsbild zu verwalten sind also mindestens 6 Cache-Blockzeilen erforderlich. Da die mittlere Zugriffszeit für irreguläre Adressierung bei einer Vergrößerung des Cache nicht sprunghaft abnimmt, ist im Hinblick auf eine kompakte Architektur eine Cache-Größe gleich der Mindestgröße von 6 Cache-Blockzeilen zweckmäßig.

6.1.2.2 Speicher für Segmentindizierte Daten / Histogramme

Neben den Bilddaten müssen auch segmentindizierte Daten verwaltet werden. Segmentindizierte Daten setzen sich aus Segmenteigenschaften und Histogrammdaten zusammen. Die Segmenteigenschaften umfassen die in Tabelle A.1 im Anhang gezeigten Felder und stellen insgesamt 409 bit dar. Bis auf 3 Felder, die nur vom High-Level-Algorithmus benutzt werden und 38 bit umfassen, werden alle Felder von der Architektur benutzt.

Aufgrund ihrer Größe müssen die segmentindizierten Daten chipextern gespeichert werden. Da der Datenbus vom Koprozessor zum externen Speicher nicht so breit dimensioniert werden kann, daß ein kompletter Datensatz an Segmenteigenschaften bzw. Histogrammdaten in einem Zugriff geladen werden kann, müssen die segmentindizierten Daten aufgeteilt werden. Bei einer Datenbusbreite von 128 bit kann ein Histogramm in 32 Teile unterteilt werden, die jeweils 8 Histogrammeinträge zu 16 bit umfassen. Für eine sinnvolle Aufteilung der Segmenteigenschaften ist eine Bündelung inhaltlich zusammengehöriger Kanäle zweckmäßig, wie in Tabelle A.2 im Anhang gezeigt.

6.1.2.2.1 Konzepte für die Pufferung segmentindizierter Daten

Im Folgenden wird untersucht, wie sich segmentindizierte Daten effizient puffern lassen, so daß trotzt externer Speicherung ein schneller Zugriff möglich ist.

Pufferregister

Segmentindizierte Daten lassen sich durch das Eingangsregister der Verarbeitungseinheit puffern. Wie in Bild 6.12 gezeigt, wird in einem zusätzlichen Register die Adresse der gespeicherten Daten abgelegt. Bei einem erneuten Zugriff wird die Zugriffsadresse mit der gespeicherten Adresse verglichen, um festzustellen, ob die gewünschten Daten im Eingangsregister vorliegen.

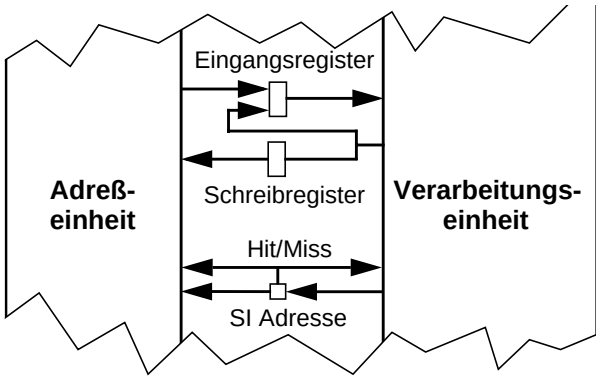


Bild 6.12: Verwendung des Eingangsregisters zur Pufferung segmentindizierter Daten

Operation	Histo-gramm	Segment-eigensch.	Fragm. prüfen	Segm. markieren	Relaxation	Gesamt
Hit-Rate	70,28%	90,57%	3,09%	0,00%	92,16%	84,06%

Tabelle 6.14: Hit-Raten des Pufferregisters für segmentindizierte Daten

Wenn dies nicht der Fall ist, wird der Inhalt des Eingangsregisters aus dem externen Speicher geladen, und anschließend die im Schreibregister abgelegten vorherigen bzw. geänderten Werte zurückgeschrieben. Tabelle 6.14 zeigt die Simulation der Hit-Raten für die Foreman Sequenz mit QCIF Bildformat. Aufgrund des Pufferregisters braucht ein Großteil der Zugriffe nicht durchgeführt zu werden. Die geringe Hit-Rate für die Operation „Segment markieren“ kann außer acht

gelassen werden, da segmentindizierte Zugriffe bei dieser Operation sehr selten auftreten.

Cache

Um segmentindizierte Zugriffe weiter zu beschleunigen, kann zusätzlich ein Cache eingesetzt werden, der häufig benutzte Daten vorhält. Bei Zugriffen, bei denen sich das Datum nicht im Pufferregister befindet, wird dann geprüft, ob es stattdessen im Cache zu finden ist. Davon abhängig wird es entweder aus dem Cache oder aus dem externen Speicher geladen. Da ein Histogrammeintrag nur 16 bit umfaßt, und andererseits bereits für den Zugriff auf segmentindizierte Daten ein wesentlich breiterer Datenbus zur Verfügung steht, können mehrere Histogrammeinträge mit einem Zugriff geladen werden. Dadurch läßt sich die Trefferrate bei Zugriffen auf Histogramme ohne zusätzliche Lesezugriffe vergrößern.

Cache-Einträge		Histo-gramm	Segment-eigensch.	Fragm. prüfen	Segm. markieren	Relaxation	Gesamt
2	(104 Byte)	0,49%	0,13%	0,00%	0,00%	0,05%	0,25%
4	(208 Byte)	1,01%	0,40%	0,00%	0,00%	0,26%	0,60%
8	(416 Byte)	1,98%	1,34%	1,03%	0,00%	0,90%	1,59%
16	(832 Byte)	3,35%	2,66%	2,06%	0,00%	1,86%	2,97%
32	(1,6 kByte)	5,81%	4,74%	5,16%	0,00%	3,34%	5,23%
64	(3,3 kByte)	9,38%	7,51%	8,25%	0,00%	6,43%	8,23%
128	(6,5 kByte)	14,16%	9,09%	10,31%	0,00%	7,50%	10,85%

Tabelle 6.15: Hit-Rate des Cache für segmentindizierte Daten je nach Größe

Cache-Einträge		QCIF	CIF
2	(104 Byte)	0,25%	0,15%
4	(208 Byte)	0,60%	0,71%
8	(416 Byte)	1,59%	1,34%
16	(832 Byte)	2,97%	2,51%
32	(1,6 kByte)	5,23%	3,78%
64	(3,3 kByte)	8,23%	5,48%
128	(6,5 kByte)	10,85%	8,05%

Tabelle 6.16: Vergleich der Hit-Raten des Cache für segmentindizierte Daten für QCIF- und CIF-Bildformat

Die Simulationsergebnisse in Tabelle 6.15 wurden durch Simulation der segmentindizierten Zugriffe beim Einsatz eines Pufferregisters in Kombination mit einem Cache bestimmt. Um die maximal erreichbare Hit-Rate abzuschätzen, wurde ein vollassoziativer Cache mit LRU Verdrängung simuliert. Die angegebenen Werte beziehen sich dabei lediglich auf die Treffer des Caches. Durch Vergrößerung des Caches kann die Hit-Rate gesteigert werden, wie in Tabelle 6.15 ersichtlich, die Steigerung ist jedoch relativ gering.

Während Tabelle 6.15 die Hit-Rate für die Foreman-Sequenz im QCIF Bildformat angibt, wird in Tabelle 6.16 die Hit-Rate für QCIF und CIF verglichen. Wie die Simulationen zeigen, kann die Hit-Rate durch den zusätzlichen Cache nur vergleichsweise gering gesteigert werden, so daß es im Bezug auf das Verhältnis zwischen Aufwand und Nutzen am günstigsten ist, für segmentindizierte Zugriffe lediglich ein Pufferregister einzusetzen.

6.1.2.3 Koordinatenspeicher

6.1.2.3.1 Charakterisierung der Zugriffe

Die Segment-Adressierung unterteilt sich, wie in Kapitel 2.5.5 dargestellt, in Startpunktsuche und Ausbreitungsprozeß. Während des Ausbreitungsprozesses werden die Koordinaten der im Folgenden zu verarbeitenden Bildpunkte in dem Koordinatenspeicher zwischengespeichert. Da alle Punkte gespeichert werden, die das Nachbarschaftskriterium erfüllen, handelt es sich bei CON4 bzw. CON8 Nachbarschaft um bis zu vier bzw. acht Punkte pro Bildpunktoperation. Zusätzlich kann die Position eines weiteren, von der Verarbeitungseinheit bestimmten Punktes im Koordinatenspeicher abgelegt werden. Da pro Bildpunktoperation immer nur ein Punkt entnommen wird, sind insgesamt bis zu sechs bzw. zehn Zugriffe möglich.

Je nach Geschwindigkeit der übrigen Architektur ergibt sich die verfügbare Zeitdauer für Ablegen bzw. Entnehmen von Koordinaten. Im schnellstmöglichen Fall stehen zwei Takte zur Verfügung. In diesem Zeitraum muß eine Koordinate entnommen und gegebenenfalls mehrere im Koordinatenspeicher abgelegt werden, wenn der Ablauf nicht verzögert werden soll. Bild 6.13 zeigt die Häufigkeitsverteilung der Anzahl zu speichernder Positionen für CON4 und CON8.

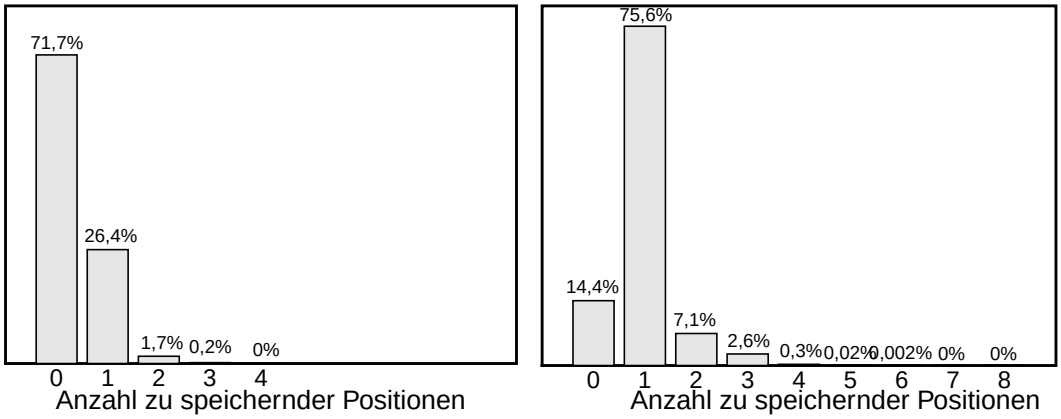


Bild 6.13: Häufigkeit der Koordinatenspeicherzugriffe bei CON4 (li.) bzw. CON8 (re.)

Da in den meisten Fällen keine oder nur eine Koordinate im Koordinatenspeicher abgelegt wird, sind die Anforderungen bezüglich paralleler Zugriffe relativ gering. Im schnellstmöglichen Fall reicht es bereits einen Lese- oder Schreibzugriff pro Takt zu gestatten, um ein hinsichtlich der Geschwindigkeit balanciertes System zu erhalten.

Der Koordinatenspeicher ist entsprechend Kapitel 2.5.8 als hierarchische LIFO implementiert. Während beim Abspeichern von Positionen die Hierarchiestufe stark variiert, wird beim Entnehmen von Positionen streng in Reihenfolge der Hierarchiestufe vorgegangen. Die Zugriffe auf den Koordinatenspeicher erfolgen somit wahlfrei.

6.1.2.3.2 Zeitliche Anforderungen

Um die zeitlichen Anforderungen näher zu untersuchen, wurde zusätzlich zur statistischen Auswertung das Zeitverhalten der Architektur unter Berücksichtigung des Koordinatenspeichers

simuliert. Der Koordinatenspeicher wird durch die Dauer für Abspeichern, für Entnehmen und Durchleiten von Koordinaten modelliert. Mehrere gleichzeitige Anforderungen zum Speichern von Positionen werden sequentiell abgearbeitet. Wahlweise kann das Ablegen und Entnehmen parallel oder sequentiell erfolgen. Die Simulationen berücksichtigen die beiden Haupteinflußfaktoren auf das Zeitverhalten: Geschwindigkeitseinbußen durch eine nicht vollständig gefüllte Schleife über Verarbeitungseinheit, Koordinatenspeicher und Adreßeinheit (siehe Bild 4.3), und Geschwindigkeitseinbußen durch Engpässe bei parallelen Anforderungen zum Abspeichern bzw. Entnehmen von Koordinaten aus dem Koordinatenspeicher.

Mit steigender Zeitdauer für Abspeichern, Entnehmen und Durchreichen ergibt sich zum einen eine Verlängerung der Koordinatenspeicher-Schleife, zum anderen entstehen Engpässe beim Abspeichern und Entnehmen. Die Simulationen (1) bis (3) in Tabelle 6.17 zeigen den gestiegenen Zeitbedarf für 1 bzw. 2 Takte. Anhand der Simulationsergebnisse (1) bis (3) ist der Einfluß der Koordinatenspeicherschleife zu erkennen. Beim Vergleich der Simulationen (1) bis (3) mit den Simulationen (4) bis (6) zeigt sich, daß die Möglichkeit zum parallelen Speichern mehrerer Positionen keinen Vorteil bietet.

Simulation	Parameter				Wartezyklen pro Bildpunktoperation					
	Abspeichern	Entnehmen	Durchleiten	Parallele Operation	Segment Dilatation.	Segment löschen	Fragm. prüfen	Segment markieren	Segment-größe	Wasser-scheide
(1)	0	0	0	Nein	0,00	1,91	0,04	1,26	0,19	0,00
(2)	1	1	1	Nein	0,00	2,93	0,07	1,97	0,27	0,00
(3)	2	2	2	Nein	0,00	3,96	0,10	2,67	0,36	0,00
(4)	0	0	0	Ja	0,00	1,91	0,04	1,26	0,19	0,00
(5)	1	1	1	Ja	0,00	2,93	0,07	1,97	0,27	0,00
(6)	2	2	2	Ja	0,00	3,99	0,11	2,67	0,36	0,00

Tabelle 6.17: Mittlere Anzahl an Wartezyklen beim Zugriff auf den Koordinatenspeicher

Außerdem fällt auf, daß Operationen, bei denen Startpunktsuche und Ausbreitungsprozeß voneinander getrennt sind, (Segment Dilatation, Wasserscheide-Verfahren) sehr unkritisch sind, und eine verschwindende Anzahl an Wartezyklen für Koordinatenspeicherzugriffe erfordern, während Operation, bei denen Startpunktsuche und Ausbreitungsprozeß alternieren, Wartezyklen in Erscheinung treten. Bei Operationen die Startpunkte aus einer Liste lesen (Segment löschen, Segment markieren), entsteht eine große Verzögerung durch Koordinatenspeicherzugriffe, jedoch ist hier auch die Verzögerung durch Cache-Misses am größten, so daß diese Operationen am kritischsten zu implementieren sind.

6.1.2.3.3 Größe des Koordinatenspeichers

Wie im vorherigen Kapitel dargelegt sind kurze Zeiten für Ablegen bzw. Entnehmen von Koordinaten im/aus dem Koordinatenspeicher wichtig für die performante Operation des Gesamtsystems. Andererseits ist eine ausreichende Größe des Koordinatenspeichers wichtig zur Vermeidung von Überläufen während der Operation. Wird der Koordinatenspeicher so groß gewählt, daß seine Größe der Anzahl der Punkte eines kompletten Bildes entspricht, ist seine

Größe zwar in jeden Falle ausreichend, er ist jedoch unnötig groß und bedingt hohe Implementierungskosten.

Die maximale Größe der Wellenfront bei Ausbreitungsprozessen entspricht der Mindestgröße des Koordinatenspeichers. Für eine freie Ausbreitung mit CON4 Nachbarschaft läßt sich die maximale Größe der Wellenfront abschätzen. Eine vergleichbare Abschätzung läßt sich auch für freie Ausbreitung mit CON8 Nachbarschaft durchführen, wobei in diesem Fall die Wellenfront doppelt so schnell wächst. Neben der Anzahl an Startpunkten wird die Größe der Wellenfront insbesondere dadurch bestimmt, aus wievielen zusammenhängenden Teilen der Satz an Startpunkten besteht. Obwohl die Abschätzung für den Fall der eingeschränkten Ausbreitung (d.h. Ausbreitung mit Hindernissen) nicht gültig ist, gibt sie einen Anhaltspunkt über die Größe der Wellenfront am Ende eines Ausbreitungsprozesses. Für eine Ausbreitung mit 3000 Startpunkten in 30 zusammenhängenden Teilen sind bei 4CIF Bildformat 32k Speicherplätze ausreichend. In der Praxis hat sich bei einer Simulation des Segmentierungsalgorithmus mit den Sequenzen „Bream“, „Coastguard“, „Foreman“, „Mother&Daughter“ und „S1“ bereits eine Anzahl von 5544 Speicherplätzen für das Bildformat QCIF als ausreichend herausgestellt. Somit läßt sich der Koordinatenspeicher mit gegenüber der Bildgröße deutlich reduzierter Kapazität implementieren, was eine chipinterne Implementierung mit hoher Zugriffsgeschwindigkeit erlaubt.

6.1.2.4 Speicherarchitektur

In diesem Kapitel wird die Speicherarchitektur insgesamt betrachtet und simuliert. Sie umfaßt insgesamt drei verschiedene Speicher: Den Bildspeicher, den Speicher für segmentindizierte Daten und den Koordinatenspeicher. Aufgrund der zuvor genannten Forderung nach hoher Zugriffsgeschwindigkeit bei wahlfreiem Zugriff wird der Koordinatenspeicher intern mit statischem RAM ausgeführt. Für die Implementierung von Bildspeicher und Speicher für segmentindizierte Daten ist zu betrachten, welche Speichertechnologie eingesetzt werden kann, welcher Pufferspeicher sich eignet und ob Speicher bzw. Pufferspeicher zweckmäßigerweise intern oder extern implementiert werden, um eine schnelle Datenzufuhr zu gestatten. Darüberhinaus können die Schnittstellen für Bildspeicher und Speicher für segmentindizierte Daten entweder getrennt oder gemeinsam implementiert werden.

Die Kombination aller Freiheitsgrade ermöglicht eine Fülle an Varianten. In Bild 6.15 wird für mögliche Varianten der Speicherarchitektur ein Entscheidungsbaum aufgestellt. Zweige, für die absehbar ist, daß sich keine sinnvolle Lösung ergibt, werden nicht weiter verfolgt, wobei der Grund jeweils in der Legende erläutert ist. Für die übrigen Zweige ist in Bild 6.14 das Ergebnis der Simulation der Bilder 1 bis 20 der QCIF-Sequenz „Foreman“ dargestellt. Die Simulationsparameter sind:

- 1 Takt Zugriffszeit auf Koordinatenspeicher
- Größe des Caches für Bilddaten (falls eingesetzt): 16896 Bildpunkte
- Blockgröße des Caches für Bilddaten: 4x4 Bildpunkte
- Direktabgebildeter Cache für Bilddaten
- Hintergrundspeicher - interner Cache: Datenbusbreite 2 Bildpunkte
- Hintergrundspeicher - externer Cache: Datenbusbreite 4 Bildpunkte

Wie in der Legende erläutert, lohnt sich die Betrachtung von Alternativen, bei denen Bildspeicher und Speicher für segmentindizierte Daten extern implementiert sind, über eine Schnittstelle zugegriffen werden und einheitlich aus statischem oder dynamischen Speicher aufgebaut sind.

Die mit 52,1 Bildern/sec schnellste Verarbeitung ermöglicht statischer Speicher, wenn ein interner Cache mit Prefetching eingesetzt wird und der parallele Zugriff auf 9 Bildpunkte des Caches möglich ist. Mit Verringerung der parallel zugreifbaren Bildpunkte ergibt sich eine leichte Verringerung der Verarbeitungsgeschwindigkeit. Während ein Zeilenspeicher die Anzahl der Taktzyklen um 16,3% (Variante G vs. H) gegenüber einer reinen SRAM-Lösung reduziert, läßt sich

durch Einsatz eines Cache eine Reduktion von 29,9% (Variante F vs. H) und beim zusätzlichen Prefetching eine Reduktion von insgesamt 51,0% (Variante C vs. H) erzielen.

Um die hohen Implementierungskosten für statischen Speicher zu vermeiden, kann stattdessen dynamischer Speicher eingesetzt werden. Auch in diesem Fall läßt sich mit 39,9 Bildern/sec eine hohe Verarbeitungsgeschwindigkeit erreichen, jedoch steigt damit die Verarbeitungsdauer um 30,7% (Variante K vs. C) gegenüber eine SRAM-Lösung. Der Nutzen des Cache bzw. Prefetching ist hier mit einer Reduktion der Taktzyklen um 51,6% bzw. 63,6% gegenüber einer Architektur ohne Pufferspeicher für Bilddaten besonders groß (Variante N bzw. K vs Q). Ein externer Cache erweist sich als langsamer als der Einsatz von SRAM Hintergrundspeicher ohne Cache.

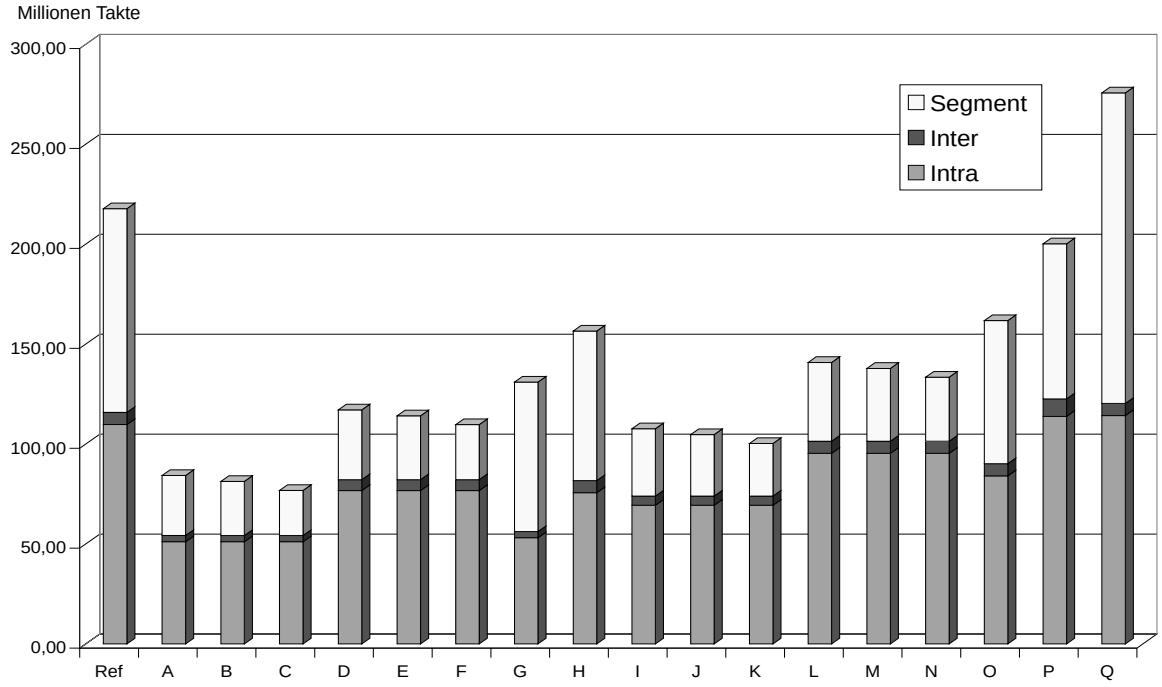


Bild 6.14: Verarbeitungsdauer der Speicherarchitekturvarianten

Legende zu Bild 6.15:

(1) Bildspeicher und Speicher für segmentindizierte Daten sind sowohl bei Einsatz von statischem als auch beim Einsatz von dynamischem Speicher zu groß, um auf dem Chip integriert zu werden.

(2) Sind der Bildspeicher und der Speicher für segmentindizierte Daten beide mit zwei getrennten Schnittstellen extern angebunden, ist die Verarbeitungsgeschwindigkeit geringer, als bei Anbindung über eine Schnittstelle, da aufgrund begrenzter Pinanzahl Bilddaten aufgeteilt in Komponentenbündel zugegriffen werden müssen.

(3) Speicher für Bilddaten und segmentindizierte Daten sollten mit gleicher Technologie implementiert werden, um die Implementierung zu vereinfachen.

(4) Ein externer Cache bringt bei statischem Hintergrundspeicher keinen Vorteil, da die Bandbreite zum Cache in gleichem Maße eingeschränkt ist wie bei direktem Zugriff auf den Hintergrundspeicher, der wie der Cache ebenfalls wahlfrei zugegriffen werden kann.

(5) Die Zugriffe auf den Speicher für segmentindizierte Daten können bereits mit sehr geringem Aufwand (Pufferregister) stark verringert werden.

(6) Ein Cache für segmentindizierte Daten bringt trotz wesentlich höherem Hardware-Aufwand gegenüber einem Pufferregister nur eine geringe Steigerung der Hit-Rate.

(7) Da die Busbreite groß genug dimensioniert werden kann, um mehr als einen Bildpunkt pro Zugriff zu laden, wird diese Variante nicht betrachtet.

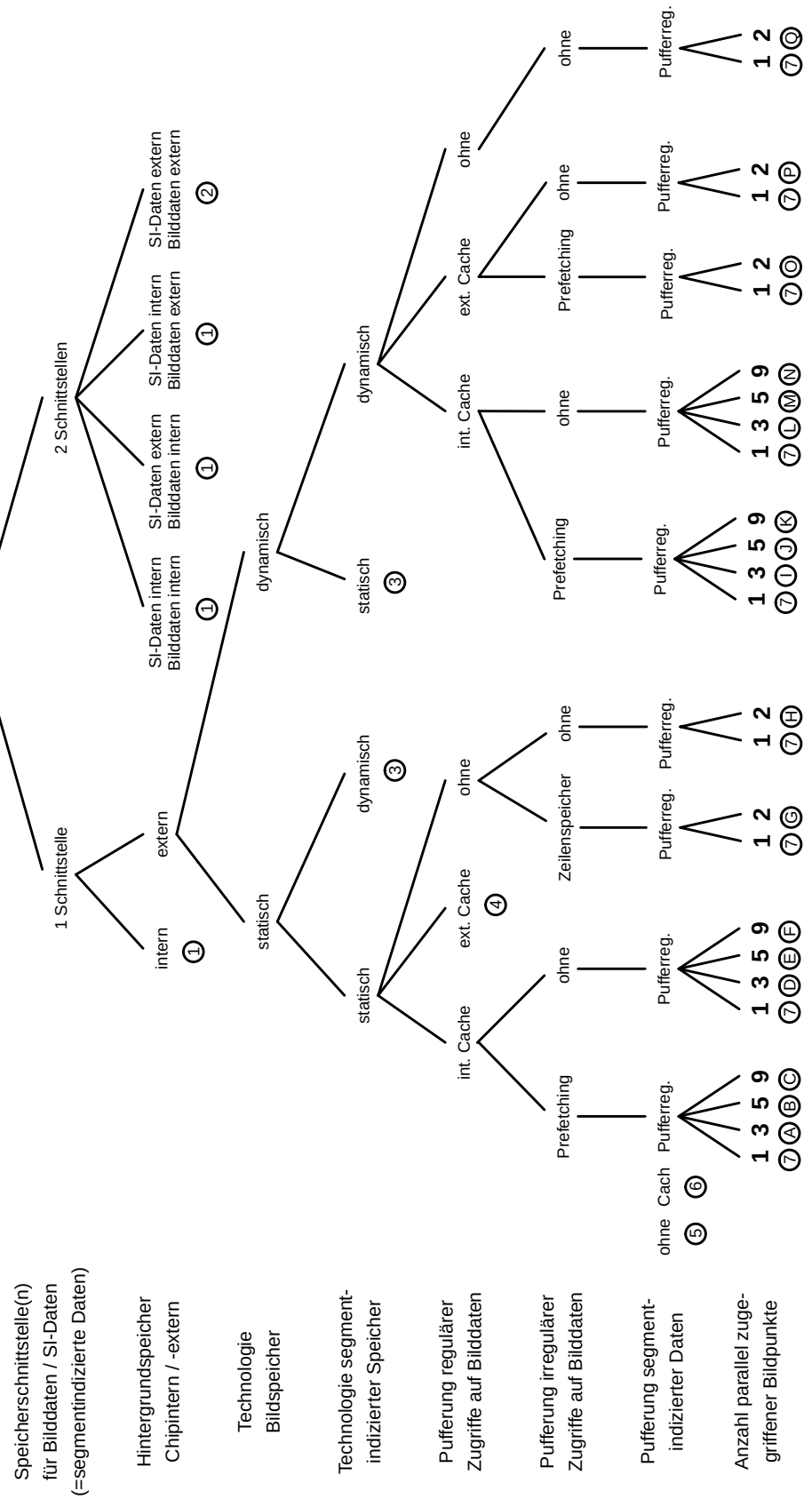


Bild 6.15: Entscheidungsbaum Speicherarchitektur

6.1.3 Architekturvorschlag

Aus den zuvor dargelegten Untersuchungen wird nun eine Architekturentscheidung abgeleitet. Als schnellste Architektur hat sich Variante C bei Einsatz statischen Speichers, bzw. Variante K bei Einsatz dynamischen Speichers erwiesen. Da sich beide lediglich in der Ansteuerung des externen Hintergrundspeichers unterscheiden, wird eine Architektur vorgeschlagen, die beide Arten von Speicher ansteuern kann und damit je nach Anwendung mit flächeneffizientem DRAM oder geschwindigkeitssteigerndem SRAM eingesetzt werden kann. Aufgrund des unwesentlichen Unterschieds zwischen Architektur B und C bzw. zwischen J und K, sollte die Anbindung der Registermatrix an den Cache für parallelen Zugriff auf 5 Bildpunkte ausgelegt werden, da hierdurch die Verdrahtung wesentlich erleichtert wird. Zusammenfassend ergeben sich folgende Architekturmerkmale:

- Operationsablauf
  - Keine Differenzierung nach Zweigen beim Start- / Nachbarschaftskriterium
  - Keine vorgezogene Prüfung des Markerkriteriums
  - Unterschiedlicher Nachbarschaftsmodus für Start- / Nachbarschaftskriterium
- Speicherung von Bilddaten und segmentindizierte Daten
  - Gemeinsamer externer Speicher
  - Zugriff über gemeinsame Schnittstelle
  - Paralleler Zugriff auf 2 Bildpunkte (alle Komponenten)
- Pufferspeicher für Bilddaten
  - Chipinterner Cache mit Prefetching bei regulärer Adressierung
  - Blockgröße 4x4, 1056 Blöcke, direktabgebildet, „write-back“-Schreibstrategie
  - Paralleler Zugriff auf 5 Bildpunkte, Bankzugehörigkeit entsprechend Bild 6.4 (h)
- Pufferspeicher für SI-Daten
  - Einzelnes Pufferregister und Schreibpuffer
- Koordinatenspeicher
  - 32k Einträge
  - Zugriff auf ein Datum gleichzeitig
  - Lesen und Schreiben sequentiell
  - Zugriffsdauer 1 Takt

6.2 Implementierung der vorgeschlagenen Architektur

6.2.1 Steuerungseinheit

Eine Bildpunktoperation besteht aus mehreren Teiloperationen, die im Interesse einer effizienten Implementierung überlappend ausgeführt werden. Die Teiloperationen können gemeinsame Ressourcen benutzen, wie z.B. die Speicherschnittstelle beim Laden und Speichern, und sie können sich je nach Konfiguration in ihrem Auftreten und ihrer Dauer unterscheiden.

In [Fuc98] und [Zuk99] ist eine Methodik beschrieben, wie eine Steuereinheit entworfen werden kann, die ein System automatisch anhand der Dauer der Teiloperationen und der Priorität der Ressourcennutzung steuert, ohne daß der Ablauf explizit festgelegt werden muß. Durch die dynamische Anpassung der Steuerung wird außerdem eine hohe Flexibilität hinsichtlich Änderungen des Ablaufes erreicht.

In [Fuc98] und [Zuk99] wurde ebenfalls ein Controller entworfen, der auf die Steuerung der vorgeschlagenen Architektur ausgerichtet ist. Er besteht aus Scan-Controller und Pixel-Level-Controller. Der Ablauf auf Bildebene wird vom Scan-Controller unter Benutzung des Koordinatenregisters gesteuert, der Ablauf einer Bildpunktoperation wird vom Pixel-Level-Controller gesteuert.

6.2.2 Verarbeitungseinheit

6.2.2.1 Blockstruktur

Um durch orthogonale Konfiguration eine hohe Flexibilität zu erreichen, orientiert sich der Aufbau der Verarbeitungseinheit am Aufbau der zu unterstützenden Bildpunktoperationen aus Teiloperationen. Jede der Teiloperationen wird in einem unabhängig konfigurierbaren Modul implementiert, so daß durch die Kombination verschiedener Teiloperationen eine große Bandbreite an Operationen implementiert werden kann.

Wie aus Tabelle 6.18 ersichtlich, unterscheiden sich die Bildpunktoperationen bezüglich ihrer Struktur der Verarbeitung und ihrer Schnittstellen je nach Adressierungsart. Aus diesem Grund wird für jede Adressierungsart ein eigener Block vorgesehen, der die jeweiligen Operationen ohne Vor- und Nachverarbeitung implementiert.

Bei vielen Operationen, insbesondere bei Segment-Operationen, hängen weitere Verarbeitungsschritte von verschiedenen Bedingungen ab. Da für unterschiedliche Operationen ähnliche Bedingungen geprüft und verknüpft werden, wird deren Auswertung in einem Block zusammengefaßt, der alle Vergleiche an zentraler Stelle ausführt. Die Ergebnisse der Vergleiche steuern die anderen Blöcke, so daß der Abfrageblock bei den meisten Bildpunktoperationen beteiligt ist.

Eine besondere Rolle spielen segmentindizierte Operationen, die mit Operationen mit regulärer Adressierung kombiniert werden können. Um Ressourcenkonflikte auszuschließen, werden daher segmentindizierte Operationen in einem eigenen Block implementiert, der auch die Auswertung der für diese Gruppe zu prüfenden Bedingungen übernimmt. Da die zu prüfenden Kriterien nur bei segmentindizierter Adressierung benützt werden, ist es auch nicht nötig, diese Kriterien an zentraler Stelle zu prüfen.

Adressierungsart	Struktur der Verarbeitung	Schnittstellen der Verarbeitung
Intra-Adressierung	Verarbeitung einer Nachbarschaft, relativ komplexe Strukturen (z.B. Rangordnungsfilter, lineares Filter)	Eingang: 9*Y/U/V und 9*A/AX Ausgang: Y/U/V und A/AX
Inter-Adressierung	Verknüpfung zweier Daten, relativ einfache Strukturen aus Grundelementen (z.B. Subtraktion, Betragsbildung)	Eingang: Y/U/V und A $Y_2/U_2/V_2$ und A_2 Ausgang: Y/U/V und A
Segment-Adressierung	Prüfung verschiedenster Kriterien, relativ komplexe Berechnung der Homogenität und des Wasserscheide-Verfahrens, ansonsten einfache Strukturen aus Grundelementen	Eingang: 9 * Y/U/V und 9 * A/AX, Hierarchieebene h, Distanzwert d, Position (x,y) Ausgang: A/AX, 9*Queue In 9*Hierarchieebene
segmentindizierte Adressierung	einfache Struktur aus Grundelementen (z.B. Addition, Minimum, Maximum)	Eingang: Y/U/V, A/AX, SI-Daten, Position (x,y), Ausgang: Y/U/V, A/AX, SI-Index, SI-Daten

Tabelle 6.18: Vergleich von Struktur und Schnittstellen der Verarbeitung

Neben der komponentenweisen Verarbeitung (wie z.B lineare Filterung) erfolgen bei vielen Bildpunktoperationen zusätzliche Vor- bzw. Nachverarbeitungsschritte. Beispielsweise werden Farb- und Helligkeitskomponenten der Verarbeitungseinheit mit einheitlich 8 bit zugeführt, Wertebereich und Zahlenformat variieren jedoch in der Regel. Bei YUV-Farbformat hat die Y-Komponente einen Wertebereich von 0 bis 255, während die Farbkomponenten im Bereich -128 bis

+127 liegen und mit einem Offset von 128 behaftet sind. Als Verarbeitungsergebnis entstehen ebenfalls Werte im Bereich von -128 bis 127 ohne Offset, die in einem folgenden Schritt wiederum als Eingangsdaten genutzt werden. Um die Verarbeitung unabhängig von Wertebereich und Zahlenformat durchführen zu können, erfolgt vor der Verarbeitung eine Formatkonversion in eine einheitliche vorzeichenbehaftete Darstellung mit 9 bit.

Bei einigen Operationen findet vor der Verarbeitung zusätzlich eine Verknüpfung von Komponenten durch Addition bzw. Subtraktion statt. Beispiele hierfür sind die Operationen, bei denen der Gradient der Summe bzw. Differenz der Farbkomponenten gebildet wird.

Bei vielen nachbarschaftsbasierten Operationen (wie z.B. Erosion und Dilatation) wird innerhalb der Nachbarschaft anhand eines Strukturelements ausgewählt, welche Punkte zur weiteren Verarbeitung herangezogen werden. Andere Intra-Operationen werden nur innerhalb eines Segments ausgeführt, d.h. es wird die Segmentzugehörigkeit der Nachbarpunkte geprüft und alle Punkte ausgeblendet, die das Kriterium nicht erfüllen. Damit die Verarbeitung durch eine Einheit erfolgen kann, die einheitlich eine CON8 Nachbarschaft bearbeitet, müssen zuvor die Werte auszublendender Punkte durch einen festen Wert ersetzt werden, der keinen Einfluß auf das Verarbeitungsergebnis hat. Je nach Operation ist dies Null (Bsp.: lineare Filter), der maximal darstellbare Wert (Bsp.: Minimum) oder der minimal darstellbare Wert (Bsp.: Rangordnungsfilter).

Bei manchen Intra- und Inter-Operationen werden die Ergebnisse der Verarbeitung für einzelne Komponenten durch Addition, Subtraktion, Minimum oder Maximumbildung miteinander verknüpft. Abschließend muß das Ergebnis in ein wählbares Zahlenformat gebracht werden. Dabei müssen die Werte für die Komponenten Y, U und V auf eine Breite von 8 bit reduziert und durch Abschneiden oder Sättigung auf den maximal darstellbaren Wertebereich begrenzt werden.

Intra-Operationen, die die Komponenten Y, U und V verarbeiten, setzen sich somit aus den in Bild 6.16 dargestellten Teiloperationen zusammen. Bei Inter-Operationen, Segment-Operationen und segmentindizierten Operationen erfolgt als Vor- und Nachverarbeitung ebenfalls eine Formatkonversion. Die Nachbarselektion entfällt bei Inter-Operationen, da lediglich mit einer CON0 Nachbarschaft gearbeitet wird. Wie bei Intra-Operationen werden auch bei Inter-Operationen die Ergebnisse der komponentenweisen Verarbeitung gegebenenfalls miteinander verknüpft.

Die Vor-/Nachverarbeitungsschritte werden durch insgesamt fünf Blöcke implementiert:

- Eingangsblock
- Eingangs-Verknüpfungs-Block
- Strukturelement-Block
- Ausgangs-Verknüpfungs-Block
- Ergebnisblock

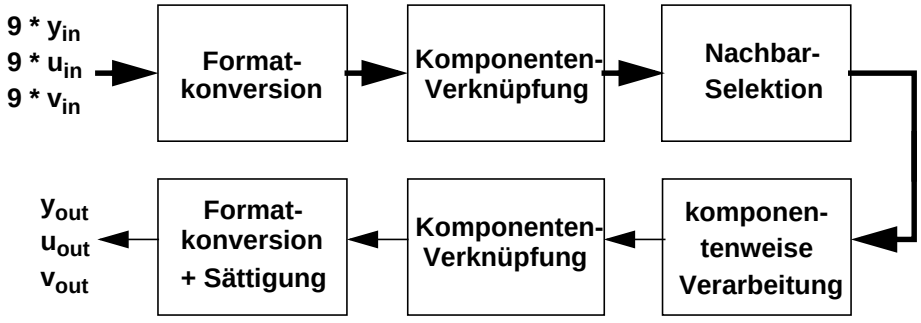


Bild 6.16: Zerlegung einer Intra-Operationen in Teiloperationen

6.2.2.1.1 Block für Intra-Operationen

Bei Intra-Operationen werden die Komponenten Y, U, V, A, sowie AX verarbeitet. Da Wertebereich und Zahlenformat der Helligkeits- und Farbkomponenten nach dem Eingangsblock einheitlich sind, wird ein Modul dreifach in identischer Ausführung verwendet, um die parallele Verarbeitung von Y-, U- und V-Komponente zu ermöglichen. Da die Komponenten A und AX eine andere Bitbreite aufweisen und sich die Operationen für diese Komponenten stark von den Operationen für die Komponenten Y, U und V unterscheiden, werden sie separat implementiert. Die Register, die bei Intra-Operationen verwendet werden, sind in einem weiteren Modul zusammengefaßt.

Das Modul für die Verarbeitung von A- und AX-Komponente beinhaltet ein Untermodul für Relaxation und Median-Filterung von Histogrammen (AX Komponente). Die Module zur Bearbeitung der Y-, U- und V-Komponenten bestehen jeweils aus einem Untermodul für lineare Filterung und einem Untermodul Rangordnungsfilter. Der Rangordnungsfilter wird zur Erosion, Dilatation, Median-Filterung, Bildung des morphologischen Gradient, des Erosions- und des Dilatations-Gradient eingesetzt. Der lineare Filter wird zur Implementierung von Gradient, Sobel-Operator und Laplace-Operator verwendet. Das Resultat wird mit einer Breite von 12 bit ausgegeben, so daß das Ergebnis einer Filterung bei Verknüpfung von Komponenten mit hoher Genauigkeit berechnet werden kann. Das dritte Modul umfaßt 5 Input- und 4 Output-Register zu je 16 bit, wobei die Input-Register von außen beschrieben und die Output-Registern von außen gelesen werden können.

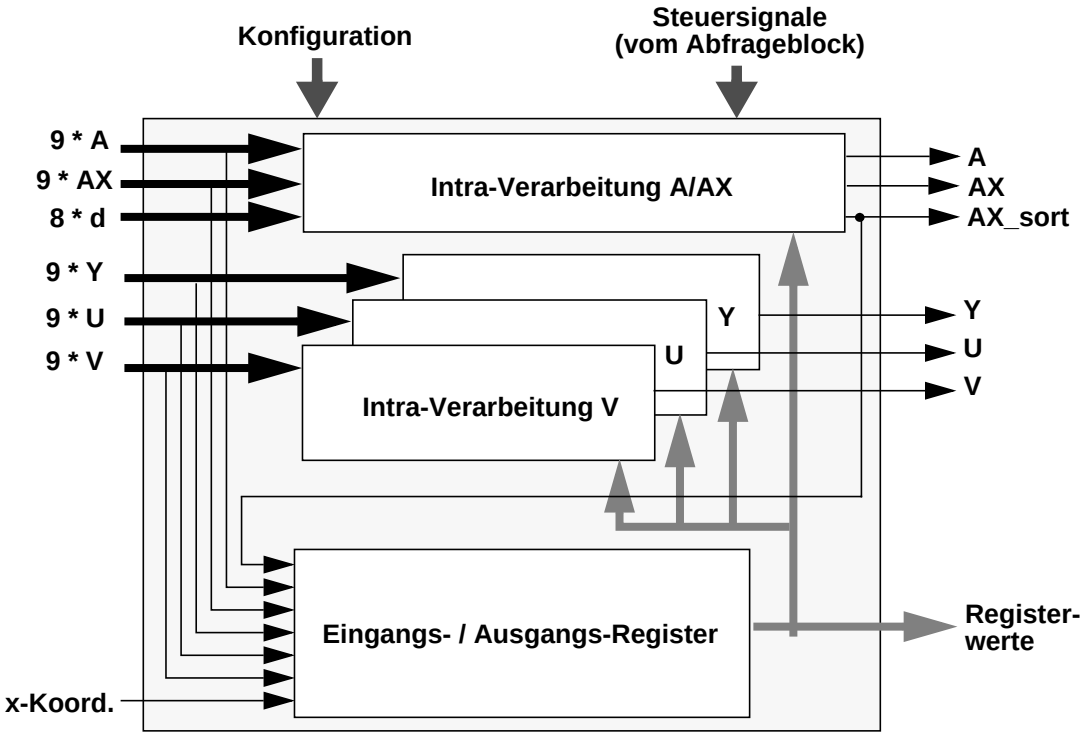


Bild 6.17: Aufbau des Blocks für Intra-Operationen

6.2.2.1.2 Block für Inter-Operationen

Der Block für Inter-Operationen beinhaltet ein Modul zur Durchführung einer gewichteten Bildaddition und zwei universelle Hardware-Module für Bildverknüpfung und Schwellwertbildung. Außerdem sind neun Register mit einer Breite von je 16 bit vorgesehen. Die Bilddifferenz und die ungewichtete Bildaddition werden durch das erste universelle Modul implementiert. Außer-

dem kann in diesem Modul auch der Mittelwert zweier Bildpunkte oder der Betrag bzw. das Quadrat der Bilddifferenz berechnet werden. Schließlich besteht noch die Möglichkeit der Akkumulation des Resultats bzw. der Speicherung des größten Ergebniswertes. Für die Schwellwert-Operation ist ein weiteres universelles Modul vorgesehen, das unterschiedliche Kombinationen von Vergleichen zuläßt. Es können zwei Bildpunkte miteinander und mit zwei Konstanten verglichen werden. Je nach Ergebnis der Vergleiche wird einer der Eingangswerte oder eine von zwei weiteren Konstanten ausgegeben.

6.2.2.1.3 Block für Segment-Operationen

Der Block für Segment-Operationen setzt sich aus vier Teilen zusammen, dem Block für die Homogenitätsberechnung, dem Block für die Wasserscheide-Operation, dem Block für Eingangs- und Ausgangs-Register und zusätzlicher Logik. Im Block für die Homogenitätsberechnung wird aus den Werten der Y-, U- und V-Komponenten und den Gewichtungsfaktoren für Luminanz und Chrominanz die gewichtete Differenz der acht Nachbarpunkte zum Zentralpunkt bestimmt. Das Ergebnis der Homogenitätsberechnung wird an das Modul für die Wasserscheide-Operation und an den Block für Intra-Operationen (Relaxation) und den Abfrageblock weitergegeben. Im Modul für die Wasserscheide-Operation wird aus Homogenitätswerten, der Entfernung (AX) und Informationen über die Auswahl von Nachbarpunkte die Position und der AX-Wert des ähnlichsten Nachbarn ermittelt. Die zusätzliche Logik liefert die Information, welche Punkte auf welcher Hierarchieebene im Koordinatenspeicher abgelegt werden.

6.2.2.1.4 Block für Segmentindizierte Operationen

Um die Werte des segmentindizierten Datensatzes zu lesen, wird im Block für segmentindizierte Operationen zunächst die Speicheradresse berechnet und ausgegeben. Hierauf werden die entsprechenden Daten aus dem Pufferregister oder der entsprechenden Adresse im Speicher gelesen. Für die einzelnen Komponenten werden dann die neuen Werte berechnet und an den Ausgang gelegt. Für jede Komponente wird ein Signal erzeugt, das signalisiert, daß der geänderte Wert abgespeichert werden soll.

6.2.2.2 Datenfluß

Der Datenfluß durch die Verarbeitungseinheit unterscheidet sich je nach Art der Adressierung. Für segmentindizierte Operationen kommen zu den Pfaden für die Verarbeitung der segmentindizierten Daten noch die Pfade der Operation hinzu, mit der die segmentindizierte Operation kombiniert wird. Im Hinblick auf einen geringen Leistungsverbrauch werden Blöcke, die nicht an einer Operation beteiligt sind, durch entsprechende Konfiguration deaktiviert. Dabei werden die Eingangsdaten des Blocks auf einen konstanten Wert gelegt, so daß innerhalb des Blockes keine Schaltvorgängen stattfinden und die Verlustleistung reduziert wird. Die Möglichkeit zur Deaktivierung ist sowohl für die Blöcke der Verarbeitungseinheit, als auch für deren Untermodule implementiert.

6.2.3 Adreßeinheit

6.2.3.1 Organisation des externen Speichers

Im externen Speicher werden Bilddaten und segmentindizierte Daten abgelegt. Abhängig von der Bildgröße muß eine unterschiedliche Zahl an Segmenten verwaltet werden. Bei QCIF ist eine maximale Anzahl von 2^{12} ausreichend, bei CIF 2^{13} und bei 4CIF 2^{14}. Für jedes Segment muß ein Histogramm mit 256 Einträgen zu je 16 bit und ein Datensatz mit Segmenteigenschaften verwaltet werden, der 5 Bündel mit je 128 bit pro Segment umfaßt. Wenn im externen Speicher drei Bilder verwaltet werden, ergeben sich die in Tabelle 6.19 gezeigten Speicheranforderungen.

Bildformat	Bildpunkte	Speicherbedarf		
		Bilddaten	SI-Daten	Gesamt
4CIF	704 x 576	9,28 MByte	9,25 MByte	18,53 MByte
CIF	352 x 288	2,32 MByte	4,63 MByte	6,95 MByte
QCIF	176 x 144	0,58 MByte	2,31 MByte	2,89 MByte

Tabelle 6.19: Speicherbedarf für Bilddaten und segmentindizierte Daten

Die Adreßeinheit wurde für dynamischen Speicher entworfen, wobei auf Anordnung der Daten im Speicher geachtet werden muß. Auf diesem Weg ist die Adreßeinheit durch Änderung der Speicheransteuerung auch für statischen Speicher einsetzbar, während der umgekehrte Weg nicht möglich wäre.

Pro Bildpunkt sind 64 bit vorgesehen, da die Breite des Datenbusses bei externem Speicher nur Vielfache von 8 bit annehmen kann. Während 58 bit zur Speicherung der Bilddaten benötigt werden, lassen sich die zusätzlichen 6 bit pro Bildpunkt zur schnellen Initialisierung der Marker-Komponente einsetzen (siehe Kapitel 6.2.3.3.2). Um parallelen Zugriff auf zwei Bildpunkte zu ermöglichen wird eine Datenbusbreite von 128 bit verwendet. Bei Einsatz von 64 Mbit SDRAMs [Inf99] werden acht SDRAMs mit 16 bit Datenbus parallel angeordnet, so daß ausreichend Platz zur Verarbeitung von 4CIF Bildern zur Verfügung steht. Damit möglichst wenige Verzögerungen durch Seitenwechsel entstehen, werden die Daten so auf den Hintergrundspeicher verteilt, daß die verschiedenen Arten gleichzeitig benutzter Daten in verschiedenen Bänken lokalisiert sind, die gleichzeitig geöffnet sein können.

Da die Adreßeinheit blockweise auf den Speicher zugreift, werden die Bilddaten so im Speicher angeordnet, daß die Bildpunkte eines Blockes fortlaufend im Speicher liegen, so daß beim Zugriff auf einen Block kein Seitenwechsel auftritt und der Zugriff im Burst-Modus erfolgen kann. Innerhalb einer Seite werden die Blöcke so angeordnet, daß diese einen rechteckige Bildausschnitt mit möglichst gleichem Seitenverhältnis umfaßt, damit sowohl bei horizontalen als auch bei vertikalem Scan-Modus möglichst wenige Seitenwechsel auftreten. In Bild 6.18 ist die Blockorganisation dargestellt; in Tabelle 6.20 ist das Seitenverhältnis für verschiedene SDRAM Größen angegeben. Histogrammdaten und segmentindizierte Datensätze werden fortlaufend im Speicher abgelegt und liegen jeweils vollständig innerhalb einer Seite.

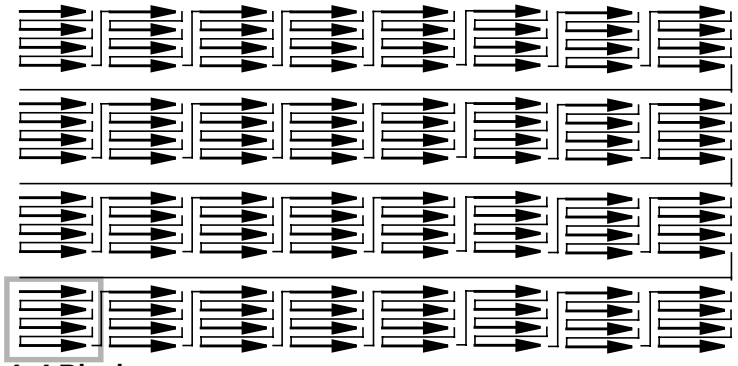


Bild 6.18: Blockanordnung innerhalb einer Speicherseite

SDRAM Größe	Organisation Zeilen x Seiten	Blockanordnung in einer Seite	Histogramme pro Seite
64 Mbit	4.096 x 256	8 x 4	1
128 Mbit	4.096 x 512	8 x 8	2
256 Mbit	8.192 x 512	8 x 8	2
512 Mbit	8.192 x 1.024	16 x 8	4
1 Gbit	16.384 x 1.024	16 x 8	4

Tabelle 6.20: Speicherorganisation in Abhängigkeit der SDRAM Größe

6.2.3.2 Organisation des Pufferspeichers

Der Pufferspeicher für Bilddaten der vorgeschlagenen Architektur wird in acht Speicherbänke unterteilt, um den parallelen Zugriff auf eine CON4 Nachbarschaft zu ermöglichen. Wie in Bild 6.3 dargestellt, werden die zugegriffenen Daten mit einem Multiplexer vertauscht und zur Registermatrix geführt. Um die Implementierung des Multiplexers zu vereinfachen wird die in Bild 6.4 (h) gezeigte Blockorganisation verwendet, bei der zwei unterschiedliche Blocktypen, sogenannte linke und rechte Blöcke auftreten.

Auf die Punkte eines Blocks wird mit der Blockadresse zugegriffen, die sich aus den niederwertigen Teilen der x- und y-Koordinaten ergibt. Die unteren 3 bit der Blockadresse werden zur Auswahl der Speicherbank verwendet, das oberste Bit gibt an, ob auf Punkte in der linken oder rechten Hälfte des Blocks zugegriffen wird. Aus den höherwertigen Teilen der x- und y-Koordinaten ergibt sich die Position eines Blockes, die mittels Plazierungsschema in eine Blocknummer umgerechnet wird. Der Zusammenhang zwischen den Bildkoordinaten und Blocknummer, Blockadresse, Cache-Speicheradresse und Banknummer ist in Bild 6.19 gezeigt.

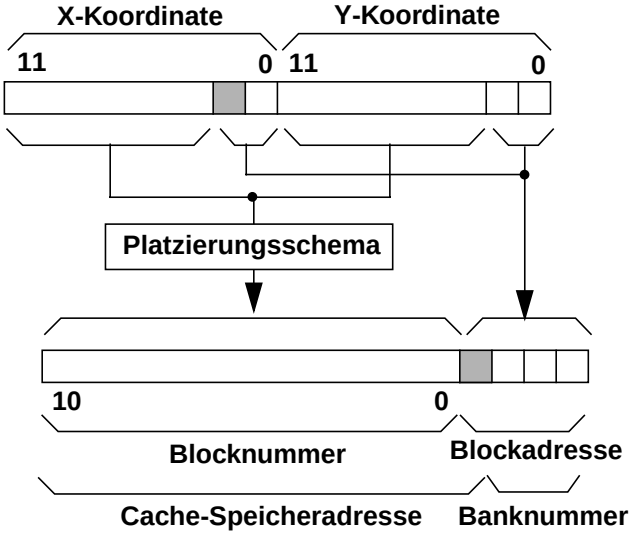
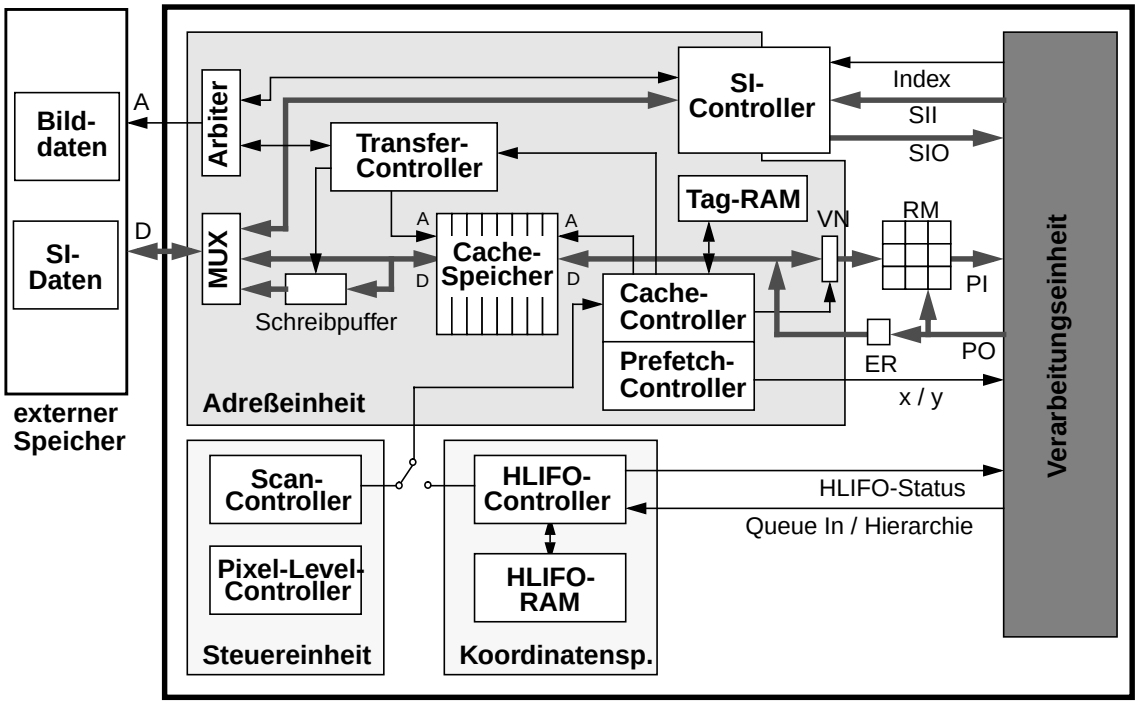


Bild 6.19: Zusammenhang zwischen Bildkoordinaten und Adressen

6.2.3.3 Blockstruktur

Bild 6.20 zeigt den Aufbau der Adreßeinheit. Sie stellt die Schnittstelle zwischen Hintergrundspeicher und Verarbeitungseinheit dar und beinhaltet zwei Datenpfade. Über den einen werden Bilddaten transferiert und im Cache zwischengespeichert, der andere dient dem Zugriff auf segmentindizierte Daten und führt zum SI-Controller.

Auf den Cache wird von zwei Seiten zugegriffen: Zum einen werden Blöcke zwischen Hintergrundspeicher und Cache übertragen, zum anderen werden für die Verarbeitung benötigte Daten bereitgestellt bzw. Ergebnisse gespeichert. Da bei regulärer Adressierung das Vorladen bzw. Zurückschreiben von Cache-Blöcken parallel zur Verarbeitung erfolgen muß, verfügt der Cache-Speicher über zwei Ports, über die gleichzeitig gelesen und/oder geschrieben werden kann. Konflikte durch gleichzeitigen Zugriff auf dieselbe Speicherzelle treten nicht auf, da beim Vorladen bzw. Zurückschreiben nicht auf die von der Verarbeitungseinheit benützen Blöcke zugegriffen wird.



A: Adresse PI: Eingang Bilddaten SII: Eingang segmentindizierte Daten
D: Daten PO: Ausgang Bildaten SIO: Ausgang segmentindizierte Daten
RM: Registermatrix ER: Ergebnisregister VN: Vertauschungsnetzwerk

Bild 6.20: Aufbau der Adreßeinheit und Einbettung ins Gesamtsystem

6.2.3.3.1 Cache-Speicher, Cache-Controller und Tag-RAM

Aufgabe des Cache-Controllers ist die Steuerung von Lese- bzw. Schreibzugriffen der Verarbeitungseinheit auf Bilddaten, die Verwaltung des Cache-Zustandes und die Initiierung des Nachladens bzw. Zurückschreibens von Cache-Blöcken vom/zum Hintergrundspeicher.

Bei einem Zugriff überprüft der Cache-Controller, ob an der vom Plazierungsschema festgelegten Stelle ein Block des Bildbereichs abgelegt ist und ob es sich um den gewünschten Block handelt. Wenn sich die benötigten Blöcke nicht bereits alle im Cache befinden, initiiert der Cache-Controller das Zurückschreiben zu verdrängender Cache-Blöcke, das Laden bzw. die Initialisierung der fehlenden Blöcke und aktualisiert den Cache-Zustand dementsprechend. Ein Zurückschreiben ist dabei nur notwendig, wenn eine Cache-Block durch die Verarbeitung verändert

wurde, was beim Abspeichern von Verarbeitungsergebnissen festgehalten wird. Anschließend werden die acht Cache-Bänke entsprechend der Position und der Nachbarschaft adressiert und für jede Bank Lese- bzw. Schreibsignale generiert. In den Datenpfaden vom Cache zur Registermatrix werden die Multiplexer im Vertauschungsnetzwerk angesteuert, welche die korrekte Zuordnung der Bildpunkte der Nachbarschaft zu den einzelnen Cache-Bänken sicherstellen. Damit bei Segment-Adressierung Cache-Blöcke, welche für das Abspeichern von Ergebnissen der laufenden Verarbeitung noch benötigt werden, nicht zwischenzeitlich verdrängt werden, werden diese gesperrt und zwischenzeitliche Zugriffe gegebenenfalls verzögert.

Zur Verwaltung des Cache-Zustandes muß für jeden Cache-Block gespeichert werden, ob er gültige Daten enthält (present) und ob diese durch Schreibzugriffe verändert wurden (dirty). Da unterschiedliche Blöcke des externen Speichers an derselben Stelle im Cache-Speicher abgelegt werden muß für jeden Cache-Block auch festgehalten werden, welchem Block im externen Speicher er entspricht. Der Speicher zur Verwaltung des Cache-Zustandes wird als Tag-RAM bezeichnet. Da bei Zugriffen auf eine zweidimensionale Nachbarschaft auf bis zu vier benachbarte Cache-Blöcke zugegriffen wird, sind die Verwaltungsdaten auf vier Tag-RAMs verteilt, so daß der Cache-Controller parallel zugreifen kann.

Da Cache-Zugriffe sehr schnell erfolgen müssen und gleichzeitig viele Operationen bei einem Zugriff durchgeführt werden müssen, ist der Cache-Controller als Pipeline aufgebaut. Wie in Bild 6.21 gezeigt, handelt es sich um eine dreistufige Pipeline. Während des ersten Taktes werden die Blocknummern anhand des Plazierungsschemas berechnet, im darauffolgenden Takt erfolgt das Auslesen der Tag-RAMs, die Prüfung, ob alle benötigten Blöcke bereits geladen sind und die Adressierung des Cache-Speichers. Im letzten Takt werden die Daten schließlich in die Registermatrix geladen.

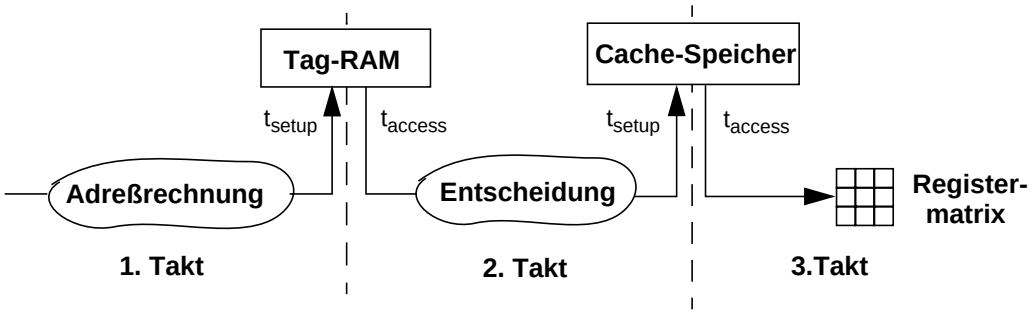


Bild 6.21: Pipeline-Struktur des Cache-Controllers

Plazierungsschemata

Das Plazierungsschema legt fest, welche Blöcke des Bildbereiches an welchen Stellen des Caches abgelegt werden. Das Plazierungsschema muß sich - neben den in Kapitel 6.1.2.1.2.2 dargestellten Anforderungen - schnell berechnen lassen und für die zu verarbeitenden Bildformate soll eine möglichst gute Nutzung des Cache-Speichers erfolgen, d.h. möglichst kein Verschnitt entstehen. Je nach eingesetzter Adressierungsart müssen ein Bild (rekursive Intra/ Segment-Adressierung), zwei Bilder (nichtrekursive Intra/Segment-Adressierung) bzw. drei Bilder (Inter-Adressierung) im Cache verwaltet werden. Um den vorhandenen Cache-Speicher optimal zu nutzen, werden daher drei unterschiedliche Platzierungsschemata verwendet, die jeweils zwei, drei bzw. sechs Zeilen von Cache-Blöcken einem Bild zuordnen.

Das Plazierungsschema wird dargestellt indem die Cache-Blocknummer über dem Bildbereich aufgetragen wird. Bild F.1 im Anhang zeigt das Plazierungsschema bei Verwaltung dreier Bilder. Um auf die Verwaltungsinformation von 2x2 Blöcken parallel zugreifen zu können, sind diese

jeweils vier unterschiedlichen Tag-RAMs zugeordnet, wie durch die nicht-umkreisten Zahlen t_b angezeigt wird. Die umkreisten Zahlen t_n geben an, bei welcher Adresse im Tag-RAM sich die zugehörigen Verwaltungsinformationen befinden. Die Nummer des Cache-Blockes berechnet sich zu

$$n_b = t_n \cdot 4 + t_b \tag{6.6}$$

Dadurch, daß die Blöcke innerhalb von zwei Zeilen fortlaufend anderen Cache-Blöcken zugeordnet werden und die Zuordnung von Zeile zu Zeile um 2 Blöcke versetzt ist, wird die Anforderung bezüglich gleichzeitiger Speicherung zweier Zeilen bzw. Spalten erfüllt. Dadurch, daß t_n von 0 bis 87 läuft, tritt kein Verschnitt auf. Die Werte von t_n und t_b berechnen sich aus den Koordinaten x, y und der Bildnummer i wie folgt, wobei „%“ die Modulo-Operation, „>>“ bitweises Schieben und „&“ die Und-Verknüpfung bezeichnet:

$$t_b = (x\&1) + 2 \cdot (y\&1) \tag{6.7}$$

$$a = ((x>>3) + (y>>3))\&255 \tag{6.8}$$

$$t_n = 8 \cdot ((a>>3)\%11) + (a\&7) + i \cdot 88 \tag{6.9}$$

Wie aus Gleichung (6.7) bis (6.9) ersichtlich ist, erfordert die Berechnung von t_n und t_b lediglich zwei Additionen und eine Modulo-11-Operation für eine 5 bit Zahl, die durch Nachschlagen in einer Tabelle implementiert werden kann. Beim in Bild F.2 gezeigten Plazierungsschema für die Verwaltung zweier Bilder berechnet sich t_b wie in Gleichung (6.7), für t_n ergibt sich

$$b = (y>>2)/3 + (x>>4) \tag{6.10}$$

$$t_n = 4 \cdot ((b>>2)\%11) + (b\&3) + ((y>>2)\%3) \cdot 44 + i \cdot 132 \tag{6.11}$$

Beim in Bild F.3 gezeigten Plazierungsschema für die Verwaltung eines Bildes ergibt sich:

$$c = (x>>3) + ((y>>3)/3)\&31 \tag{6.12}$$

$$t_n = 8 \cdot ((c>>3)\%11) + (c\&7) + ((y>>3)\%3) \cdot 88 \tag{6.13}$$

Da die Implementierung der Platzierungsschemata sehr zeitkritisch ist, wurden die drei Platzierungsschemata getrennt implementiert.

6.2.3.3.2 Transfer-Controller

Der Transfer-Controller übernimmt auf Anforderung des Cache- bzw. des Prefetch-Controllers die Steuerung von Laden bzw. Zurückschreiben von Cache-Blöcken vom/zum Hauptspeicher. Mittels Konfiguration werden für die verarbeiteten Bilder die Parameter festgelegt, die deren Lage im externen Speicher bestimmen, d.h. Banknummer, Startadresse und Bildbreite. Beim Laden eines Cache-Blocks errechnet der Transfer-Controller die Zeilenadresse entsprechend der Anforderung des Cache-Controllers bzw. Prefetch-Controllers. Sobald der Zugriff auf den Hintergrundspeicher möglich ist, wird ein Burst-Read Kommando ausgegeben und nach Ablauf der Zugriffszeit ohne weitere Kommandos in jedem zweiten Takt ein Datum bestehend aus zwei Bildpunkten vom Speicher gelesen (da der externe Speicher mit der halben Taktfrequenz des Koprozessors betrieben wird). Parallel werden Blockadresse, Bildpunktadresse und Steuersignale für den Cache-Speicher entsprechend dem Fortschritt der Ladeoperation generiert und die Weiterleitung der Daten in Abhängigkeit des Blocktyps (links/rechts) gesteuert.

Um die Verzögerung bei Verdrängung eines veränderten Blockes so gering wie möglich zu halten, existiert ein Kommando für kombiniertes Neuladen *und* Zurückschreiben. Dadurch wird ermöglicht, den Inhalt des zu verdrängenden Cache-Blocks in einem Schreibpuffer zwischenzu-

speichern und gleichzeitig die zu ladenden Daten vom externen Speicher anzufordern. Da vom externen Speicher lediglich jeden zweiten Takt Daten geliefert werden, können Zwischenspeicherung und Laden abwechselnd erfolgen, so daß die beim Laden überschriebenen Daten zuvor im Zwischenspeicher gesichert werden. Anschließend wird der Zwischenspeicher parallel zur fortgesetzten Verarbeitung zurückgeschrieben, so daß bei dieser Operation das Laden eines Cache-Blockes genauso schnell abgeschlossen ist, wie bei einer einfachen Ladeoperation.

Neben dem Transfer zweidimensional organisierter Blöcke ist für die Bearbeitung von Histogrammen eine Betriebsart vorgesehen, in der eindimensional organisierte Daten übertragen werden. Der Transfer-Controller unterstützt außerdem Über- und Unterabtastung im Verhältnis 1:2 bzw. 2:1 beim Laden der Bilddaten, wobei jeweils in x- bzw. in y-Richtung oder in beide Richtungen skaliert werden kann (siehe Bild 6.22). Da dabei nicht alle Punkte eines Blocks benötigt werden, werden die erforderlichen Bildpunkte mit einzelnen Lesekommandos aus dem externen Speicher geladen. Unterabtastung wird nur für Cache-Blöcke unterstützt, die innerhalb einer Seite liegen, da ansonsten Seitenwechsel die Adressierung verkomplizieren würden.

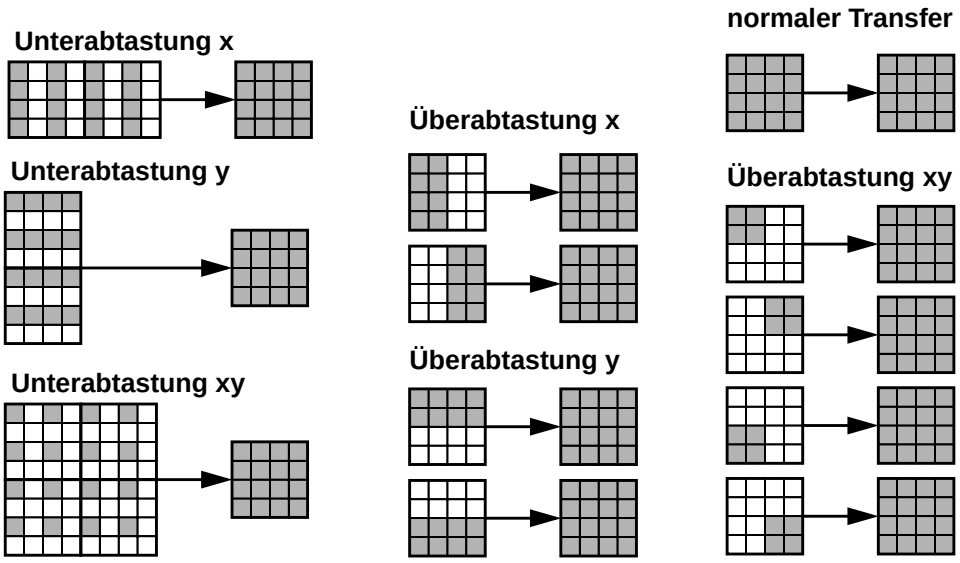


Bild 6.22: Über- und Unterabtastung von Bilddaten

Bei Operationen mit Segment-Adressierung muß der Verarbeitungsstatus, der in der Marker-Komponente gespeichert ist, vor Operationsbeginn initialisiert werden. Um zu vermeiden, daß hierfür das Eingangsbild gelesen, modifiziert und geschrieben werden muß, wird stattdessen die Gültigkeit des Verarbeitungsstatus beim Laden eines Blockes geprüft. Wenn er sich nicht als gültig erweist, wird der Verarbeitungsstatus während des Ladevorgangs initialisiert. Um diese Prüfung durchführen zu können, muß für jeden Block gespeichert sein, durch welche Operation er zuvor geschrieben wurde. Die Prüfung, ob der Verarbeitungsstatus gültig ist, entspricht damit der Prüfung, ob der Block von der aktuellen Operation geschrieben wurde. Da pro Bildpunkt 6 bit nicht für Bilddaten verwendet werden, können pro Cache-Block 96 bit zur Speicherung der Operationsnummer verwendet werden, die nach Abschluß jeder Operation um eins erhöht wird. Ein Zählerüberlauf tritt aufgrund der großen Wortbreite nicht auf.

6.2.3.3.3 Prefetch-Controller

Der Prefetch-Controller steuert das Vorladen und das Zurückschreiben von Cache-Blöcken bei regulärer Adressierung, um Cache-Misses bzw. die Verdrängungen von Blöcken zu vermeiden. Er erhält vom Scan-Controller Informationen über den Fortschritt der durchgeführten Verarbeitung. Entsprechend positionsabhängiger Sollwerte für Vorladen bzw. Zurückschreiben steuert der

Prefetch-Controller den Transfer-Controller und ändert nach Abschluß eines Transfers den entsprechenden Eintrag im Tag-RAM. Bei Segment-Adressierung wird Vorladen während der Startpunktsuche verwendet und während des Ausbreitungsprozesses abgeschaltet. Cache-Misses haben Vorrang gegenüber den Anforderungen des Prefetch-Controllers, da sie aktuelle Anforderungen von Daten darstellen.

6.2.3.3.4 SI-Controller

Der SI-Controller lädt und speichert segmentindizierte Daten bzw. Histogramme auf Anforderung der Verarbeitungseinheit. Ebenso wie der Transfer-Controller generiert der SI-Controller Zeilen- und Spaltenadressen und greift über den Arbiter auf den Hintergrundspeicher zu. Aufgrund der zur Verfügung stehenden Busbreite wird ein segmentindizierter Datensatz in mehrere Kanalbündel aufgeteilt, wie in Anhang A gezeigt. Bei der Anforderung *eines* Histogrammeintrags werden - um die Busbreite zum Hintergrundspeicher auszunutzen - gleichzeitig acht Einträge geladen. Durch das Pufferregister wird ein Großteil der Lese- und Schreiboperationen abgefangen. Um die Wartezeit für die Verarbeitungseinheit zu verkürzen, wird der Inhalt des Pufferregister vor dem Zurückschreiben in einem Schreibpuffer zwischengespeichert, so daß zuerst das Laden und anschließend das Abspeichern erfolgen kann.

6.2.3.3.5 Arbiter

Neben dem Transfer-Controller greift auch der SI-Controller auf den externen Speicher zu. Um Zugriffskonflikte zu lösen, wird der externe Speicher von einem Arbiter verwaltet, der die Anforderungen entgegennimmt und die Zugriffe entsprechend der Priorität der Anforderungen gestattet. Da eine Verzögerung von Zugriffen auf segmentindizierte Daten einen Leerlauf der Verarbeitungseinheit zur Folge hat, haben Anforderungen des SI-Controllers Priorität. Falls der Transfer-Controller während der Anforderung eines segmentindizierten Zugriffes einen Transfer durchführt, muß es sich um einen vom Prefetch-Controller initiierten Transfer handeln, der wenig kritisch für die Verarbeitungsleistung des Systems ist.

Neben der Ressourcenvergabe setzt der Arbiter die Anforderungen von Transfer- und SI-Controller in Befehle an das SDRAM um. Er verwaltet den Status der Speicherbänke, und erteilt gegebenenfalls Kommandos zum Öffnen (row activate) und Schließen (precharge) einer Seite, bevor die angeforderten Lese- und Schreib-Anforderung ausgeführt werden. Außerdem überwacht der Arbiter die Zeit bis zum nächsten Auffrischen (Refresh) und die Zeit, die eine Seite maximal geöffnet sein darf.

7. Ergebnisse und Bewertung

In diesem Kapitel werden die Ergebnisse der HW-Synthese und der Architektursimulation der CONIAN-Architektur vorgestellt. Zur Bewertung der Kosten für die Implementierung wird die Logik-, Verdrahtungs- und Speicher-Fläche bestimmt. Ausgehend von den Verarbeitungsdauern der Bildpunktoperationen wird die Systemgeschwindigkeit für verschiedene Benchmark-Operationen simuliert. Zum Beleg, daß sich mit der vorgeschlagenen Architektur Segmentierungsalgorithmen in Echtzeit ausführen lassen wird eine Simulation des in Kapitel 2.6 untersuchten Farbsegmentierungsalgorithmus durchgeführt. Im Anschluß erfolgt ein Vergleich mit der Geschwindigkeit und dem Flächenbedarf eines Universalprozessors bzw. anderer Hardware-Architekturen, und eine Bewertung der CONIAN-Architektur.

Des weiteren wird in diesem Kapitel die Einsetzbarkeit der Simulationsmethodik untersucht. Zunächst wird die Simulationsgeschwindigkeit für das Fallbeispiel CONIAN bestimmt und in Relation zur Architekturgröße gesetzt, anschließend wird der Einsatzbereich und der Nutzen der Methodik diskutiert.

7.1 Ergebnisse für die CONIAN-Architektur

7.1.1 Flächenbedarf

Module der CONIAN-Architektur wurden in der Hardware-Beschreibungssprache VHDL auf RTL-Ebene modelliert und die Korrektheit durch Simulation typischer und extremer Betriebsfälle verifiziert. Zusätzlich erfolgte eine Verifikation der Konfiguration der Verarbeitungseinheit durch Vergleich der Verarbeitungsergebnisse des Hardware-Modells mit denen des Algorithmenmodells als Referenz. Der Flächenbedarf wurde durch Synthese der RTL-Modelle ermittelt. Die Verarbeitungseinheit, die Adreßeinheit und die Registermatrix wurden mit der 0,25 µm Technologiebibliothek HCMOS7 von ST Microelectronics synthetisiert [HCMOS7], für den Pixel-Level- und den Scan-Controller wird auf Syntheseergebnisse mit der 0,25 µm Technologiebibliothek CB-C10 von NEC [CBC10] zurückgegriffen. Die Syntheseergebnisse sind in Tabelle 7.1 gezeigt, für den HLIFO-Controller, den SI-Controller und den Prefetching-Controller, die nicht in der Flächenangabe für die Adreßeinheit enthalten sind, sind Schätzungen für eine 0,25 µm Technologiebibliothek angegeben.

Modul	Logikfläche	Verdrahtung
Verarbeitungseinheit	1,84 mm^2	1,87 mm^2
Adreßeinheit	1,19 mm^2	1,19 mm^2
Registermatrix	0,48 mm^2	0,63 mm^2
Pixel-Level-Controller	0,21 mm^2	0,21 mm^2
Scan-Controller	0,22 mm^2	0,22 mm^2
HLIFO-Controller	0,13 mm^2	0,13 mm^2
SI-Controller und Prefetching-Controller	0,8 mm^2	0,8 mm^2
Gesamt	**4,87 mm^2**	**5.05 mm^2**

Tabelle 7.1: Logik und Verdrahtungsfläche der CONIAN-Architektur

Für das größte Modul, die Verarbeitungseinheit, wurde eine Layout-Synthese durchgeführt, die auch die Verdrahtung beinhaltet. Für die Signalleitungen, Scan-Pfad, Taktbaum und Versorgungsleitungen umfassende Verdrahtungsfläche ergab sich etwa derselbe Wert, wie für die Logikfläche. Die Verdrahtungsfläche für die übrigen Module wurde somit auf 100% der jeweiligen Logikfläche geschätzt, abgesehen von der Registermatrix, für die aufgrund der zahlreichen Verbindungspfade ein Routing der Signalleitungen zu einer Abschätzung des Verdrahtungsaufwand von 130% führte.

Zur Chipfläche für Logik und Verdrahtung kommt die Chipfläche für den internen Speicher hinzu. Diese wurden mit einem RAM-Compiler mit der DRP2- bzw. SPS3-Charakteristik [DPR2],[SPS3] für die 0,25µm Technologiebibliothek HCMOS7 erzeugt. Bei einer Dimensionierung des Speichers für das Bildformat 4CIF ergeben sich die in Tabelle 7.2 gezeigten Flächen. Die ersten drei Zeilen umfassen die Speicherblöcke für den Cache (A,Y,U,V,AX-Komponenten sind von der M-Komponente getrennt) und für das Tag-RAM, die folgenden drei Zeilen umfassen die Speicherblöcke für den Koordinatenspeicher.

Speicher	Größe	Fläche	Charakteristik
Cache-Speicher und Tag-RAM	8 x (2112x56)	8 x 4,116 mm^2	DPR2
	8 x (2112x2)	8 x 0,338 mm^2	DPR2
	4 x (264x20)	4 x 0,143 mm^2	SPS3
Koordinaten-speicher	2 x (16384x15)	2 x 7,403 mm^2	DPR2
	4 x (16384x12)	4 x 6,003 mm^2	DPR2
	2 x (256x15)	2 x 0,107 mm^2	SPS3
Gesamt	**279,1 kByte**	**75,24 mm^2**	

Tabelle 7.2: Größe und Flächenbedarf der chipinternen Speicher

Während sich für die Logik insgesamt ein Flächenbedarf von 9,92 mm^2 ergibt, überwiegt der Flächenbedarf für die Speicherblöcke mit 75,24 mm^2. Aus diesem Grund fällt die Ungenauigkeit in der Flächenberechnung, bedingt durch die Verwendung von Schätzwerten bzw. Technologiebibliotheken unterschiedlicher Hersteller nicht ins Gewicht.

Wenn die CONIAN-Architektur zur Verarbeitung von Bildern mit kleinerem Format eingesetzt wird, kann die Speichergröße, wie in Tabelle 7.3 angegeben, reduziert werden. Aufgrund der erreichbaren Verarbeitungsleistung ist eine Architektur, die die Bildformate CIF und QCIF unterstützt, von besonderem Interesse. Bei dieser Dimensionierung beträgt die Chipfläche der CONIAN-Architektur 47,65 mm^2.

Bildformat	Speichergröße	Fläche	
		Speicher	Gesamt
4CIF	**279,1 kByte**	**75,24 mm^2**	**85,16 mm^2**
CIF	**140,0 kByte**	**37,73 mm^2**	**47,65 mm^2**
QCIF	**70,5 kByte**	**18,97 mm^2**	**28,89 mm^2**

Tabelle 7.3: Größe und Flächenbedarf bei verschiedenen Bildformaten

7.1.2 Verarbeitungsdauer der Bildpunktoperationen

Die maximale Taktfrequenz wird durch das langsamste Modul bestimmt. Sie wurde durch eine statische Timing-Analyse ermittelt, die im Zusammenhang mit der Synthese erfolgte. Während einige Module eine höhere Taktfrequenz zulassen würden, wie z.B. der Pixel-Level-Controller 300 MHz, oder der Scan-Controller 222 MHz, erreicht die Adreßeinheit maximal 200 MHz, da diese am kritischsten hinsichtlich der Laufzeit ist und bei der Synthese entsprechende Vorgaben (Constraints) gemacht wurden. Übliche SDRAMs [Inf99] bzw. SRAMs [Sam99] lassen sich jedoch nicht mit einer Taktfrequenz von 200MHz ansteuern. Da der Takt der Speicherschnittstelle ein ganzzahliges Vielfaches des Systemtaktes betragen sollte und da sich beim Einsatz von SDRAM bei 100 MHz gespeicherte Daten mit einer deutlich kürzere Pipelineverzögerung lesend zugreifen lassen als bei dem maximalen Takt von 166 MHz [Inf99], wird der Takt der Speicherschnittstelle zu 100 MHz festgelegt.

Die Geschwindigkeit der Verarbeitungseinheit hat keinen Einfluß auf die Taktfrequenz, da die Bildpunktoperationen als Multizyklus-Operationen implementiert sind, so daß die Verarbeitungsdauer die jeweils erforderliche Anzahl an Taktzyklen bestimmt. In Tabelle 7.4 ist die Verarbeitungsdauer ausgewählter Operationen bei Synthese der Verarbeitungseinheit mit der Technologiebibliothek HCMOS7 ohne Vorgaben gezeigt. Die Anzahl der benötigten Taktzyklen ergibt sich aus der maximalen Taktfrequenz von 200 MHz und der Dauer der jeweiligen Operation plus 1,5 ns für die externe Verzögerung an den Ein- und Ausgängen. Die zugehörigen Werte sind in Tabelle 7.4 in der linken der drei Spalten angegeben. Außer der Relaxation, die mit 7 Takten am längsten dauert, benötigt keine der Operationen länger als 5 Takte.

Operation	Verarbeitungsdauer	Taktzyklen		
Laplace-Operator	14,98 ns	4	3	3
Sobel-Operator	18,64 ns	5	4	3
3x3 FIR-Filter	14,98 ns	4	3	3
Erosion / Dilatation	11,95 ns	3	3	2
Median-Filter	11,61 ns	3	3	2
Relaxation	32,91 ns	7	6	5
Histogramm	4,09 ns	2	1	1
Schwellwert	6,08 ns	2	2	2
Skelettierung	1,53 / 6,79 ns	1/2	1/2	1/2
zush. Bereiche bestimmen	4,21 / 14.03 ns	2/4	1/3	1/3
Fragmentierung prüfen	2,97 / 4,56 ns	1/2	1/2	1/1
Wasserscheide-Verfahren	2,38 / 18,32 ns	1/4	1/4	1/3

Tabelle 7.4: Verarbeitungsdauer ausgewählter Operationen und korrespondierende Taktanzahl bei 200 MHz, externe Eingangs-/Ausgangsverz. = 1,5ns, Optimierungsgrad = 1 / 1,25 / 1,5

Um die Architektur mit Universalprozessoren vergleichen zu können, wurde die Zeitdauer bei Charakterisierung für typische Verzögerungszeiten benutzt (siehe Kapitel 7.2.2). Für den Vergleich mit anderen Implementierungen ist zu beachten, daß die Bildpunktoperationen parallel auf Y-, U- und V-Komponente angewandt werden und mit einer CON8 Nachbarschaft arbeiten. Bei

auf Segment-Adressierung basierenden Operationen sind in der Tabelle zwei Werte angegeben. Der linke der durch Schrägstriche getrennten Werte bezieht sich auf die Startpunktsuche, der rechte bezieht sich auf den Ausbreitungsprozeß.

Werden bei der Synthese Vorgaben gemacht, läßt sich damit ein Geschwindigkeitsgewinn zugunsten einer größeren Fläche erreichen. Da die Latenz der Verarbeitungseinheit für die bei verschiedenen Bildpunktoperationen konfigurierten Pfade minimiert werden soll, ist die Erstellung von Vorgaben für die Teilmodule aufwendig, möglicherweise ist sogar ein iteratives Vorgehen erforderlich. Um diesen Aufwand zu vermeiden wurde lediglich abgeschätzt, welche Auswirkung eine Optimierung mit dem als

$$O = \frac{t_{\text{Verarbeitung, ohne Vorgaben}}}{t_{\text{Verarbeitung, mit Vorgaben}}} \tag{7.1}$$

definierten Optimierungsgrad hätte. Der Optimierungsgrad, der die Grenze zwischen einer Dauer von n und n+1 Takten darstellt, läßt sich durch Auflösen von Gleichung (7.2) nach O berechnen. Die Ergebnisse sind graphisch in Bild 7.1 dargestellt.

$$(t_{\text{Verarbeitung, ohne Vorgaben}} / O) + t_{\text{ext}} = n \cdot t_{\text{clk}} \tag{7.2}$$

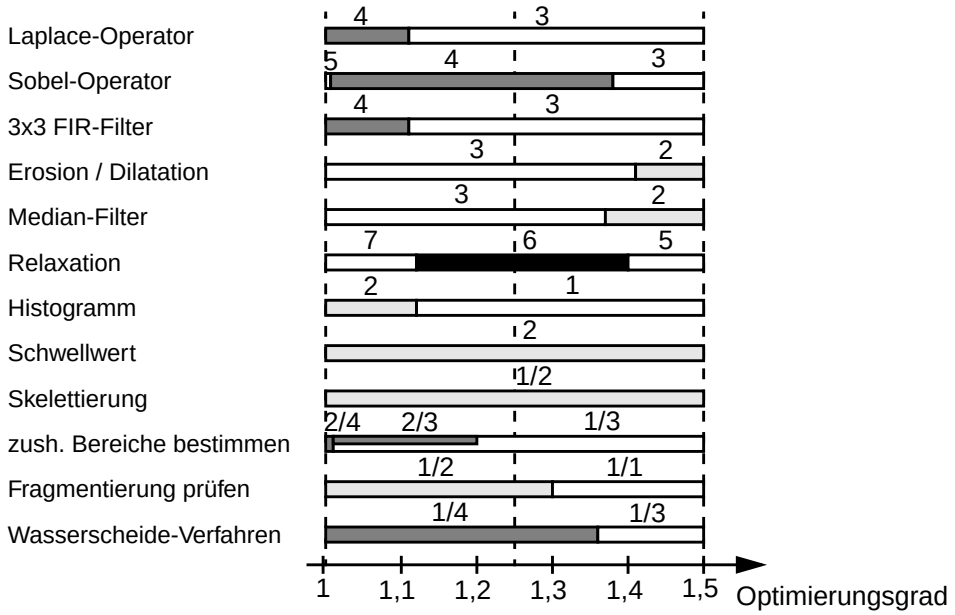


Bild 7.1: Abhängigkeit der Anzahl an Takten vom Optimierungsgrad

7.1.3 Systemgeschwindigkeit

Für einen Vergleich mit anderen Architekturen ist die Dauer von Bild- bzw. Segmentoperationen von Bedeutung. Diese wird nicht durch die Verarbeitungsdauer für eine Bildpunktoperationen sondern durch die Verarbeitungsgeschwindigkeit des Gesamtsystems bestimmt.

7.1.3.1 Benchmark-Operationen

Für den Vergleich der Geschwindigkeit von Prozessoren bzw. Architekturen werden oftmals Benchmark-Operationen eingesetzt. Dabei handelt es sich um bestimmte Algorithmen oder Algorithmenteile für die die Ausführungsgeschwindigkeit ermittelt und verglichen wird. Bei Architekturen für Bildverarbeitung werden hierzu oftmals Bild- bzw. Segmentoperationen eingesetzt. Dies hat außerdem den Vorteil, daß Architekturen, die einen unterschiedlichen Satz an

Operationen umfassen, bezüglich gemeinsamer Operationen verglichen werden können. Für die CONIAN-Architektur ergeben sich die in Tabelle 7.5 gezeigten Werte.

Benchmark-Operation	QCIF	CIF
Laplace-Operator	0,51 ms	2,03 ms
Sobel-Operator	0,63 ms	2,53 ms
3x3 FIR-Filter	0,51 ms	2,03 ms
Erosion / Dilatation	0,38 ms	1,52 ms
Median-Filter	0,38 ms	1,52 ms
Relaxation	0,92 ms	3,60 ms
Histogramm	0,46 ms	1,48 ms
Schwellwert	0,26 ms	1,02 ms
Skelettierung	0,29 ms	1,14 ms
zush. Bereiche bestimmen	0,58 ms	3,60 ms
Fragmentierung prüfen	0,98 ms	3,92 ms
Wasserscheide-Verfahren	2,00 ms	8,56 ms

Tabelle 7.5: Dauer ausgewählter Benchmark-Operationen auf der CONIAN-Architektur

7.1.3.2 Farbsegmentierungsalgorithmus

Mit der in Kapitel 5 beschriebenen Simulationsmethodik wird im Folgenden der LIS Farbsegmentierungsalgorithmus (siehe Kapitel 2.6) untersucht. Dabei wurden die Simulationen für die MPEG-4 Testsequenzen „Bream“, „Coastguard“, „Foreman“, „Mother&Daughter“ und die Bosch-Testsequenz „S1“ im Bildformat QCIF durchgeführt.

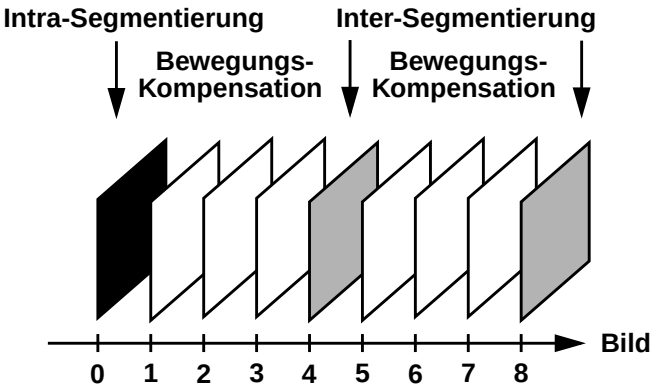


Bild 7.2: Unterschiedliche Verarbeitung der zu segmentierenden Bilder

Beim ersten Bild erfolgt eine Intra-Segmentierung, da dort nicht auf die Segmentmaske eines vorherigen Bildes zurückgegriffen werden kann. Um eine hohe Verarbeitungsgeschwindigkeit zu erreichen, wird für die drei folgenden Bilder die Segmentmaske durch Bewegungskompensation der Segmentmaske des vorherigen Bildes erzeugt. Im vierten Bild erfolgt nach der Bewegungs-

kompensation der Segmentmaske eine Inter-Segmentierung, d.h. die Segmentmaske wird unter Berücksichtigung des Bildinhaltes korrigiert, um Segmentkonturen anzupassen, Segmente zu löschen bzw. neue zu identifizieren. Bei guter Qualität der Bewegungskompensation ergibt sich der in Bild 7.2 gezeigte Ablauf, ansonsten wird für die jeweiligen Bilder Inter-Segmentierung anstelle der Bewegungskompensation verwendet.

In Bild 7.3 ist die Anzahl an Taktzyklen pro Bild für die Foreman Sequenz dargestellt. Man erkennt die im Mittel 3,2-fach höhere Anzahl an Taktzyklen für die Intra-Segmentierung gegenüber der Inter-Segmentierung. Für die Kompensation der Segmentmaske sind vergleichsweise wenige Taktzyklen erforderlich. Die Geschwindigkeit für den Farbsegmentierungsalgorithmus ergibt sich im eingeschwungenen Zustand, d.h. ohne Berücksichtigung der zur Initialisierung nötigen Intra-Segmentierung, zu 49,2 Bilder pro Sekunde.

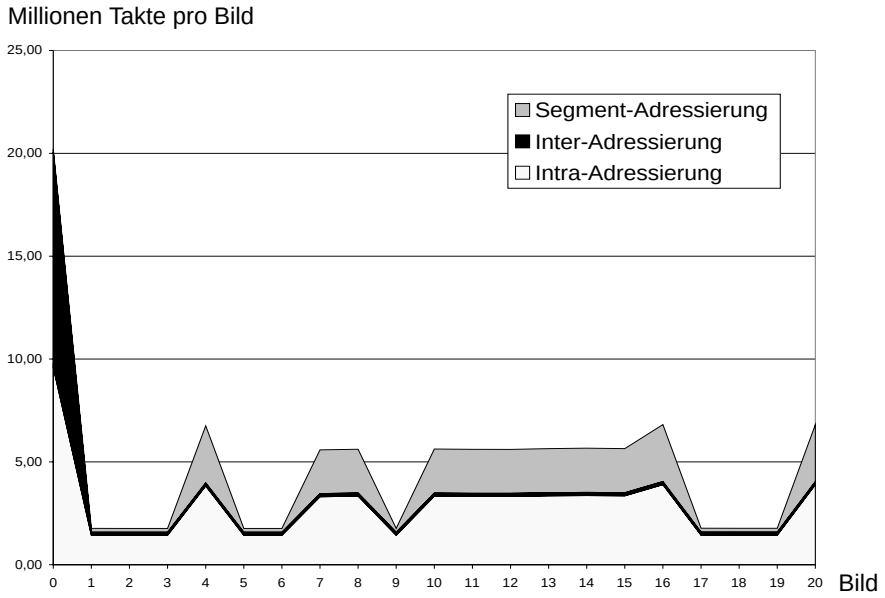


Bild 7.3: Taktzyklen pro Bild aufgeschlüsselt nach verwendeter Adressierungsart

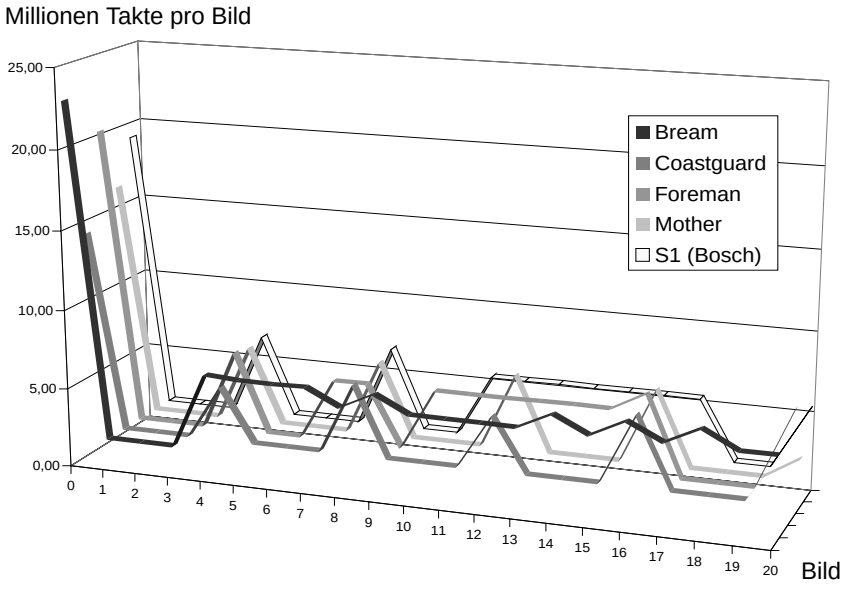


Bild 7.4: Taktzyklen pro Bild verglichen für verschiedene Sequenzen

In Bild 7.4 wird die Verarbeitungsdauer für die genannten Sequenzen verglichen. Da der Algorithmus den Einsatz der Inter-Segmentierung aufgrund der Qualität der Bewegungskompensation steuert, ergeben sich starke Schwankungen. Für eine Echtzeit-Anwendung muß auch die benötigte Rechenleistung zur Steuerung herangezogen werden. Mit den in Tabelle 7.6 angegebenen Werten für die maximale Dauer der Bewegungskompensation bzw. der Inter-Segmentierung bei den in Bild 7.4 verglichenen Sequenzen, läßt sich die erreichbare Verarbeitungsgeschwindigkeit zu 65,9 Bilder pro Sekunde abschätzen, wenn ausschließlich für jedes 4. Bild eine Inter-Segmentierung erfolgt, bzw. zu 29,3 Bilder pro Sekunde, wenn die Inter-Segmentierung für jedes Bild erfolgt.

	Bewegungs-kompensation	**Inter-Seg-mentierung**	**Intra-Seg-mentierung**
Minimum	$1{,}73 \cdot 10^6$	$3{,}10 \cdot 10^6$	$14{,}17 \cdot 10^6$
Mittelwert	$1{,}74 \cdot 10^6$	$5{,}81 \cdot 10^6$	$18{,}50 \cdot 10^6$
Maximum	$1{,}77 \cdot 10^6$	$6{,}82 \cdot 10^6$	$22{,}86 \cdot 10^6$

Tabelle 7.6: Taktzyklen pro Bild unterteilt nach Art der Verarbeitung des Bildes

7.2 Vergleich und Bewertung

7.2.1 Vergleich mit Software-Implementierung

Für den Vergleich mit einer Software-Implementierung auf einem Universalprozessor, wird ein Intel Celeron betrachtet, da die im Celeron enthaltene IA32, P6 Architektur wie der CONIAN mit 0,25 µm Strukturgröße entworfen wurde [ScWa98].

Die Simulation der Software-Implementierung erfolgte durch Simulation und Profiling des mit der AddressLib implementierten Farbsegmentierungsalgorithmus. Als Simulationsplattform wurde ein Celeron benutzt, der mit 466 MHz getaktet ist. Dadurch entspricht die simulierte Geschwindigkeit in etwa der Geschwindigkeit der IA32, P6 Architektur mit 0,25 µm Strukturgröße, die mit maximal 450 MHz getaktet werden kann [ScWa98].

Simulation und Profiling des optimierten Codes (Compiler gcc, Option -O3) ergaben für Bild 0 bis 20 der Foreman-Sequenz eine Simulationsdauer von 46 s[1]. Durch den Einsatz einer generischen Bibliothek für die Adressierung der Bilddaten ergibt sich gegenüber einer optimierten Software-Implementierung ein erheblicher Zusatzaufwand. Dieser ist bei Inter-Adressierung am größten, und bei Segment-Adressierung am kleinsten. Bei Intra-Adressierung ergab ein Vergleich verschiedener Bildoperationen, daß bei einer optimierte Implementierung bis zu 75% weniger Befehle ausgeführt werden. Da die Inter-Adressierung im Algorithmus nur selten verwendet wird und durch die Segment-Adressierung ausgeglichen wird, liegt auch der gesamte Zusatzaufwand für die Implementierung mittels AddressLib bei maximal 75%, was einer Nettodauer von 11,5 s entspricht.

Die Ausführungsdauer auf dem CONIAN liegt bei 0,48 s, was gegenüber der Nettodauer einer Beschleunigung um den Faktor 24,0 entspricht. Da bei der Implementierung die Multimediaerweiterungen des Celeron nicht genutzt wird, wird im Folgenden abgeschätzt, welches zusätzliche

1. Die Dauer der High-Level-Operationen, die nicht vom CONIAN implementiert wurden, wurde bereits abgezogen

Beschleunigungspotential sich damit maximal ergeben würde.

Es wird davon ausgegangen, daß bei einer optimierten Implementierung die gesamte Verarbeitung bei regulärer Adressierung durch MMX beschleunigt werden kann. Die Verarbeitung besitzt 60% Anteil an der Gesamtrechenleistung. Bei einem Anteil der regulären Adressierungsarten von 75% ergibt sich, daß 45% des Programmes durch MMX beschleunigt werden können. Wenn man annimmt, daß eine vollständige Parallelisierung um den Faktor 8 möglich ist, ergibt sich eine Nettodauer von 7 s.

Tabelle 7.7 zeigt einen Überblick, über die mit dem Celeron bzw. dem CONIAN erforderliche Verarbeitungsdauer. Ein Pentium IV mit 2 GHz besitzt einen um den Faktor 4,3 schnelleren Takt, bei dem SPEC CINT2000 Benchmark [SPEC], erreicht er aber nur eine um ca. Faktor 3,2 schnellere Verarbeitung. Unter der Annahme, daß diese Beschleunigung auch für den Segmentierungsalgorithmus erreichbar ist, ergeben sich die weiteren in Tabelle 7.7 gezeigten Werte.

Trotz der erreichbaren Beschleunigung ist der Flächenbedarf für den CONIAN mit insgesamt 47,7 mm^2 deutlich kleiner als für einen Celeron, der eine Fläche von 131 mm^2 aufweist, zu der außerdem der 128kByte große externe Cache hinzugerechnet werden muß[1]. Da die Verlustleistung durch eine Reduzierung der Versorgungsspannung abgesenkt werden kann, wenn die für eine Berechnung nötige Taktfrequenz geringer ist, wirkt sich die gegenüber dem Celeron bzw. Pentium IV deutlich niedrigere Taktfrequenz des CONIAN vorteilhaft für mobilen Einsatz aus.

Architektur	Taktfrequenz	Dauer
Celeron	466 MHz	11,5 s
Celeron mit MMX	466 MHz	7,0 s
Pentium IV	2 GHz	3,6 s
Pentium IV mit MMX	2 GHz	2,2 s
CONIAN	200 MHz	0,48 s

Tabelle 7.7: Vergleich der Verarbeitungsdauer des CONIAN mit dem Celeron / Pentium IV

7.2.2 Technologieskalierung

Da veröffentlichte Architekturdaten in der Regel auf unterschiedlichem Stand der Halbleitertechnologie beruhen, müssen bei einem Vergleich, der auf das Potential einer Architektur abzielt, die Einflüsse von Technologie und Entwurfswerkzeugen von den Eigenschaften der Architektur getrennt werden. Da dies jedoch nur bedingt möglich ist, sei bereits an dieser Stelle darauf hingewiesen, daß der anschließende Vergleich mit der erforderlichen Zurückhaltung interpretiert werden sollte.

Bei der Implementierung von Hardware-Architekturen als semi-custom ASIC stehen unterschiedlichste Technologien verschiedener Hersteller zur Verfügung. Bei der zuvor erfolgenden Synthese wird eine Bibliothek an Grundelementen benötigt (Standardzellenbibliothek), die sich je nach Hersteller und Technologie unterscheidet, außerdem werden verschiedene Entwurfswerkzeuge eingesetzt, deren Optimierungsalgorithmen sich unterscheiden und die in Abhängigkeit zusätzlicher Vorgaben (Constraints) andere Ergebnisse produzieren.

1. Grundsätzlich muß jedoch angemerkt werden, daß der CONIAN für den Einsatz zusammen mit einem Prozessor für die Durchführung der High-Level-Operationen konzipiert ist.

Im Bezug auf die verwendete Technologie üben folgende drei Faktoren Einflüsse auf Flächenbedarf, Geschwindigkeit und weitere Eigenschaften implementierter Architekturen aus:

- Strukturgröße
- Auslegung des Zeitverhaltens
- Versorgungsspannung

Die Strukturgröße ist die charakteristische Größe für die auf einem ASIC implementierten Strukturen. Typischerweise wird die gezeichnete Gate-Länge der Transistoren als Strukturgröße angegeben. Da jedoch manche Hersteller die effektive Gate-Länge als Strukturgröße angeben, ist ihr Wert nicht immer exakt vergleichbar.

Da sich bei der ASIC-Herstellung technologisch bedingte Schwankungen ergeben, schwankt auch die Geschwindigkeit, mit der diese betrieben werden können. Daher muß man entweder ausreichend Toleranzen vorsehen, so daß gegebene Anforderungen in jedem Fall erreicht werden, oder man muß nach der Herstellung nach erreichbarer Geschwindigkeit selektieren. Im ersten Fall muß das Zeitverhalten auf den ungünstigsten Fall (worst case) ausgelegt werden, während im zweiten Fall eine Auslegung auf den typischen Fall (nominal) möglich ist.

Alle drei genannten Faktoren haben einen Einfluß auf die Taktfrequenz. Entsprechend den Skalierungsregeln in [Rug99b] ist die durch Gatterlaufzeiten bedingte Verzögerungszeit umgekehrt proportional zur Versorgungsspannung und direkt proportional zum Quadrat der Strukturgröße. Gemäß [SIA97] kann von einer Skalierung zwischen den Technologiegenerationen ausgegangen werden, bei der die Versorgungsspannung proportional zur Strukturgröße sinkt, so daß das elektrische Feld konstant bleibt. Wenn man annimmt, daß in erster Näherung auch Leitungsverzögerungen proportional zur Strukturgröße sind, kann eine skalierte Frequenz

$$f_{skal} = f_x \cdot \frac{x}{0,25\mu m} \cdot \frac{1,4}{r} \tag{7.3}$$

angegeben werden kann. Die Strukturgröße wird dabei mit x bezeichnet, die gewählte Auslegung im Verhältnis zum Zeitverhalten im ungünstigsten Fall wird mit r bezeichnet. Mit der Änderungen der Strukturgröße gehen jedoch schwer vorhersehbare Effekte einher. Beispielsweise kann eine Änderung der Koppelkapazitäten zwischen Signalleitungen bzw. eine Änderung der Betriebsspannung bewirken, daß die Leitungsverzögerungen nicht exakt proportional zur Strukturgröße sind. Durch Angabe der skalierten Frequenz kann somit der Einfluß technologiebedingter Faktoren nur in begrenztem Maß beseitigt werden.

Der Einfluß der Strukturgröße auf die Chipfläche läßt sich durch Angabe der skalierten Fläche

$$A_{skal} = A_x \cdot \left(\frac{0,25\mu m}{x}\right)^2 \tag{7.4}$$

in erster Näherung beseitigen läßt, die gemäß Gleichung (7.4) auf die Strukturgroße 0,25 µm bezogen ist.

Standardzellenbibliotheken können sich je nach Hersteller, Strukturgröße und Version im Hinblick auf Aufbau und Verfügbarkeit von Bibliothekselementen unterscheiden. Aus diesen Gründen werden Architekturen durch Synthesewerkzeuge anders umgesetzt. Durch Synthesevorgaben (Constraints) kann auch die Gewichtung zwischen Flächenbedarf und Verarbeitungsgeschwindigkeit verschoben werden. Da sich jedoch diese Faktoren schlecht quantifizieren lassen, sind derartige Angaben in Publikationen praktisch nicht zu finden. Ein Vergleich von Architekturen ist somit selbst dann mit Vorsicht zu interpretieren, wenn beide mit gleicher Technologiegeneration, d.h. mit gleicher Strukturgröße umgesetzt wurden.

7.2.3 Vergleich mit anderen Architekturen

Ein aussagekräftiger Vergleich der Verarbeitungsleistung von Hardware-Architekturen ist nur anhand einer echten Implementierungen des Algorithmus mit den zu vergleichenden Architekturen möglich. Da jedoch für Verfahren zur Videosegmentierung kein Standard existiert, gibt es keinen kompletten Algorithmus für den Daten verschiedener Architekturen veröffentlicht sind. Daher wird an dieser Stelle auf veröffentlichte Daten über die Verarbeitungsdauer wichtiger Benchmark-Operationen zurückgegriffen.

Da sich die veröffentlichten Daten auf unterschiedliche Bildformate beziehen, werden sie anhand der Anzahl verarbeiteter Bildpunkte und Helligkeits-/Farbkomponenten auf das QCIF Bildformat bei Verarbeitung von 3 Komponenten (bzw. 1 Komponente im Fall der Histogramm Erzeugung) skaliert. Für einen exakten Vergleich wäre die Kenntnis der Verarbeitungsdauer für ein einheitliches Bildformat erforderlich.

In Tabelle 7.9 sind die veröffentlichten Daten bekannter Architekturen über die Verarbeitungsdauer wichtiger Bildoperationen und die zugehörige skalierte Verarbeitungsdauer angegeben. Dabei wurde die Bildgröße, Anzahl der Komponenten und die skalierte Taktfrequenz berücksichtigt. In Bild 7.5 ist die skalierte Verarbeitungsdauer im Verhältnis zur Verarbeitungsdauer des CONIAN graphisch dargestellt. Tabelle 7.8 zeigt die Implementierungseigenschaften der Architekturen.

Soweit die Daten einen Vergleich gestatten, zeigt sich, daß die CONIAN-Architektur in den meisten Fällen schneller arbeitet, als die verglichenen Architekturen. Beim HiPar gilt dies für die Operationen „3x3 FIR Filter“, „Skelettierung“ und „zusammenhängende Bereiche bestimmen“. Bei der Erzeugung von Histogrammen erreicht der CONIAN im Vergleich zu HiPar-4/16 eine deutlich geringere Geschwindigkeit. Bei Kantendetektion ist der HiPar-16 schneller als der CONIAN, während der HiPar-4 langsamer ist. In ähnlicher Weise verhält es sich mit dem PIMM-10 bei den Operationen Erosion und Dilatation. Bei den irregulären Operationen „Wasserscheide-Verfahren“ und „zusammenhängende Bereiche markieren“ ist jedoch der CONIAN deutlich schneller. Im Vergleich zu den FPGA basierten Architekturen Spash-2, Sonic-1 und MoM-PDA, sowie der Architektur [NoMe95] erweist sich der CONIAN in jedem Fall als deutlich schneller. Die Array-Architektur [Nog97] ermöglicht eine sehr schnelle Implementierung des Wasserscheide-Verfahrens. Allerdings ist in der dort angegebenen Zeit die Zu- und Abfuhr von Daten nicht enthalten. Außerdem handelt es sich um eine dedizierte Architektur, die lediglich das Wasserscheide-Verfahren durchführen kann und den 4,6-fachen Flächenbedarf wie die CONIAN-Architektur besitzt. Die HDPP-Architektur implementiert die Median-Filterung mit höherer Geschwindigkeit als der CONIAN. Für diese Architektur ist jedoch von einem noch höheren Flächenbedarf als für [Nog97] auszugehen, da ebensoviele Prozessorelemente erforderlich sind, diese jedoch aufgrund ihrer Programmierbarkeit einen höheren Platzbedarf erfordern dürften. Da bei Ausbreitungsprozessen die parallelen Prozessorelemente nur sehr schwach ausgelastet werden können, kann bei Operationen mit Segment-Adressierung nur eine geringe Geschwindigkeit der HDPP-Architektur erwartet werden.

Insgesamt erweist sich somit die Verarbeitungsgeschwindigkeit der vorgeschlagenen CONIAN Architektur bei Operationen mit regulärer Adressierung als vergleichbar mit sehr leistungsfähigen bekannten Architekturen, während sie bei Operationen, die auf Ausbreitungsprozessen beruhen, diese deutlich übertrifft. Wenn besonders hohe Geschwindigkeit bei Intra- und Inter-Adressierung erforderlich ist, bietet außerdem eine Parallelisierung des Verarbeitungsteils weiteres Potential.

Architektur	Architekturkonzept	Implementierungsdaten	skalierte Taktfreq.	skalierte Fläche
vorgeschl. Architektur CONIAN	konfigurierbare Verarbeitungseinheit + dedizierte Adreßeinheit	0,25 µm, CMOS, 6 Metallebenen, 2,5V, 200 MHz, 85,2 mm^2	200 MHz	47,7 mm^2
HiPar-4 [RöKn96]	4-fach SIMD-Prozessorsystem	0,6 µm, CMOS, 2 Metallebenen, 80 MHz, 250 mm^2	269 MHz	43,4 mm^2
HiPar-16 [RöKn96]	16-fach SIMD-Prozessorsystem	0,35 µm, CMOS, 3 Metallebenen, 100 MHz, 200 mm^2	196 MHz	102,0 mm^2
PIMM-1 [PIMM-1], [Pey92]	umschaltbare Verarbeitungseinheit + Zeilenspeicher	1,5 µm, CMOS, 5V, 20 MHz, 100 mm^2	168 MHz	2,8 mm^2
PIMM-10 [Lem96]	zwei parallele Verarbeitungseinheiten + Zeilenspeicher	0,7 µm, CMOS, 40 MHz, 180 mm^2	157 MHz	23,0 mm^2
Splash-2 [RaJa97]	FPGAs mit Pipeline-Verbindungen und Crossbar	FPGA	-	-
Sonic-1 [HaSt00]	Module mit FPGA+RAM mit Pipeline- und Busverbindung	FPGA	-	-
MoM-PDA [HaHe98]	Datenfluß Architektur, 32bit PE mit Routing und Verarbeitungseinheit	FPGA, 25 MHz	-	-
[NoMe95]	problemangepaßte Architektur für irreguläre Segmentoperationen	10 MHz, sonst k.a.	-	-
[Nog97]	massiv parallele Architektur mit dedizierten 8bit PE (Wasserscheide)	0,7 µm, CMOS, 100 MHz, 70 mm^2, für 32 x 32 PE	392 MHz	221,0 mm^2
HDPP [GeHe96]	massiv parallele Architektur mit programmierbaren 1bit PE	10 MHz, sonst k.a.	-	-

Tabelle 7.8: Übersicht über die Implementierungsdaten

Architektur	Operation	Bildgröße	Komp.	Verarb.-dauer	skalierte Dauer
vorgeschl. Architektur CONIAN	3x3 FIR Filter	QCIF	3	0,51 ms	
	Kantendetektion	QCIF	3	0,63 ms	
	Median-Filter	QCIF	3	0,38 ms	
	Histogramm	QCIF	1	0,46 ms	
	Skelettierung	QCIF	3	0,29 ms	
	Erosion / Dilatation	QCIF	3	0,38 ms	
	Wasserscheide	QCIF	3	2,00 ms	
	zush. Bereiche bestimmen	QCIF	3	0,58 ms	
HiPar-4	3x3 FIR Filter	512x512	1	21,40 ms	1,85 ms
	Kantendetektion	512x512	1	11,50 ms	0,99 ms
	Histogramm	512x512	1	2,75 ms	0,08 ms
	Skelettierung	512x512	1	21,20 ms	1,83 ms
	zush. Bereiche bestimmen	512x512	1	25,70 ms	2,22 ms
HiPar-16	3x3 FIR Filter	512x512	1	4,30 ms	0,64 ms
	Kantendetektion	512x512	1	2,30 ms	0,34 ms
	Histogramm	512x512	1	0,55 ms	0,03 ms
	Skelettierung	512x512	1	4,30 ms	0,64 ms
	zush. Bereiche bestimmen	512x512	1	10,50 ms	1,55 ms
PIMM-1	Erosion / Dilatation	512x512	1	13,10 ms	0,45 ms
	Wasserscheide Verfahren	512x512	1	13000 s	448,87 ms
PIMM-10	Erosion / Dilatation	512x512	1	3,28 ms	0,24 ms
	Wasserscheide Verfahren	256x256	1	130,00 ms	38,47 ms
	zush. Bereiche bestimmen	256x256	1	50,00 ms	14,80 ms
Splash-2	Kantendetektion	512x512	1	13,89 ms	4,03 ms
Sonic-1	3x3 FIR Filter (2xPIPE)	720x576	1	47,17 ms	8,65 ms
MoM-PDA	3x3 FIR Filter (12x5 rPDUs)	512x512	1	49,80 ms	14,44 ms
[NoMe95]	Wasserscheide Verfahren	512x512	1	452,00 ms	131,10 ms
[Nog97]	Wasserscheide Verfahren	128x128	1	0,011 ms	0,013 ms
HDPP	Median-Filter	512x512	1	0,50 ms	0,14 ms

Tabelle 7.9: Verarbeitungsleistung verschiedener Architekturen

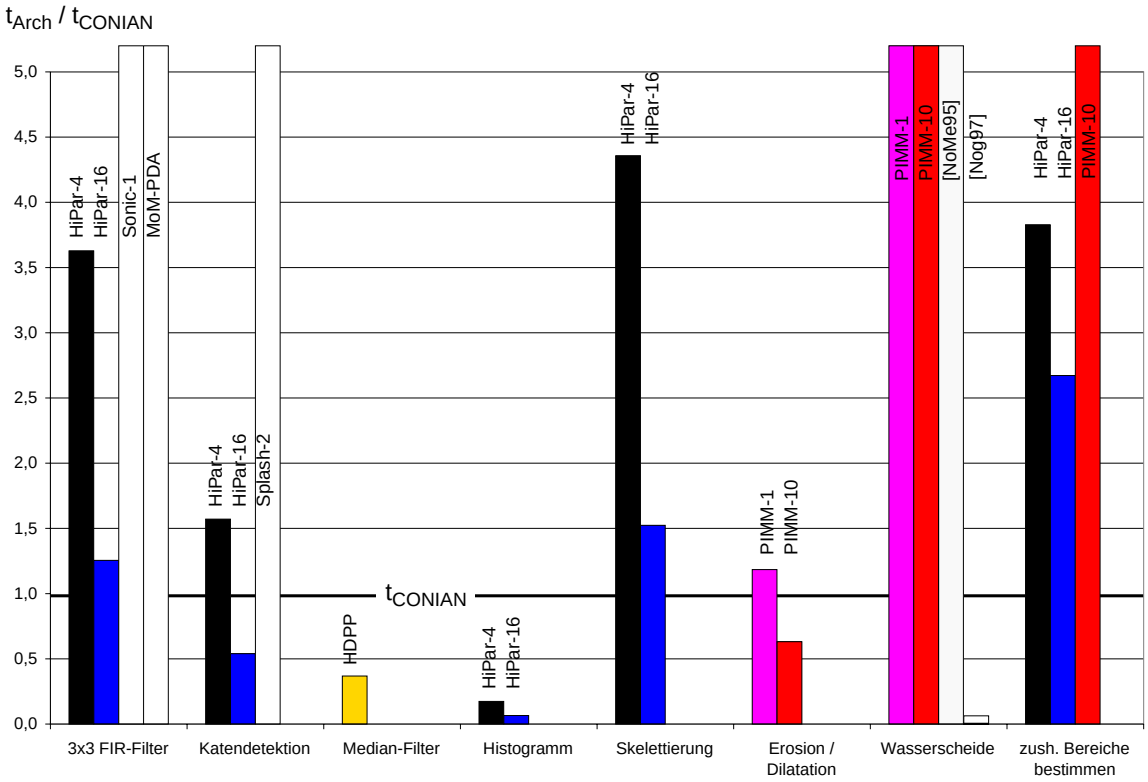


Bild 7.5: Verarbeitungszeit verschiedener Architekturen bezogen auf die Verarbeitungszeit des CONIAN (Werte größer als 5 abgeschnitten)

7.3 Bewertung der Simulationsmethodik

7.3.1 Simulationsgeschwindigkeit

Da für den Architekturvergleich eine schnelle Simulation wichtig ist, wurde die Simulationsdauer für die in Kapitel 6.1.2.4 verglichenen Architekturvarianten gemessen. Die erforderliche Rechenzeit bei QCIF Bildformat schwankt zwischen 267,6 s und 402,3 s und beträgt im Mittel 352,9 s. Gegenüber der Simulationsdauer des Algorithmus, die 46 s beträgt, liegt die Simulationsdauer der Architektur (ohne Auswertung) im Mittel um den Faktor 7,7 höher. Das Verhältnis zwischen simulierter Zeit und Dauer für die Architektursimulation liegt zwischen 1:197 und 1:1029, und beträgt im Mittel 1:596.

Aufgrund seiner Designgröße von ca. 180 kGatter plus 279,1 kByte chip-internem Speicher und des breiten Spektrums durchgeführter Untersuchungen kann der Entwurf der CONIAN-Architektur als aussagekräftiges Fallbeispiel für ein mittelgroßes Design betrachtet werden. Wenn man für eine grobe Angabe der Designgröße 1kByte gleich 1kGatter setzt, bedeutet dies eine Simulationsgeschwindigkeit von $177 \cdot 10^6$ kGatter · Clk pro Sekunde auf einem Celeron 466 gegenüber $2{,}5 \cdot 10^3$ kGatter · Clk pro Sekunde bei einer ereignisgesteuerten VHDL-Simulation auf einer Ultra-Sparc-60. Die vorgeschlagene Methodik gestattet somit eine um den Faktor 10^4 bis 10^5 schnellere Simulation.

Die Ergebnisse sind in Tabelle 7.10 zusammengefaßt. Für Vorgänge, die mit dem Algorithmenmodell in kurzer Zeit simuliert werden können, liegt auch der Aufwand für die Architektursimulation innerhalb praktikabler Grenzen. Aufgrund der hohen Simulationsgeschwindigkeit lassen sich mit der vorgeschlagenen Methodik vergleichende Simulationen auf Systemebene und Untersuchungen von Hardware/Software-Systemen durchführen. Die hohe Simulationsgeschwindig-

keit ist auch von Nutzen, um selten auftretendes kritisches Verhalten zu erkennen bzw. um statistisches Verhalten zu simulieren, wie z.B. die Geschwindigkeit datenabhängiger Operationen, Ressourcen- bzw. Zugriffskonflikte und schwankende Speichernutzung.

Untersuchte Architektur	CONIAN	Verarbeitungseinheit des CONIAN
Größe der Architektur	180 kGatter + 279,1 kByte RAM	68 kGatter
Simulations-methodik	vorgeschlagene Simulationsmethodik	Ereignisgesteuerte VHDL-Simulation
Simulations-plattform	Celeron 466MHz, 128 kByte Cache	Ultra-SPARC-60 Workstation
Simulierte Zeit (Architektur)	$136 \cdot 10^6$ Clk (= 0,68 s)	$101 \cdot 10^3$ Clk (= 5,1 ms)
Simulationsdauer (Architektur)	352,9 s	2700 s
Simulationsdauer (Algorithmus)	46 s	---

Tabelle 7.10: Simulierte Zeit und Simulationsdauer für Algorithmus und Architektur

7.3.2 Einsatzbereich

Der Einsatzbereich der vorgeschlagenen Simulationsmethodik ist auf die Simulation digitaler Architekturen bzw. HW/SW-Systeme begrenzt, deren zugrundeliegender Algorithmus in einer Programmiersprache beschrieben ist. Voraussetzungen für den Einsatz ist, daß für die Teilfunktionen, die Architekturelementen zugeordnet werden, Schätz- oder Erfahrungswerte für die Operationsdauer zur Verfügung stehen. Da die Simulationsgranularität frei wählbar ist, ist eine Simulation auch möglich, wenn Performance-Angaben nur für bestimmte Granularitätsebenen verfügbar sind. Architekturmodelle müssen den Algorithmus auf Ebene der gewählten Simulationsgranularität exakt implementieren, d.h. Architekturen, die äquivalente aber unterschiedliche Algorithmen implementieren, können simuliert werden, wenn sich die Unterschiede zum Algorithmenmodell unterhalb der Simulationsgranularität bewegen. Diese Einschränkung kann jedoch insoweit abgemildert werden, daß Architekturen, die Abbruchkriterien einsetzen, trotzdem simuliert werden können, falls abgebrochene Berechnungen keinen Einfluß auf das Ergebnis haben.

Da im Rahmen dieser Arbeit keine Möglichkeit vorgesehen wurde, den Ablauf der Operationen und Datenzugriffe zur Steuerung des Architekturmodells zu ändern, ergibt sich die weitere Einschränkung, daß der Ablauf (Schedules) in den zu untersuchenden Architekturen der Abfolge im Algorithmus entsprechen muß (auf Ebene der Simulationsgranularität). Im Rahmen von Erweiterungen ist es jedoch möglich, den Ablauf im Algorithmus für die Zuordnung von Operationen zu verschiedenen Architekturelementen zu nutzen und ihn in eine andere Abfolge zur Steuerung des Architekturmodells umzusetzen.

Bei der Untersuchung des CONIAN-Architekturkonzeptes wurden sowohl Architekturvarianten verglichen, als auch Architekturparameter optimiert. Dadurch wurde gezeigt, daß sich ein weites Spektrum an Architekturuntersuchungen durchführen läßt.

Die vorgeschlagene Methodik bietet eine Unterstützung beim Entwurfsprozeß, um Architekturentscheidungen mit möglichst geringem Aufwand zu bewerten. Sie bietet jedoch keine automatische Optimierung bzw. Synthese. Zur Architekturmodellierung ist weder die Erstellung detaillierter Hardware-Modelle erforderlich, noch muß die Funktion bzw. der Algorithmus erneut modelliert zu werden. Es sind auch keine aufwendig zu modellierenden und schwer umzustrukturierende Schnittstellen zwischen Hardware-Blöcken erforderlich. Außerdem ist keine Verifikation mehrerer funktionaler Modelle gegeneinander erforderlich. Der Aufwand zur Erstellung von Architekturmodellen ist daher gering.

Ein Vorteil der vorgeschlagenen Simulationsmethodik ist, daß Architekturen nicht vollständig modelliert werden müssen, um untersucht werden zu können. So können Architekturen simuliert werden, die in ein HW/SW-System eingebettet sind, indem lediglich für die Hardware-Teile Architekturmodelle erstellt werden. Darüberhinaus kann die Architekturmodellierung auf kritische Teile beschränkt werden. Da eine sehr grobe Modellierung möglich ist, können einfache Architekturmodelle erstellt werden und es kann von Annahmen, Schätz- bzw. Erfahrungswerten für die Operationsdauer ausgegangen werden. Durch die Wahlmöglichkeit der Simulationsgranularität kann anschließend eine schrittweise Verfeinerung durchgeführt werden, um weitere als relevant identifizierte Vorgänge in den Architekturen zu simulieren. Umgekehrt kann durch eine Vergröberung die Simulationsgeschwindigkeit erhöht werden.

Durch die gekoppelte Simulation von Architektur und Algorithmus läßt sich darüberhinaus auch die Auswirkung einer Algorithmenoptimierung simulieren. Aus der Sicht eines Architekturentwicklers steuert der Algorithmus die Vorgänge in der konzipierten Architektur bzw. dem Architekturteil, so daß dessen Performance für den simulierten Algorithmus automatisch ausgewertet werden kann. Aus der Sicht eines Algorithmenentwicklers ermöglicht ein vorhandenes Architekturmodell eine Art Profiling für diese Architektur durchzuführen, solange die Änderung der Algorithmenstruktur nicht die Koppelungspunkte im Algorithmus betrifft. Es ergibt sich somit eine größere Unabhängigkeit zwischen Algorithmen- und Architekturentwicklung.

8. Ausblick

In der vorliegenden Arbeit wurde eine neue Architektur zur Videoobjekt-Segmentierung vorgeschlagen. Die als CONIAN (*CONfigurable Image ANalysis* Koprozessor) bezeichnete Architektur besteht aus einem effizienten Speichersystem und einer konfigurierbaren Verarbeitungseinheit. Die Architektur unterstützt die Verarbeitung mit Inter-, Intra-, Segment- und segmentindizierter Adressierung und deckt ein weites Spektrum an Bild- und Segmentoperationen ab. Bei einer Auslegung auf Bildformate bis maximal CIF und Implementierung mit einer 0,25 µm Standardzellen-Technologiebibliothek hat die CONIAN-Architektur einen Flächenbedarf von 47,7 mm^2. Bei der maximalen Taktfrequenz von 200 MHz wurde für einen komplexen iterativen Farbesegmentierungsalgorithmus mit Bildformat QCIF eine Verarbeitungsgeschwindigkeit von 29,3 Bilder pro Sekunde erreicht, wenn jedes Bild segmentiert wird, bzw. 65,9 Bilder pro Sekunde, wenn die Segmentierung für jedes vierte Bild erfolgt und dazwischen eine Bewegungskompensation vorgenommen wird. Gegenüber einem Celeron Prozessor mit 466 MHz erreicht die CONIAN-Architektur eine um den Faktor 24 höhere Geschwindigkeit, gegenüber einem Pentium IV mit 2 GHz bei Nutzung von MMX wurde für die CONIAN-Architektur eine Beschleunigung um den Faktor 4,6 geschätzt. Da bei niedriger Taktfrequenz eine hohe Verarbeitungsgeschwindigkeit erreicht wird, ergibt sich durch Absenkung der Versorgungsspannung die Möglichkeit zur Reduzierung des Leistungsverbrauches.

Damit wurde gezeigt, daß sich die vorgeschlagene Architektur einsetzen läßt, um Videoobjekt-Segmentierung in Echtzeit zu implementieren, oder um die Produktion von segmentiertem Material, die nicht notwendigerweise in Echtzeit erfolgen muß, zu beschleunigen. Einsatzgebiete sind sowohl stationäre Geräte wie PCs und Bildtelefone, bei denen besonders die hohe Geschwindigkeit der Architektur von Nutzen ist, als auch mobile Geräte, bei denen sich geringer Leistungsverbrauch positiv auswirkt. Damit ergeben sich vielfältige Anwendungsmöglichkeiten für die vorgeschlagene Architektur.

Eine mit der CONIAN-Architektur implementierte Segmentierung läßt sich z.B. in Kommunikationsanwendungen einsetzen, um den Hintergrund, der insbesondere bei Aussenaufnahmen komplexe Strukturen und viele bewegte Objekte enthalten kann, zu vereinfachen bzw. entfernen. Damit kann die - insbesondere bei mobilen Anwendungen begrenzte - Übertragungsrate für wichtige Objekte wie Personen und Gesichter eingesetzt werden. Da Objekte aus verschiedenen Quellen kombiniert werden können, sind neuartige Anwendungen denkbar, bei denen beispielsweise mehrere Teilnehmer an unterschiedlichen Standorten in ein Bild eingeblendet werden (virtueller Konferenzraum), so daß ein Konferenzteilnehmer die anderen Teilnehmer mit einem Blick erfassen kann. Dies ist insbesondere bei mobilen Geräten mit kleinen Displays vorteilhaft. Die Produktions von segmentiertem Material bietet die Möglichkeit, für Videoobjekte Links zu definieren, so daß sich beispielsweise Kleidungsstücke bei einer Modenschau oder Sehenswürdigkeiten bei einer Stadttour anklicken lassen. Auf diese Weise kann der Betrachter auf intuitive Weise gezielt Informationen auswählen, die ihn interessieren. Im Fall der Modenschau kann z.B. der Ablauf der Präsentation verändert werden, oder es können interaktiv zusätzliche Informationen über Preis, Lieferbarkeit, etc. angefordert und angezeigt werden.

Neben diesen Anwendungen läßt sich die CONIAN-Architektur aufgrund Ihrer Flexibilität auch in Überwachungsanwendungen einsetzen oder auch zur Segmentierung von Videomaterial, um Merkmale für eine automatische Suche zu extrahieren. Auch andere Anwendungen im Bereich Bildanalyse wie z.B. industrielle und medizinische Anwendungen kommen in Betracht.

Aufgrund ihrer Verarbeitungsleistung läßt sich die CONIAN-Architektur für Anwendungen mit einem Bildformat bis zu CIF einsetzen. Für Anwendungen die größere Bildformate und Echtzeit-Verarbeitung benötigen, ist eine Verbesserung der Skalierbarkeit das Ziel weiterführender Arbeiten. In diesem Zusammenhang besteht Potential durch eine Parallelisierung auf Bildpunktebene.

Außerdem sind neue Erkenntnisse von der Untersuchung der Zusammenschaltung mehrerer Instanzen der CONIAN-Architektur und Ermittlung der Möglichkeiten einer hybriden Verarbeitungseinheit zu erwarten, die sowohl programmierbare als auch konfigurierbare Flexibilität beinhaltet. Im Rahmen weiterführender Arbeiten ist außerdem interessant zu untersuchen, wie sich 3D-Segmentierungsverfahren implementieren lassen. Soweit sich die Operationen eines solchen Algorithmus von der CONIAN-Architektur unterstützen lassen, kann mit der entwickelten Simulationsmethodik untersucht werden, welche Verarbeitungsgeschwindigkeit mit einem HW/SW-System bestehend aus der CONIAN-Architektur und einem Universalprozessor erreicht werden kann.

Obwohl die im Rahmen dieser Arbeit entwickelte Simulationsmethodik vor dem Hintergrund einer speziellen Aufgabenstellung entworfen wurde, nämlich dem Vergleich von Architekturalternativen für eine flexible Architektur zur Videoobjekt-Segmentierung, liegen ihr allgemeine Prinzipien zugrunde, die sich für einen weitergehenden Einsatz eignen. Der neue Ansatzpunkt der Methodik ist dabei die Kopplung von Architekturmodellen an Software-Algorithmenmodelle. Da sie komplexe manuelle Entwurfsschritte durch einfachere und genauere Beurteilung der konzeptionierten Architekturvarianten unterstützt, eignet sich die Simulationsmethodik für den typischerweise gegebenen Fall, daß zu große Komplexität eine automatische Synthese verhindert. In der Arbeit wurde gezeigt, daß mit der vorgeschlagenen Methodik Modelle mit geringem Modellierungsaufwand erstellt werden können, die hohe Simulationsgeschwindigkeit gestatten. Gegenüber Hardware-Beschreibungssprachen wie VHDL ist weiterhin von Vorteil, daß es möglich ist, partielle Modelle zu erstellen, mit denen nur die Teile von Architekturen bzw. Systemen beschrieben werden, die als entscheidend und kritisch angesehen werden.

Die vorgeschlagene Simulationsmethodik eignet sich für den Vergleich von Hardware-Architekturen bzw. HW/SW-Systemen, deren zugrundeliegender Algorithmus durch ein Software-Modell beschrieben ist. Sie bietet hohes Potential zur Unterstützung der Implementierung datendominierter Algorithmen, wie z.B. im Bereich der Bild- bzw. Videoverarbeitung, der Kommunikationstechnik und anderen Bereichen der Signalverarbeitung, wobei insbesondere bei datenabhängigem Ablauf ein hoher Nutzen zu erwarten ist. Um die Grenzen des Einsatzbereiches abzustecken und ggf. zu erweiten, wäre eine eingehende Untersuchung der vorgeschlagenen Methodik anhand weiterer Anwendungsbeispiele lohnend.

Im Hinblick auf eine Erweiterung des Einsatzbereiches ist es besonders von Interesse zu untersuchen, wie der zeitliche Ablauf der Operationen in der Architektur (Scheduling) unabhängig von der Operationsreihenfolge im Algorithmus gewählt werden kann. In diesem Zusammenhang sind auch neue Erkenntnisse durch den Ansatz zu erwarten, die Sprache SystemC für die Modellierung der Architekturelemente einzusetzen. Im Rahmen weiterführender Arbeiten kann außerdem die Einbindung der Simulationsmethodik in den weiteren Entwurfsablauf beleuchtet werden. Dabei zu untersuchende Fragestellungen sind Verifikation bzw. Co-Simulation synthetisierbarer Hardware-Modelle mit einem Architekturmodell der vorgeschlagenen Methodik bzw. Möglichkeiten zur Unterstützung des Entwurfes synthetisierbarer Hardware-Modelle.

Mit den genannten Erweiterungen kann die entwickelte Simulationsmethodik einen guten Beitrag zur Schließung der Lücke zwischen der Entwicklung der Halbleitertechnologie und der Entwurfsproduktivität liefern.

Anhang

A. Organisation der segmentindizierten Daten

Bezeichnung	Bedeutung	Bitbreite	Zugriff
prevsegno	Segmentnummer im vorherigen Bild	16 bit	-
mergeto	Nummer des Segmentes, mit dem dieses Segment verbunden wurde	16 bit	R/W
newlev	Anzahl der Teilsegmente	2 bit	-
map	Status, der angibt ob das Segment auf Fragmentierung geprüft wurde	2 bit	R/W
nextunused	nächste unbenutzte Segmentnummer	16 bit	R
px	x-Koordinate eines Punktes im Segment	12 bit	R/W
py	y-Koordinate eines Punktes im Segment	12 bit	R/W
size	Anzahl der Bildpunkte im Segment	20 bit	R/W
sizelimit	Obere Grenze für die Anzahl der Bildpunkte im Segment	20 bit	R
xmin	kleinste x-Koordinate	12 bit	R/W
xmax	größte x-Koordinate	12 bit	R/W
ymin	kleinste y-Koordinate	12 bit	R/W
ymax	größte y-Koordinate	12 bit	R/W
xsum	x-Koordinatensumme für die Schwerpunktberechnung	32 bit	R/W
ysum	y-Koordinatensumme für die Schwerpunktberechnung	32 bit	R/W
xcenter	x-Koordinate des Schwerpunktes	12 bit	R
ycenter	y-Koordinate des Schwerpunktes	12 bit	R
ys	Summe der Luminanzwerte	28 bit	R/W
us	Summe der U-Chrominanzwerte	28 bit	R/W
vs	Summe der V-Chrominanzwerte	28 bit	R/W
ym	Mittelwert der Luminanz	8 bit	R
um	Mittelwert der U-Chrominanz	8 bit	R
vm	Mittelwert der V-Chrominanz	8 bit	R
maxgeodist	Geodätische Distanz des charakteristischen Punktes zum Segmentrand	20 bit	-
centredist	Distanz des charakteristischen Punktes zum Schwerpunkt	20 bit	R/W
colordist	Differenz des charakteristischen Punktes zu den Farbmittelwerten	9 bit	R/W
Gesamt		**409 bit**	

Tabelle A.1: Komponenten eines segmentindizierten Datensatzes

1. Bündel	2. Bündel	3. Bündel	4. Bündel	5. Bündel
px, py, xcenter, ycenter, ym, um, vm, maxgeodist, centredist, colordist	size, ys, us, vs	size, xsum, ysum, xcenter, ycenter	px, py, size, xmin, xmax, ymin, ymax	prevsegno, mergeto, newlev, map, nextunused, sizelimit
121 bit	104 bit	108 bit	92 bit	62 bit

Tabelle A.2: Aufteilungen der segmentindizierten Daten in Bündel

B. Ablauf der Bildpunktoperationen

Bild B.1 zeigt den Ablauf einer Bildpunktoperation bei Segment-Adressierung für den Fall der Referenzarchitektur. Der Ablauf wird durch den Pixel-Level-Controller so gesteuert, daß sich die Teiloperationen überlappen. Bei der gezeigten Bildpunktoperation erfolgen auch segmentindizierte Schreib- und Lesezugriffe, so daß sich insgesamt eine Zyklusdauer von 12 Takten ergibt. Die graue bzw. hellgraue Markierung bezeichnet jeweils die zu einer Bildpunktoperation gehörenden Teiloperationen.

In der CONIAN-Architektur wird ein Cachespeichers mit acht Bänken eingesetzt, der einen gleichzeitigen Zugriff auf eine CON4-Nachbarschaft ermöglicht. Während bei der Referenzarchitektur die Speicherschnittstelle in jedem Fall den Engpaß darstellt, ist dies bei der CONIAN-Architektur von der Verarbeitungsdauer abhängig. Ist sie groß, bestimmt sie die Anzahl an Taktzyklen, die für eine Bildpunktoperation erforderlich sind. Ist sie klein, bestimmt die Anzahl an Speicherzugriffen die Dauer der Bildpunktoperation. Die nachfolgend gezeigten Abläufe beziehen sich auf den Fall, daß die Verarbeitungsdauer und die Dauer für Speicherzugriffe balanciert sind.

Bei Intra-Adressierung ist - abgesehen von der Initialisierung - ein Zugriff zum Lesen und ein Zugriff zum Schreiben ausreichend, da die Möglichkeit zum Schieben der Bilddaten in der Registermatrix besteht. Es ergibt sich der in Bild B.2 gezeigte Ablauf mit einer Mindestdauer von 2 Taktzyklen. Bei Inter-Adressierung sind Daten aus zwei verschiedenen Eingangsbildern zu laden. Damit sind drei Speicherzugriffe pro Bildpunktoperation erforderlich und es ergibt sich der in Bild B.3 gezeigte Ablauf. Bei Segment-Adressierung mit einer CON4-Nachbarschaft kann Laden und Schreiben der Bilddaten in je einem Takt erfolgen, so daß der Ablauf dem der Intra-Adressierung entspricht. Für Segment-Adressierung mit einer CON8 Nachbarschaft sind zum Laden bzw. Speichern je zwei Zugriffe erforderlich. Es ergibt sich der in Bild B.4 gezeigte Ablauf mit einer Mindestdauer von 4 Takten.

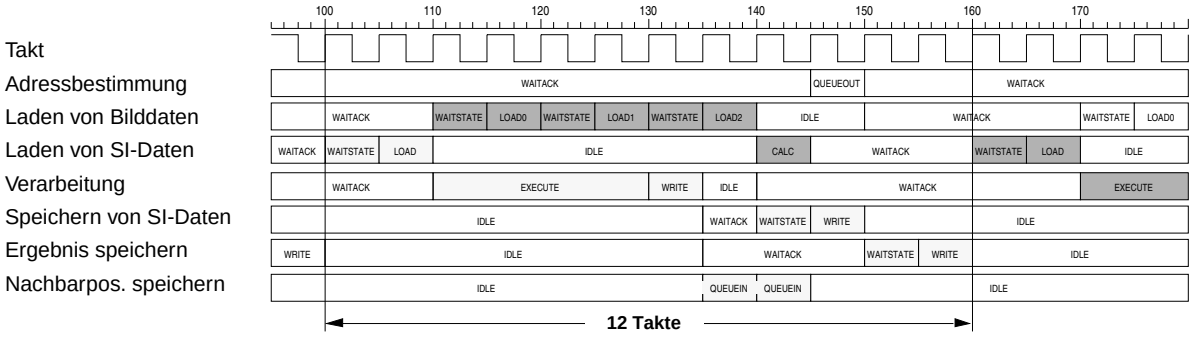


Bild B.1: Ablauf einer Bildpunktoperation bei direktem Zugriff auf den Bildspeicher

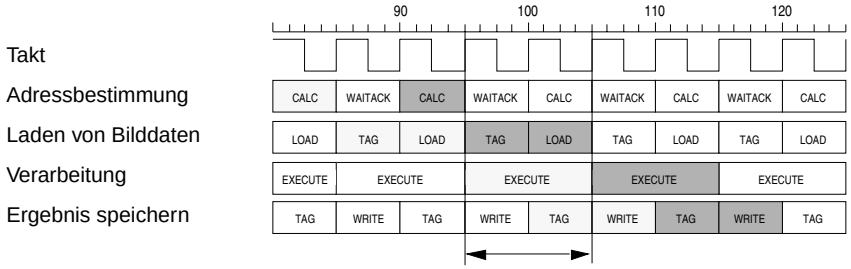


Bild B.2: Ablauf einer Bildpunktoperation bei Intra-Adressierung

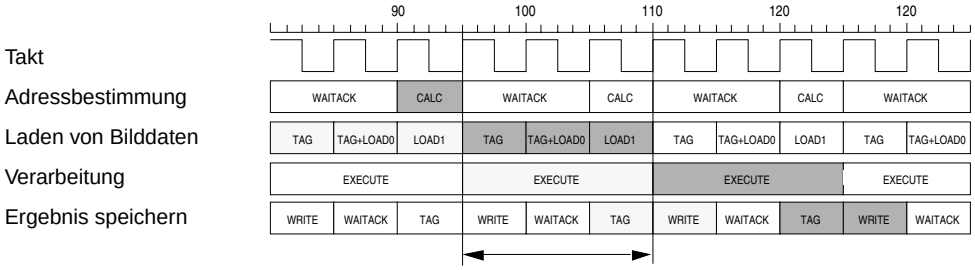


Bild B.3: Ablauf einer Bildpunktoperation bei Inter-Adressierung

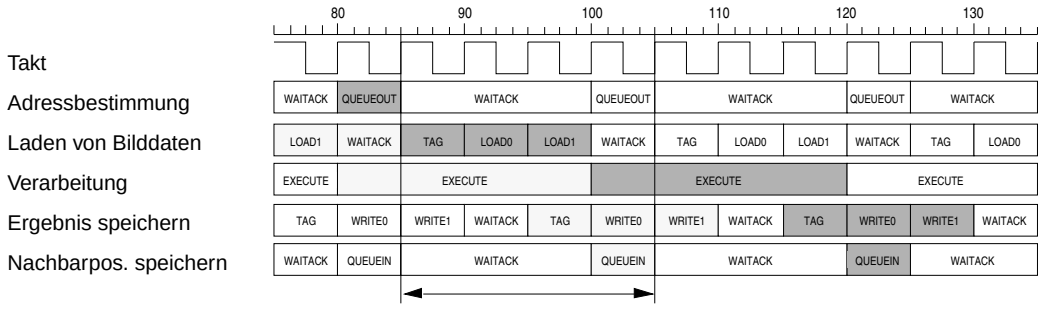


Bild B.4: Ablauf einer Bildpunktoperation bei Segment-Adressierung

C. Verarbeitungsdauer verschiedener Bildpunktoperationen

Bildpunktoperation	Nachbar-schaftstyp	Bildpunkte laden	Bildpunkte speichern	SI-Daten laden	SI-Daten speichern	Nachbarpos. speichern	Verarbei-tungsdauer	Gesamt-dauer
Bilddifferenz	CON0	2	✔				1	6
SAD Berechnung	CON0	2					2	6
Histogramm	CON0	1		✔	✔		2	6
Segm. Erosion	CON4	3	✔				2	8
Inhom. Pkt. löschen	CON0	1	✔				2	4
Farbdifferenz	CON0	1	✔				2	4
Farbsumme	CON0	1	✔				2	4
Segmentposition	CON0	1		✔	✔		2	6
Segmentfarbe	CON0	1		✔	✔		3	6
Segmentverfolgung	CON0	1		✔	✔		2	6
Hist. Filterung	CONH2	1	✔				3	4
Kopieren	CON0	1	✔				1	4
Wertebereich	CON0	1					2	4
Relaxation	CON8	3	✔	✔			7	9
Initialisierung	CON0		✔				1	4
Sobeloperator	CON8	3	✔				5	8
global Pkt. löschen	CON0	1	✔				2	4
Segm. Dilatation (SC)	CON4	5	✔			✔	1	8
Segm. löschen (SC)	CON8	9	✔			✔	1	8
Fragmentierungstest (SC)	CON8	9	✔	✔		✔	1	9
Segm. markieren (SC)	CON8	9	✔	✔		✔	1	9
Segmentgröße (SC)	CON8	9				✔	1	8
Wasserscheide (SC)	CON4	5	✔			✔	1	8
Segm. Dilatation (NC)	CON4	5	✔			✔	1	14
Segm. löschen (NC)	CON8	9	✔			✔	1	22
Fragmentierungstest (NC)	CON8	9	✔			✔	2	22
Segm. markieren (NC)	CON8	9	✔			✔	4	22
Segmentgröße (NC)	CON8	9				✔	4	22
Wasserscheide (NC)	CON4	5	✔			✔	4	14

Tabelle C.1: Verarbeitungsdauer verschiedener Bildpunktoperationen

D. Zugriffshäufigkeit bei unterschiedlicher Speicherschnittstelle

parallel zu ladende Kanalbündel			1	2	6	1	2	6	1	2	6	1	2	6
parallel zu ladende Bildpunkte			1	1	1	3	3	3	5	5	5	9	9	9
Kanäle	Bildp.	Häufigkeit												
Y	1	2,3%	1	1	1	1	1	1	1	1	1	1	1	1
AX	1	4,6%	1	1	1	1	1	1	1	1	1	1	1	1
U,V	1	6,4%	2	1	1	2	1	1	2	1	1	2	1	1
A,M	5	2,5%	10	5	5	6	3	3	2	1	1	2	1	1
A,M	9	21,5%	18	9	9	6	3	3	4	2	2	2	1	1
Y,A	1	10,5%	2	2	1	2	2	1	2	2	1	2	2	1
Y,U,V	3	2,3%	9	3	3	3	1	1	3	1	1	3	1	1
Y,U,V,A	1	30,3%	4	2	1	4	2	1	4	2	1	4	2	1
Y,U,V,A	2	4,5%	8	4	2	4	2	1	4	2	1	4	2	1
Y,U,V,A	3	7,3%	12	6	3	4	2	1	4	2	1	4	2	1
Y,U,V,A,M	9	0,6%	45	18	9	15	6	3	10	4	2	5	2	1
Y,U,V,A,AX,M	1	7,1%	6	2	1	6	2	1	6	2	1	6	2	1
Y,U,V,A,AX,M	5	0,1%	30	10	5	12	4	2	12	2	1	6	2	1
gewichtete Summe			**791**	**396**	**311**	**413**	**211**	**149**	**357**	**183**	**122**	**310**	**160**	**100**

Tabelle D.1: Relative Anzahl der Lesezugriffe bei unterschiedlicher Speicherschnittstelle

parallel zu speichernde Kanalbündel			1	2	6	1	2	6	1	2	6	1	2	6
parallel zu speichernde Bildpunkte			1	1	1	3	3	3	5	5	5	9	9	9
Kanäle	Bildp.	Häufigkeit												
-	-	9,6%	0	0	0	0	0	0	0	0	0	0	0	0
Y	1	8,7%	1	1	1	1	1	1	1	1	1	1	1	1
A	1	10,5%	1	1	1	1	1	1	1	1	1	1	1	1
M	1	15,8%	1	1	1	1	1	1	1	1	1	1	1	1
A,M	1	0%	2	1	1	2	1	1	2	1	1	2	1	1
A,AX,M	1	8,8%	3	1	1	3	1	1	3	1	1	3	1	1
Y,U,V,A	1	39,5%	4	2	1	4	2	1	4	2	1	4	2	1
Y,U,V,A,AX,M	1	7,1%	6	2	1	6	2	1	6	2	1	6	2	1
gewichtete Summe			**278**	**137**	**90**	**278**	**137**	**90**	**278**	**137**	**90**	**278**	**137**	**90**

Tabelle D.2: Relative Anzahl der Schreibzugriffe bei unterschiedlicher Speicherschnittstelle

E. Vergleich von Zeilenspeicher und Prefetching

Im Folgenden wird die Effizienz eines Caches mit Prefetching für reguläre Adressierung abgeschätzt. Während beide Architekturvarianten im eingeschwungenen Zustand dieselbe Menge an Daten laden und speichern, und so bei balancierter Busauslastung dieselbe Geschwindigkeit erreichen, bestehen während der Initialisierungs- und der Abschlußphase Unterschiede.

Im Sinne einer pessimistischen Abschätzung der Prefetching-Effizienz wird davon ausgegangen, daß für jedes Eingangsbild die erste Zeile von Cache-Blöcken vor der Verarbeitung vollständig geladen wird und daß die letzte Zeile des Ergebnisbildes erst nach Beendigung der Verarbeitung zurückgeschrieben wird. Für das Bildformat QCIF wird angenommen, daß unabhängig von der Blockgröße beim Laden einer Cache-Blockzeile 6 Seitenwechsel auftreten (da der in einer Speicherseite abgelegte Bildausschnitt nicht von der verwendeten Blockgröße abhängt). Mit der Anzahl an Bildpunkten pro Cacheblock n_{pixel}, der Anzahl an Blocks pro Zeile n_{blocks} ergibt sich für Initialisierungs- und Abschlußphase eine Dauer von

$$t_{OH,Intra} = 2 \cdot ((1 + n_{page}) \cdot (t_{page} + t_{pipe}) + n_{blocks} \cdot (n_{pixel}/n_{parallel}) \cdot t_{acc}) \quad (8.1)$$

$$t_{OH,Inter} = 3 \cdot ((1 + n_{page}) \cdot (t_{page} + t_{pipe}) + n_{blocks} \cdot (n_{pixel}/n_{parallel}) \cdot t_{acc}) \quad (8.2)$$

Die Operation „Kopieren“ wird häufig auf einen Bereich von 4x4 Punkten angewandt. Da dieser Bereich nicht notwendigerweise mit der Blockeinteilung übereinstimmt, wird in diesen Fällen für das Laden der ersten Zeile bei den Blockgrößen 2x2 und 4x4 ein Verschnitt von einem Block zugrundegelegt. Damit ergibt sich

$$t_{OH,K4x4} = 2 \cdot ((t_{page} + t_{pipe}) + n_{blocks,K4z4} \cdot (n_{pixel}/n_{parallel}) \cdot t_{acc}) \quad (8.3)$$

Bei der Operation „inhomogene Punkte löschen“ stimmt der Verarbeitungsbereich meist nicht mit den Blockgrenzen überein. Es wird in beiden Koordinatenrichtungen ein Verschnitt von einem Block angenommen. Innerhalb der typischen Seitenlänge von 80 Bildpunkten treten 3 Seitenwechsel auf. Für die Dauer von Initialisierungs- und Abschlußphase ergibt sich

$$t_{OH,IPL} = 2 \cdot ((1 + n_{page}) \cdot (t_{page} + t_{pipe}) + n_{blocks,IPL} \cdot (n_{pixel}/n_{parallel}) \cdot t_{acc}) \quad (8.4)$$

In Tabelle E.1 und E.2 sind die Parameter und die Ergebnisse obiger Abschätzung angegeben.

Block-größe	Intra		Inter		K4x4		IPL	
	n_{blocks}	t_{OH}	n_{blocks}	t_{OH}	n_{blocks}	t_{OH}	n_{blocks}	t_{OH}
2x2	88	816	88	1224	2+1	40	40+1	392
4x4	44	1520	44	2280	1+1	80	20+1	736
8x8	22	2928	22	4392	1	144	10+1	1472
16x16	11	5744	11	8616	1	528	5+1	3136

Tabelle E.1: Initialisierungs- und Abschlußdauer bei Einsatz eines Cache mit Prefetching

Operation	Anzahl	Typ		Zeilen-speicher	Prefetching			
		Z-Sp.	Pref.		2x2	4x4	8x8	16x16
Bilddifferenz	20	Inter	Inter	1280	24480	45600	87840	172320
SAD Berechnung	20	Inter	Inter	1280	24480	45600	87840	172320
Histogramm	156	Intra1D	Intra1D	6240	127296	237120	456768	896064
Segm. Erosion	21	Intra2D	Intra2D	10584	17136	31920	61488	120624
inhom.Pt. löschen	321	Intra1D	IPL	12840	125832	236256	472512	1006656
Farbdifferenz	12	Intra1D	Intra1D	480	9792	18240	35136	68928
Farbsumme	12	Intra1D	Intra1D	480	9792	18240	35136	68928
Segmentpos.	156	Intra1D	Intra1D	6240	127296	237120	456768	896064
Segmentfarbe	21	Intra1D	Intra1D	840	17136	31920	61488	120624
Segmentverfolgung	93	Intra1D	Intra1D	3720	75888	141360	272304	534192
Kopieren	63	Intra1D	Intra1D	2520	51408	95760	184464	361872
Kopieren	31612	Intra1D	K4x4	1264480	1264480	2528960	4552128	16691136
Wertebereich	12	Intra1D	Intra1D	480	9792	18240	35136	68928
Relaxation	84	Intra2D	Intra2D	42336	68544	127680	245952	482496
Initialisierung	68	Intra1D	Intra1D	2720	55488	103360	199104	390592
Sobel-Operator	20	Intra2D	Intra2D	10080	16320	30400	58560	114880
global Pt. löschen	20	Intra1D	Intra1D	800	16320	30400	58560	114880
Summe:				**1367400**	**2041480**	**3978176**	**7361184**	**22281504**
Bezogen auf Verarbeitungsdauer:				**2,1%**	**3,2%**	**6,2%**	**11,5%**	**34,9%**

Tabelle E.2: Vergleich der Initialisierungs- und Abschlußdauer von Zeilenspeicher und Cache mit Prefetching

F. Platzierungsschemata

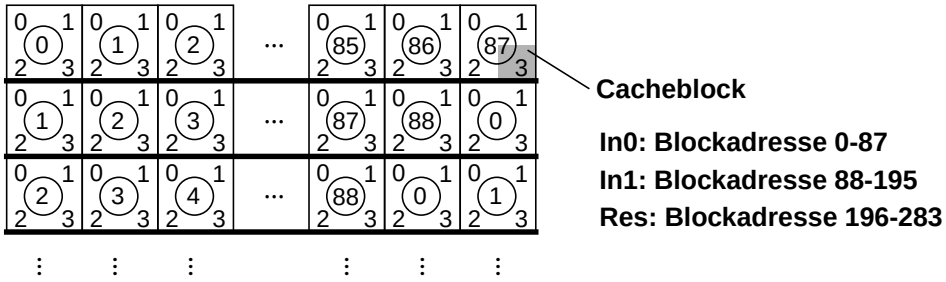


Bild F.1: Plazierungsschema bei Verwaltung von drei Bildern

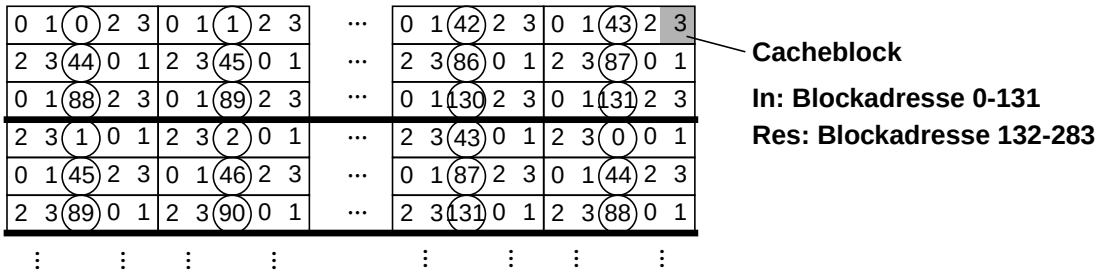


Bild F.2: Plazierungsschema bei Verwaltung von zwei Bildern

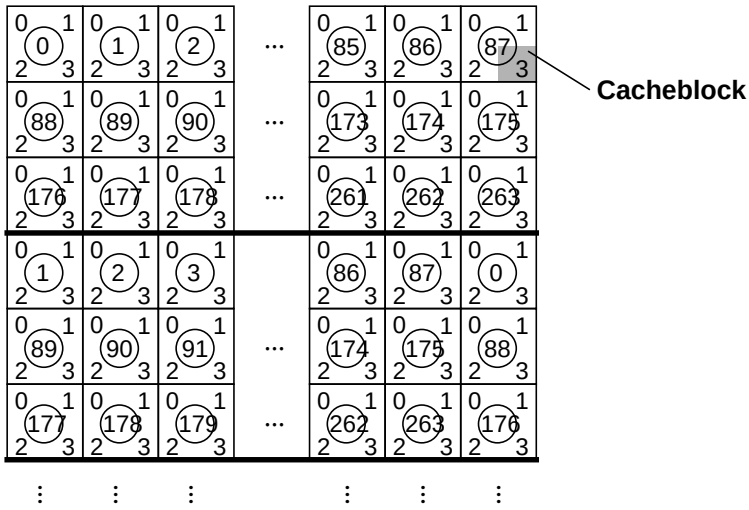


Bild F.3: Plazierungsschema bei Verwaltung von einem Bild

Literaturverzeichnis

[AdRo99] A. Adario, E. Roehe, S. Bampi: „Dynamically Reconfigurable Architecture for Image Processor Applications“, Design Automation Conference, DAC’99, New Orleans, S. 623-628, 1999

[AlOn98] A.A. Alatan, L. Onural, M. Wollborn, R. Mech, E. Tuncel, T. Sikora: „Image Sequence Analysis for Emerging Interactive Multimedia Services - the European COST 211 Framework“, IEEE Transaction on Circuits and Systems for Video Technology, Bd. 8, S. 802-813, November 1998

[Ant96] K. Antreich: „Rechnergestützte Entwurfsverfahren“, Vorlesungsunterlagen, Vorlesung gehalten 1996 an der Technischen Universität München

[Aphelion] Aphelion Development Tool, ADCIS, 10 avenue de Garbsen, 14200 Herouville Saint-Clair, Frankreich, http://www.adcis.net

[BaIr93] P. Balsara, M. Irwin: „Intermediate-Level Vision Tasks on a Memory Array Architecture“, Machine Vision and Applications, Nr. 6, S. 50-64, 1993

[Bar98] E.Barke: „Bei Electronic Design Automation bleibt der Weg das Ziel“, in GMM Report, Herausgegeben von der VDI/VDI-Gesellschaft Mikroelektronik, Mikro- und Feinwerktechnik, S.162-165, 1998

[BePi98] M. Berekovic, P. Pirsch, J.Kneip: „An Algorithm-Hardware-System Approach to VLIW Multimedia Processors“, Journal of VLSI Signal Processing, Bd. 20, S. 163-180, 1998

[BiCa93] A. Biancardi, V. Cantoni, M. Mosconi: „Program development and coding on a fine-grained vision machine“, Machine Vision and Applications, Bd. 7, Nr. 1; S. 23-29, 1993

[BoWa95] V.M. Bove, J.A. Watlington: „Cheops: A Reconfigurable Data-Flow System for Video Processing“, IEEE Transactions on Circuits and Systems for Video Technology, Bd. 5, Nr. 2, S. 140-149, April 1995

[BPMo98] J. Benois-Pineau, F. Morier, D. Barba, H. Sanson: „Hierarchical Segmentation of Video Sequences for Content Manipulation and Adaptive Coding“, Signal Processing, Bd. 66, S. 181-201, 1998

[Brä93] T. Bräunl: „Parallele Programmierung“, Braunschweig/Wiesbaden: Vieweg-Verlag, 1993

[BrFr92] S. D. Brown, R. J. Francis, J. Rose, Z. Vranesic: „Field-Programmable Gate Arrays“, Dordrecht/Boston/London: Kluwer-Verlag, 1992

[CaHa00] T. J. Callahan, J. R. Hauser, J. Wawrzynek: „The Garp Architecture and C Compiler“, Computer, Bd. 33, Nr. 4, S. 62-69, April 2000

[CBC10] „CB-C10 Design Manual“, NEC Electronics Inc., 2880 Scott Boulevard, Santa Clara, CA 95050, USA, http://www.necel.com, 1997

[ChEn96] M. Chiodo, D.Engels, P.Giusto, H.Hsieh, A.Jurecska, L. Lavagno, K. Suzuki, A. Sangiovanni-Vincentelli: „A Case Study in Computer Aided Co-design of Embedded Controllers“, Design Automation for Embedded Systems, Bd. 1, Nr. 1-2, S. 51-67, 1996

[ChMi97] M. Choudhury, J. Miller: „A 300 MHz CMOS Microprocessor with Multi-Media Technology“, IEEE International Solid-State Circuits Conference, ISSCC‘97, S. 170-171/450, 1997

[ChTs93] A. Chihoub, A.Tsai, M. La Valava, J. Avins, J. Turlip: „A Field Programmable Gate Array Implementation of a Systolic Architecture for a Morphology Engine“, Proceedings of the SPIE, Bd. 2064, S. 95-106, 1993

[CoCo93] L.D. Cohen, I. Cohen: „Finite-Element Methods for Active Contour Models and Balloons for 2D and 3D Images“, IEEE Transactions on Pattern Analysis and Machine Intelligence, Bd. 15, Nr. 11, S. 1131-1147, November 1993

[CoPe97] P. Correia, F. Pereira: „Segmentation of Video Sequences in a Video Analysis Framework“, Proceedings of the WIAMIS`97, S. 155-160, Juni 1997

[Cossap] „COSSAP User‘s Manual“, Synopsys Inc. 700 E. Middlefield Rd., Mountain View, CA 94043, USA

[C60] „TMS320C6000 CPU and Instruction Set Reference Guide“, Literature Number: SPRU189F, http://www.ti.com, Oktober 2000

[C80] „TMS320C8x System-Level Synopsis“, Literature Number: SPRU113B, http://www.ti.com, Oktober 2000

[Dav97] E.R. Davies: „Machine Vision“, New York: Academic Press, 1997

[DC] „Design Compiler User‘s Manual“, Synopsys Inc. 700 E. Middlefield Rd., Mountain View, CA 94043, USA

[dHo00] A. De Hon: „The Density Advantage of Configurable Computing“, Computer, S. 41-49, April 2000

[Die98] H. Dietrich: „Bewegungsanayse und Zusammenfassung von Segmenten in Videosequenzen“, Diplomarbeit am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München, Dezember 1998

[dMGu97] G. de Micheli, R. K. Gupta: „Hardware/Software Co-Design“, Proceedings of the IEEE, Bd. 85, Nr. 3, S. 349-365, März 1997

[DPR2] „DPR2 Synchronous Double Port RAM“, Datenblatt, ST Microelectronics, Bretonischer Ring 4, D-85626 Grasbrunn

[DrAn92] P.J. Drongowski, K. Andress: „A Morphology-based Processor for Realtime Enhancement“, Proceedings of the SPIE, Bd. 1823, S. 25-36, 1992

[Dra97] D.Draper, et al.: „An x86 Microprocessor with Multimedia Extensions“, IEEE International Solid-State Circuits Conference, ISSCC‘97, S. 172/450, 1997

[dWi97] D. de Wildt: „Segmentrandverfolgung durch ein aktives Konturenmodell“, Diplomarbeit am Institut für Angewandte Mathematik der Universität Hamburg, April 1997

[eArch] „eArchitect^{TM} Users Manual“, Benutzerhandbuch, Innoveda, 293 Boston Post Road West, Marlborough, MA 01752, USA, http://www.innoveda.com

[Envision] C. Goldberg: „Visual Architect Bridges the Gap Between Systems and ASIC Designers“, Cadence Design Systems Inc., 2655 Seely Avenue, San Jose, CA 95134, USA, http://www.cadence.com

[ErAl97] P. Eren, Y. Altunbask, M. Tekalp: „Region based affine motion segmentation using color information“, Proceedings of the ICASSP‘97, Bd. 4, S. 3005, April 1997

[Ern98] R. Ernst: „Codesign of Embedded Systmes: Status and Trends“, IEEE Design and Test of Computers, S. 45-54, April/Juni 1998

[EyBi98] J. Eyre, J.Bier: „DSP Processors Hit the Mainstream“, Computer, S. 51-59, August 1998

[Fär98] G. Färber: „Mikroprozessoren“, Vorlesungsmanuskript, Vorlesung gehalten im Sommersemester 1998 an der Technischen Universität München

[Fly66] M.J. Flynn: „Very High Speed Computing Systems“, Proceedings of the IEEE, Bd. 54, 1966

[Fre00] F. Fremerey: „Eine Revolution in Silizium - Rekonfigurierbare Logik im Vergleich“, c’t Magazin für Computertechnik, Nr. 17, S. 202-209, September 2000

[Fre99] F. Fremerey: „Hardware im Fluss - Rekonfigurierbare Logik zwischen Forschung und Massenmarkt“, c’t Magazin für Computertechnik, Nr. 21, S. 282-287, Oktober 1999

[Fuc98] T. Fuchs: „Architekturentwurf und Modellierung des Pixellevel-Controllers eines Segmentierungsprozessors“, Diplomarbeit am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München, Dezember 1998

[GaMa97] L. Garrido, F. Marques, M. Pardas, P. Salembier, V. Vilaplana: „A hierarchical technique for image analysis“, Proceedings of the WIAMIS`97, S. 13-20, Juni 1997

[GaVa94] D. D. Gajski, F. V. Vahid, S. Narayan, J. Gong: „Specification and Design of Embedded Systems“, London: Prentice Hall, 1994

[GeGa95] W. Gehrke, K. Gaedke: „Associative Controlling of Monolithic Parallel Processor Architectures“, IEEE Transactions on Circuits and Systems for Video Techology, Bd. 5, Nr. 5, S. 453-464, Oktober 1995

[GeGe84] S. Geman, D. Geman: „Stochastic Relaxation, Gibbs Distributions and the Byesian Restoration of Images“, IEEE Transactions on Pattern Analysis and Machine Intelligence, Bd. 6, Nr. 721-741, S. 721-741, November 1984

[GeHe96] J. Gealow, F. Herrmann, L. Hsu, C. Sodini: „System Design for Pixel-Parallel Image Processing“, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Bd. 4, Nr. 1, März 1996

[GöRe89] M. Gössel, B. Rebel, R. Creuzburg: „Speicherarchitektur und Parallelzugriff“, Berlin: Akademie-Verlag, 1998

[GoMe95] J. Goodenough, R. J. Meacham, J. D. Morris, N. L. Seed, P. A. Ivey: „A Single Chip Video Processing Architecture for Image Processing, Coding and Computer Vision“, IEEE Transactions on Circuits and Systems for Video Techology, Bd. 5, Nr. 5, S. 436-445, Oktober 1995

[GoSc00] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi M. Moe, R. R. Taylor: „PipeRench: A Reconfigurable Architecture and Complier“, Computer, p.70-77, April 2000

[HaHe98] R. Hartenstein, M. Herz, T. Hoffmann, U. Nageldinger: „Using Kress Arrays for Reconfigurable Computing“, Proceedings of the SPIE, Conference on Configurable Computing: Technology and Applications, Boston, USA, Bd. 3526, S. 150-161, November 1998

[HaSt00] S.D. Hayes, J. Stone, P.Y.K. Cheung, W. Luk: „Video Image Processing with the Sonic Architecture“, Computer, S. 50-57, April 2000

[HCMOS7] „Corelib HCMOS7“, Datenbuch, ST Microelectronics, Bretonischer Ring 4, D-85626 Grasbrunn, http://www.st.com

[HeMo97a] S. Herrmann, H. Mooshofer: „Complexity Analysis of a Color Segmentation Algorithm“, COST211ter Simulation Subgroup, Hannover, Germany, SIM(97)4,. März

1997

[HeMo97b] S. Herrmann, H. Mooshofer, W. Stechele, „A Toolbox Approach for Image Segmentation and Complexity Analysis“, Proceedings of the WIAMIS’97, Leuven-la-Neuve, S. 173-178, Juni 1997

[HeMo97c] S. Herrmann, H. Mooshofer, W. Stechele: „An Architecture Concept for a Segmentation Hardware Accelerator“, Proceedings of the DCIS`97, S. 655-660, September 1997

[HeMo99a] S. Herrmann, R.Sasportas, H. Mooshofer, J-C. Klein, F. Meyer, W. Stechele, „Application of Recursive Methods for Object Based Video Processing and Feature Extraction“, Proceedings of the WIAMIS’99, Berlin, S. 121-124, Juni 1999

[HeMo99b] S. Herrmann, H. Mooshofer, H. Dietrich, W. Stechele: „A Video Segmentation Algorithm for Hierarchical Object Representations and its Implementation“, IEEE Transaction on Circuits and Systems for Video Technology, Bd. 9, Nr. 8, p.1204-1215, Dezember 1999

[HeNi01] S. Hermann, U. Niedermeier, W. Stechele: „The Application Model of the MPEG-7 Reference Software“, Proceedings of the WIAMIS‘01, Tampere, Finnland, S. 83-88, Mai 2001

[HePa90] J. Hennessy, D. Patterson: „Computer Architecture - A Quantitative Approach“, San Fransisco: Morgan Kaufman Publishers, 1990

[Hos88] F. Hossfeld: „Vector-Supercompter“, Parallel-Processing, Bd. 7, S. 373-385, 1988

[Hut99] A. Hutter: „Eine flexible Architektur für blockbasierte Algorithmen in der Videocodierung“, Doktorarbeit am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München, Januar 1999

[HwTs89] K. Hwang, P. Tseng, D. Kim: „An Orthogonal Multiprocessor for Large Grain Scientific Computations“, IEEE Transactions on Computer, Bd. 38, Nr. 1, 1989

[Inf99] „HYB 39S64400/800/160BT(L) 64 MBit Synchonous DRAM“, Produktspezifikation, Infineon Technologies AG, St.-Martin-Str. 53, 81669 München, April 1999

[Iprof] iprof, anonymous ftp: ftp.lis.e-technik.tu-muenchen.de/pub/iprof

[Isatec] „Der ISATEC Systola 1024 Parallelrechner“, ISATEC Soft- und Hardware GmbH, Am Moorwiesengraben 27, D-24113 Kiel, Deutschland

[Jäh93] B. Jähne: „Digitale Bildverarbeitung“, 3. Auflage, Berlin/Heidelberg/NewYork: Springer-Verlag, 1993

[Jan99] F. Janku: „Analyse von Architekturvarianten für einen Koprozessor zur Segmentierung und Bildanalyse“, Diplomarbeit am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München, Mai 1999

[JeDi97] A. Jerraya, H. Ding, P. Kission, M. Rahmouni: „Behavioural Synthesis and Component Reuse with VHDL“, Dordrecht/Boston/London: Kluwer-Verlag, 1997

[Kau95] A. Kaup: „Modelle zur regionenorientierten Bildbeschreibung“, VDI Fortschrittsbericht, Reihe 10, Nr. 381, Düsseldorf: VDI-Verlag, 1995

[KaWi87] M. Kass, A. Witkin, D. Terzopoulus: „Snakes: Active Contur Models“, First International Conference on Computer Vision, S. 259-268, 1987

[Khoros] D. Argiro, S. Kubica, M. Young, S. Jorgensen: „Khoros: An Integrated Development Environment for Scientific Computing and Visualization“, Khoral Inc., 6200 Uptown Blvd NE Suite 200, Albuquerque, NM 87108, USA, http://www.kho-

ral.com, 2001

[Kis00] F. Kissling: „Untersuchung und Implementierung eines Speicherinterfaces für einen Koprozessor zur Segmentierung und Bildanalyse“, Diplomarbeit am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München, April 2000

[KlLe94] J-C. Klein, F. Lemonier, C. Fernandez: „Morphological Codec Simulation and demonstrator presentation“, Deliverable R2053/CMM/DS/012/b1 des Projektes Morpheco (Morphological Codec Simulation and demonstrator presentation), Fontainebleau, Dezember 1994

[KlLe95] J-C. Klein, F. Lemonnier, M. Gauthier, P. Peyrad: „Hardware Implementation of the Watershed Zone Algorithm Based on a Hierarchical Queue Structure“, IEEE Workshop on Nonlinear Signal Processing, 1995

[KlSa98] J-C. Klein, R. Sasportas: „Hardware Architectures for Real-Time Encoder Image Segmentation“, Projektbericht WP3-033 im Rahmen des MEDEA-Projektes MPEG Fo(u)r Mobiles, Januar 1998

[KlZa92] R. Klette, P. Zamperoni: „Handbuch der Operatoren für die Bildbearbeitung“, Braunschweig/Wiesbaden: Vieweg-Verlag, 1992

[Kna96] D. W. Knapp: „Behavioural Synthesis - Digital System Design Using the Synopsys Behavioural Compiler“, London: Prentice Hall, 1996

[KnBe97] J. Kneip, M. Berekovic, J. Wittenburg, W. Hinrichs, P. Pirsch: „An Algorithm Adapted Autonomous Controlling Concept for a Parallel Single-Chip Digital Signal Processor“, Journal of VLSI Signal Processing, Bd. 16, S. 31-40, Mai 1997

[Kne97] J. Kneip: „Objektorientierte Cache-Speicher für programmierbare monolithische Multiprozessoren in der digitalen Bildverarbeitung“, VDI Fortschrittsbericht, Reihe 9, Nr. 267, Düsseldorf: VDI-Verlag, 1997

[Kne98] J. Kneip: „Multimedia Instruction Set Extensions for General Purpose Processors and DSPs“, Projektbericht WP3-003 im Rahmen des MEDEA-Projektes MPEG Fo(u)r Mobiles, 1998

[KrKa98] K. Kreeger, A. Kaufman: „PAVLOV: A Programmable Architecture for Volume Processing“, Proceedings of SIGGRAPH, Eurographics Workshop on Graphics Hardware, S. 77-86, August 1998

[KrKa99] K. Kreeger, A. Kaufman: „Interactive Volume Segmentation with the PAVLOV Architecture“, Proceedings of the IEEE Parallel Visualization and Graphics Symposium, S. 61-119, 1999

[KuDi98] P. Kuhn, G. Diebel, S. Herrmann, A. Keil, H. Mooshofer, A. Kaup, R. Meyer, and W. Stechele: „Complexity and PSNR-comparison of several fast motion estimation algorithms for MPEG-4“, Proceedings of the SPIE, Application of Digital Image Processing XXI, Bd. 3460, S. 486-499, Juli 1998

[Kuh99] Kuhn P.: „Algorithms, Complexity-Analysis and VLSI-Architectures for MPEG-4 Motion Estimation“, Dordrecht/Boston/London: Kluwer-Verlag, 1999

[Kum97] A. Kumar: „The HP PA-8000 RISC CPU“, IEEE Micro, S. 27-30, März/April 1997

[Lak96] R. Lahkämper: „Analyse und Implementierung des Wasserscheidenverfahrens zur Objektsegmentierung in Grauwertbildern“, Diplomarbeit am Institut für Angewandte Mathematik der Universität Hamburg, September 1996

[Lan94] M. Lang: „Mensch-Maschine-Kommunikation 1“, Vorlesungsunterlagen, Lehr-

stuhl für Mensch-Mschine-Kommunikation, Technischen Universität München, 2. Auflage, Juni 1994

[LeSi00] M-H. Lee, H. Singh, G. Lu, N. Bagherzadeh, F.J. Kurdahi, E.M.C. Filho, V.C. Alves: „Design and Implementation of the MorphoSys Reconfigurable Computing Processor“, Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, Bd. 24, Nr. 2-3, S. 147-164, März 2000

[LeSm96] R. Lee, M.Smith: „Media Processing: A New Design Target“, IEEE Micro, S. 6-9, August 1996

[LiDo00] G. Linan, R. Dominguez-Castro, S. Espejo, E. Roca, P. Foldesy, A. Rodriguez-Vazquez: „Experimental Demonstration of Real-Time Image-Processing usingf a VLSI Analog Programmable Array Processor“, Proceedings of the SPIE, Bd. 3962, S. 235-245, 2000

[Lip00] C. Lipsky: „Entwurf einer flexibel konfigurierbaren Pixelverarbeitungseinheit für den Segmentierungs- und Bildanalyse-Koprozessor CONIAN“, Diplomarbeit am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München, August 2000

[LivdW99] P. Lieverse, P. van der Wolf, E. Deprettere, K. Vissers: „A Methodology for Architecture Exploration of Heterogenous Signal Processing Systems“, Proceedings of the IEEE Workshop on Signal Processing Systems, SiPS‘99, S. 181-190, 1999

[LuWy91] A. Lumsdaine, J. L. Wyatt, I. M. Elfadel: „Nonlinear Analog Networks for Image Smoothing and Segmentation“, Journal of VLSI Signal Processing, Bd. 3, S. 53-68, 1991

[MaJa92] J. Mao, A.K. Jain: „Texture Classification and Segmentation using Multiresolution Simulatanous Autoregressive Models“, Pattern Recognition, Bd. 25, Nr. 2, S. 173-188, 1992

[MeMa01] R. Mech, F. Marques: „Objective Evaluation Criteria for 2D-Shape Estimation Results of Moving Objects“, Proceedings of the WIAMIS’01, Tampere, Finnland, S. 23-28, Mai 2001

[MeNo95] A. Merle, D. Noguet, R. Lionti, D. David: „Using Queues in Mathematical Morphology: Algorithms Analysis and Propositions for a Hardware Implementation“, IEEE Workshop on Non Linear Signal and Image Processing, Griechenland, Bd. 1, Juni 1995

[MeTa97] J.D. Mellott, F. Taylor: „Very Long Instruction Word Architectures for Digital Signal Processing“, International Conference on Acoustics, Speech, and Signal Processing, ICASSP‘97, Bd. 1, S. 583-586, 1997

[MeWo97] R.Mech, M.Wollborn: „A Noise Robust Method for Segmentation of Moving Objects in Video Sequences“, Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, ICASSP‘97, Bd. 4, S. 2657-2660, April 1997

[Mey91] F. Meyer: „An Optimal Algorithm for the Watershed Line“, Dans RFIA, S. 847-857, 1991

[Mey98a] F. Meyer: „Levelings and Morphological Segmentation“, Workshop SIBGRAPI, Rio Del Janeiro, Brasilien, Oktober 1998

[Mey98b] F. Meyer: „The Levelings“, Mathematical Morphology and its Applications to Image and Signal Processing, Dordrecht/Boston/London: Kluwer-Verlag, S. 199-206, 1997

[MiBa97] J. Miteran, R. Bailly, P. Gorria: „Classification Board for Real-Time Image Segmentation“, Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal, ICASSP’97, Bd. 5, S. 4069-4072, 1997

[MoBh98] F. Moscheni, S. Bhattacharjee, M. Kunt: „Spatiotemporal Segmentation Based on Region Merging“, IEEE Transaction on Pattern Analysis and Machine Intelligence, Bd. 20, Nr. 9, S. 897-915, September 1998

[MoHe99] H. Mooshofer, S. Herrmann, W. Stechele: „A Motion Analysis Method for a Hierarchical Segmentation Algorithm and its Efficient Implementation“, Proceedings of the WIAMIS`99, S. 57-60, Juni 1999.

[MPEG4] R. Koenen: „MPEG-4 Overview (Version 18)“, ISO/IEC JTC1/SC29/WG11 N4030, Singpore, März 2001

[MPEG7] J.M. Martinez: „Overview of the MPEG-7 Standard (Version 5.0)“, ISO/IEC JTC1/SC29/WG11 N4031, Singpore, März 2001

[Mul97] P.J. Mulroy: „Video Content Extraction: Review of Current Automatic Segmentation Algorithms“, Proceedings of the WIAMIS’97, Leuven-la-Neuve, June 1997

[M0408] I. Corset, S. Bouchard, S. Jeannin, P. Salembier, F. Marques, M. Pardas, R. Morros, F. Meyer, B. Marcotegui: „Technical Description of SESAME“, ISOIEC JTC1/SC29/WG11, MPEG95/M0408, 1995

[M0921] P. Kuhn: „A portable Instruction Level Profiler for Complexity Analysis - Software“, ISO/IEC JTC1/SC29/WG11 M0921, Tampere, Finnland, 1996

[Nog97] D. Noguet: „A Massively Parallel Implementation of the Watershed Based on Cellular Automata“, Proceedings of the IEEE International Conference on Applications-Specific Systems, Architectures and Processors, S.42-52, 1997

[NoMe95] D. Noguet, A. Merle, D. Lattard, D. David, R. Lionti: „Queues for Advanced Morphological Operators: From Algorithms to a Data Dependent Architecture“, International Conference on Quality Control by Artificial Vision, QCAV‘95, Univ. Bourgogne, Le Creusot, France, p.141-149, 1995

[Nor98] K. Normoyle, et. al.: „Ultra-SPARC-IIi: Expanding the Boundaries of a System on a Chip“, IEEE Micro, S. 14-23, März/April 1998

[N2C] „CoWare N2C Design System“, Datenblatt, CoWare, 2845 Bowers Avenue, Santa Clara, CA 95051, USA, http://www.coware.com

[Ohm95] J-R. Ohm: „Digitale Bildcodierung“. Berlin/Heidelberg/NewYork: Springer-Verlag, 1995

[OkFu95] S. Okazaki, Y. Fujita, N. Yamashita: „A Compact Real-Time Vision System Using Integrated Memory Array Processor Architecture“, IEEE Transactions on Circuits and Systems for Video Techology, Bd. 5, Nr. 5, S. 446-452, Oktober 1995

[OMI] „1499 IEEE Standard Interferance for Hardware Description Models of Electronics Components“, Institute of Electrical and Electronics Engineers (IEEE), 1828 L Street, N.W., Suite 1202, Washington, D.C. 20036, USA, 1998

[ÖsBe97] A. Österling, Th. Benner, R. Ernst, D. Herrmann, Th. Scholz, W. Ye: „Hardware/Software Co-Design: Principles and Practice“, Kapitel „The COSYMA System“, Dordrecht/Boston/London: Kluwer-Verlag, 1997

[Pat97] D. Patterson et al: „Intelligent RAM (IRAM): the Industrial Setting, Applications, and Architectures“, Proceedings. International Conference on Computer Design, VLSI in Computers and Processors, S.2-7, 1997

[PC] „Protocol Compiler User‘s Manual“, Synopsys Inc. 700 E. Middlefield Rd., Mountain View, CA 94043, USA

[PeGe94] P. Pirsch, W. Gehrke, K. Gaedke, K. Hermann: „A Parallel VLSI Architecture for Object-Based Analysis-Synthesis Coding“, Proceedingd of the IEEE Workshop on Visual Signal Processing and Communications, S. 136-140, September 1994

[PIMM1] „PIMM1 User Guide“, Ecole des Minnes de Paris, Centre de Morphologie Mathématique, 35 rue Saint Honoré, 77305 Fontainebleau Cedex, Frankreich, 1990

[Pir96] P. Pirsch: „Architekturen der digitalen Signalverarbeitung“, Stuttgart: Teubner, 1996

[Pir98] P. Pirsch, H-J. Stolberg: „VLSI Implementations of Image and Video Multimedia Processing Systems“, IEEE Transactions on Circuits and Systems for Video Technology, Bd. 8, Nr. 7, S. 878-891, November 1998

[Ptolemy] E.A. Lee: „Overview od the Ptolemy Project“, Department of Electrical Enigneering and Computer Science, University of California, Berkeley CA 94726, USA, http://www.ptolemy.eecs.berkeley.edu

[Rad93] B. Radig: „Verarbeiten und Verstehen von Bildern“, München: Oldenbourg-Verlag, 1993

[RaJa97] N.K. Ratha, A.K. Jain: „FPGA-based Computing in Computer Vision“, Fourth IEEE International Workshop on Computer Architecture for Machine Perception, CAMP'97, S.128-137, 1997

[Rob97] F. Robin: „Etude d'architectures VLSI numériques parallèles et asynchrones pour la mise en œuvre de nouveaux algorithmes d'analyse et rendu d'images“, Thèse ENST, 1997

[RoCh93] T. Roska, L.O. Chua: „The CNN Universal Machine: An Analogic Array Computer“, IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing, Bd. 40, Nr. 3, S. 163-173, März 1993

[RöKn96] K. Rönner, J. Kneip: „Architecture and Applications of the HiPAR Video Signal Processor“, IEEE Transactions on Circuits and Systems for Video Technology, Bd. 6, Nr. 1, S. 56-66, Februar 1996

[Roj93] R. Rojas: „Theorie der neuronalen Netze“, Berlin/Heidelberg/NewYork: Springer-Verlag, 1993

[Rug99a] I. Ruge: „Integrierte Schaltungen 1“, Vorlesungsmanuskript, Vorlesung gehalten im Sommersemester 1999 an der Technischen Universität München

[Rug99b] I. Ruge: „Integrierte Schaltungen 2“, Vorlesungsmanuskript, Vorlesung gehalten im Wintersemester 1999/2000 an der Technischen Universität München

[SaBr96] P. Salembier, P. Brigger, J.R. Casas, M. Pardas: „Morphological Operators for Image and Video Compression“, IEEE Transactions on Image Porcessing, Bd. 5, Nr. 6, S. 881-898, Juni 1996

[SaKl99] R. Sasportas, J-C. Klein: „Morphological Filters Implementation based on a Co-Design Approach“, 10th International Conference on Image Analysis and Processing, S. 154-159, 1999

[SaMa97] P. Salembier, F. Marques, M. Pardàs, J.R. Morros, I. Corset, S. Jeannine, L. Bouchard, F. Meyer, B. Marcotegui: „Segmentation Based Video Coding Allowing the Manipulation of Objects“, IEEE Transaction on Circuits And Systems for Video Technology, Bd. 7, Nr. 1, S. 60-74, Februar 1997

[Sam99] „512x16 bit Low Power CMOS Static RAM“, Produktspezifikation, Samsung Electronics CO. LTD., One Samsung Place, Ledgewood, NJ 07852, USA, http://www.samsungelectronics.com, Juni 1999

[Sch94] J. Schönfeld: „Kompakte Implementierung konturorientierter Bildverarbeitungssysteme mit VLSI-Bausteinen“, VDI Fortschrittsbericht, Reihe 10, Nr. 321, Düsseldorf: VDI-Verlag, 1994

[ScMa87] I. Scherson, Y. Ma: „Vector Computations on an Orthogonal Memory Access Multiprocessing System“, 8th IEEE Symposium on Computer Arithmetic, S. 28-37, 1987

[ScWa98] J. Schütz, R.Wallace: „A 450MHz IA32 P6 Family Microprocessor“, Proceedings of the IEEE International Solid-State Circuits Conference, ISSCC‘98, S. 236-237, Februar 1998

[Ser82] J. Serra: „Image Analysis and Mathematical Morphology“, New York: Academic Press, 1982

[Sha97] A. K. Sharma: „Semiconductor Memories: Technology, Testing and Reliability“, New York: IEEE Press, 1997

[SIA97] „The National Roadmap for Semiconductors“, Semiconductor Industry Association, http://www.sematech.org, 1997

[SiFa96] M.Y. .Siyal, M. Fathy: „Triple RISC Image Operator for Real-Time Image Processing Applications“, Electronic Letters, Bd. 32, Nr. 24, S. 2224-2225, November 1996

[SiSe97] T. Sikora and H. Seguin: „The European COST 211ter Group - Research on Redundancy Reduction Techniques and Content Analysis for Multimedia Services“, Proceedings of the WIAMIS`97, S. 127-134, Juni 1997

[SPS3] „SPS3 Synchronous Single Port RAM“, Datenblatt, ST Microelectronics, Bretonischer Ring 4, D-85626 Grasbrunn

[SPEC] „SPEC CPU2000 Benchmark Suite“, Standard Performance Evaluation Corporation (SPEC), 6585 Merchant Place, Suite 100, Warrenton, VA 20187, USA, http://www.spec.org

[SPW] „SPW User‘s Manual“, Cadence Design Systems, 919 E. Hillsdale Blvd., Foster City, CA 94404 USA

[SuAr98] K. Suzuki, T. Arai, K. Nadehara, I. Kuroda: „V830R/AV: Embedded Multimedia Superscalar RISC Processor“, IEEE Micro, S. 36-47, März/April 1998

[SystemC] „Functional Specification for System C 2.0“, Open SystemC Initiative, http://www.systemc.org

[tHa95] K. Ten Hagen: „Abstrakte Modellierung digitaler Schaltungen“, Berlin/Heidelberg/NewYork: Springer-Verlag, 1995

[Thi98] T. Thiel: „Entwurf eines konfigurierbaren Verarbeitungsteils für einen Koprozessor zur Segmentierung und Bildanalyse“, Diplomarbeit am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München, April 1998

[TM1300] „TriMedia TM-1300: Programmable Media Prozessor“, Datenblatt, Philips Semiconductors Trimedia Business Line, 811 East Arques Avenue M/S 71, Sunnyvale CA 94088-3409, http://www.trimedia.philips.com

[TriA7] „Triscend A7 - 32 bit Configurable System-on-Chip“, Product Brief, http://www.triscend.com/products/TextTechLit.html, März 2001

[TriE5] „Triscend E5 - 8 bit Configurable System-on-Chip“, Product Brief, http://www.triscend.com/products/TextTechLit.html, März 2001

[umorph] Micromorph for Windows, Ecole des Minnes de Paris, Centre de Morphologie Mathématique, 35 rue Saint Honoré, 77305 Fontainebleau Cedex, Frankreich, http://cmm.ensmp.fr/Micromorph

[VCC] „Cadence Virtual Component Co-Design (VCC)“, Datenblatt, Cadence Design Systems Inc., 2655 Seely Avenue, San Jose, CA 95134, USA, http://www.cadence.com

[VeWe95] P.L. Venetianer, F. Werblin, T. Roska, L.O. Chua: „Analogic CNN Algorithms for Some Image Compression and Restoration Tasks“, IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications, Bd. 42, Nr. 5, S. 278-284, Mai 1995

[ViSo91] L. Vincent, P. Soille: „Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations“, IEEE Transactions on Pattern Analysis and Machine Intelligence, Bd. 13, Nr. 6, S. 583-598, Juni 1991

[VSI] „VSI Alliance Architecture Document“, VSI Alliance, 15495 Los Gatos Blvd., Suite 3, Los Gatos, Calif. 95032, USA, http://www.vsi.org

[WaHo96] M.A.Wahab, J.M. Holden, J.S. Rees: „Reconfigurable Image Coprocessor for Mathematical Morphology“, Proceedings of the SPIE, Bd. 2727, S. 1027-1033, 1996

[Wan98] D. Wang: „Unsupervised Video Segmentation Based on Watersheds and Temporal Tracking“, IEEE Transaction on Circuits and Systems for Video Technology, Bd. 8, Nr. 5, S. 539-546, September 1998.

[Wid96] D. Widmann, H. Mader, H. Friedrich: „Technologie hochintegrierter Schaltungen“, Berlin/Heidelberg/NewYork: Springer-Verlag, 1996

[Wild] Produktübersicht, http://www.annapmicro.com/products.html, März 2001

[WiSh92] D. J. Williams, M. Shah: „A Fast Algorithm for Active Contours and Curvature Estimation“, CVGIP Image Understanding, Bd. 55, Nr. 1, S. 14-26, Januar 1992

[WoMe97] M. Wollborn, R. Mech: „A Noise Robust Method for Segmentation of Moving Objects in Video Sequences“, Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, ICASSP‘97, München, Bd. 4, S. 2657-2660, April 1997

[YeWo96] T-Y. Yen, W. Wolf: „Hardware-Software Cosynthesis of Distributed Embedded Systems“, Dordrecht/Boston/London: Kluwer-Verlag, 1996

[Yoh00] Binoy Yohannan: „Get the Architecture Right“, White Paper, Innoveda, http://www.innoveda.com/products/eArchitect/white_paper.htm, Juni 2000

[Zuk99] R. Zukunft: „Architekturentwurf und Modellierung des Scan-Controllers eines Segmentierungsprozessors“, Diplomarbeit am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München, Januar 1999