Konzeption und Entwicklung einer Architektur zur parametrisierbaren Filterung von Grauwertbildern und deren ASIC–Realisierung

Norbert Peschel

Universität Hamburg
Fachbereich Informatik

Hamburg, August 1995

Zusammenfassung

Besonders im Bereich der Bildverarbeitung zeigt sich, daß Mikroprozessoren und DSPs trotz steigender Taktfrequenzen nicht schnell genug sind, um den Anforderungen einer Echtzeitverarbeitung zu genügen. Im Rahmen dieser Arbeit wird ein ASIC (GIPP – General Image PreProcessor) entworfen, welches die zeitkritischen Aufgaben der Bildvorverarbeitung bearbeiten soll. Als Anwendungsgebiet wird die automatische Klassifikation von Hölzern behandelt.

Dazu wird im ersten Teil ein Konzept erarbeitet, daß die Extraktion typischer Bildmerkmale unter Echtzeitbedingungen ermöglicht. In dem so gewonnenen mehrdimensionalen Merkmalsraum findet dann die eigentliche Klassifikation der Holzproben statt. Die Tragfähigkeit dieses Ansatzes wird mit Hilfe von Softwaresimulationen überprüft.

Der zweite Teil der Arbeit beschäftigt sich mit dem Chipentwurf des GIPP. Das IC implementiert einen programmierbaren 2-D Filter. Die Größe der Filtermatrizen kann an die Aufgabenstellung angepaßt werden. Ein Chip kann Faltungen mit 40 Koeffizienten durchführen. Die Kombination mehrerer ICs erlaubt die Umsetzung größerer Filter, während zwei Ausgabebusse die gleichzeitige Berechnung kleinerer Filter ermöglichen. Durch besondere Maskierungstechniken können Teilbilder innerhalb eines Eingabebildes (bis $1024 \times n$) unterschieden werden. Dadurch wird die Behandlung von Bildpyramiden in der Hardware effizient unterstützt. Da die Verzögerungszeit des Datenstroms durch ein IC konstant ist (abhängig von der internen Pipeline), können mehrere ASICs kaskadiert werden, um so ein komplexeres Bildverarbeitungssystem aufzubauen. Jedes IC der Pipeline verarbeitet einen kontinuierlichen Datenstrom unter Echtzeitbedingungen.

Abstract

It is well known that in image processing applications microprocessors and DSPs are not fast enough to meet the timing requirements for real-time operation. Within the scope of this work an ASIC (GIPP – General Image PreProcessor) is designed, to accelerate the time consuming tasks during image pre-processing. The target application area is the automatic classification of wood boards.

In the first part of this report an image processing concept is developed, that allows to extract typical image features under real-time conditions. Within the multidimensional feature space the classification of the samples is done. The concept is validated through extensive software simulations.

The second part of this work concentrates on the IC design of the GIPP. The circuit implements a programmable 2-D filter. The size of the filter matrices can be chosen according to the task to be performed. One chip can compute convolutions with up to 40 coefficients. Combining multiple ICs larger filters can be implemented, while two output busses allow the computation of smaller filters within one chip. Through special masking techniques parts of an input image (of up to $1024 \times n$) can be separated to deal efficiently with image pyramides in hardware. Since the data delay through one IC is constant (depending on the pipeline configuration), several ASICs can be cascaded to build a complex image processing system. Each IC in the pipeline works on a continuous pixel stream under real-time conditions.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Bildverarbeitung komplexer optischer Strukturen

Diese Arbeit beschäftigt sich mit der Detektion optischer Strukturen und dazu benötigter Bildverarbeitungsmethoden und Hardware. Das Anwendungsgebiet liegt im Bereich der maschinellen Verarbeitung von Hölzern. Dieser Bereich ist personalintensiv, da alle Hölzer vor der Weiterverarbeitung in Güteklassen eingeteilt werden müssen. Um hier rationalisieren zu können, bedarf es einer automatischen, optisch basierten Klassifikation von Hölzern. Da konventionelle Rechner zur Klassifikation eines Holzes um Größenordnungen mehr Zeit brauchen als ein Mensch, kommen nur Systeme mit beschleunigender Spezialhardware in Betracht. Ein Bildverarbeitungsprozessor, der sich für diesen Zweck eignet, sollte hier entwickelt werden.

1.2 Problemklassifikation

Bildsensoren liefern üblicherweise rechtwinklig gerasterte Bilder. Den einzelnen Bildpunkten, sogenannten *Pixeln* (picture elements), wird zum Beispiel bei Grauwertbildern je ein Helligkeitswert zugeordnet. Der Mensch nimmt dagegen Bilder als im Gesichtsfeld angeordnete *optische Merkmale* visuell wahr. Da der Begriff "Merkmal", der allgemein als Eigenschaft von Objekten definiert ist, in unterschiedlichen Zusammenhängen bei der sprachlichen Beschreibung von Bildern und für Ergebnisse von Bildverarbeitungsoperatoren auf diesen gebraucht wird, wird im folgenden zwischen visuellen und klassifizierenden Merkmalen unterschieden. Als visuelle Merkmale nimmt der Mensch z. B. Bildkanten und *Texturen* in einem Bild wahr. Diese Merkmale werden jedoch sofort unwillkürlich in eine visuelle Bildrepräsentation von *Objekten* umgesetzt. So *sieht* man beim Anblick eines Holzes z. B. Astlöcher oder verschiedenartige Maserung. Bei der Bildbeschreibung wird von den gesehenen Objekten selbst abstrahiert und stattdessen von Instantiierungen bekannter *Objektklassen* gesprochen. Als Ersatz für stärker als üblich von der Objektklasse abweichende visuelle Merkmale einzelner Objekte oder in Bereichen, in denen die Sprache nicht differenziert genug ist, versieht man Objekte bei Bedarf noch mit *modifizierenden Merkmalsbegriffen*. Je nach dem Anspruch an Detailliertheit der Beschreibung, die erforderlich ist, werden zur Beschreibung Objekte verschiedener Abstraktionsebenen gewählt, die untereinander Hierarchien bilden.

Der Eindruck, den ein Blick auf ein Holz bei einem Sortierer hinterlassen könnte, ist etwa folgender: "Ich sehe ein lebhaft gemasertes Holz mit fünf beieinanderliegenden Astlöchern". Neben dem Holz als Objekt, werden in dieser Bildbeschreibung noch fünf Objekte der Objektklasse Astlöcher aufgeführt, die in der Objektklassenhierarchie eine Ebene tiefer als die Objektklasse Hölzer anzusiedeln ist. Es wurde bei der Beschreibung zwar von der absoluten Position der Astlöcher abstrahiert, ihre relative Position zueinander war jedoch bedeutsam genug, um das Holz noch mit dem modifizierenden Merkmalsbegriff "beieinanderliegende Astlöcher" zu versehen. Bei der Maserung ist gleiches zu beobachten. Der modifizierende Merkmalsbegriff "lebhafte Maserung" abstrahiert zwar von der Maserungsrichtung im einzelnen, weist jedoch auf stärkere Abweichung der Maserungsrichtung zueinander hin, und zwar an irgendwelchen Stellen des Holzes.

Die Abstraktionsleistung der visuellen Merkmalswahrnehmung und Objekterkennung, die beim Menschen unbewußt von den tieferen Schichten der Netzhaut bzw. dem Gehirn geleistet wird, muß bei automatisch klassifizierenden Bildverarbeitungssystemen von Hard– oder Software geleistet werden. Zudem werden bei der Begründung einer Klassifikation ähnliche Ansprüche gestellt, wie bei einer qualifizierten Bildbeschreibung, die ein Sortierer abgibt. Dieser wird Objekte und Merkmale nur unter den Gesichtspunkten beschreiben, denen er seine Aufmerksamkeit widmet, die also für die Klassifikation relevant sind. Auch wenn nur das Klassifikationsergebnis gefragt ist, wird man bei dessen Auswahl Strategien verfolgen müssen, die denjenigen zur Gewinnung einer Bildbeschreibung äquivalent sind.

Bei der Realisierung automatischer Klassifikationssysteme scheint es sinnvoll, die Aufgabenteilung in Objekterkennung und visuelle Merkmalsbewertung einerseits und Klassifikation andererseits beizubehalten. In der ersten Bildverarbeitungsstufe werden dann Verfahren benötigt, die das Äquivalent visueller Merkmale bei der automatischen Bildverarbeitung, die klassifizierenden Merkmale berechnen, und mit ihrer Hilfe Objekte detektieren, oder sie zur statistischen Auswertung, z. B. in Form eines Histogramms, weiterreichen. Die zweite Bildverarbeitungsstufe nimmt, basierend auf den Detektionsergebnissen und den statistischen Meßwerten der ersten Verarbeitungsstufe, die Klassifikation der Hölzer vor.

Die Trennung der Merkmalsbegriffe bei der automatischen und menschlichen Klassifizierung ist nötig, da sich der Abstraktionsgrad und die Qualität der Merkmale unterscheiden. Klassifizierende Merkmale haben im Gegensatz zu visuellen Merkmalen den Charakter von Meßwerten. Prinzipiell können Objekte auch durch klassifizierende Merkmale detektiert werden, die Menschen gar nicht als visuelle Merkmale wahrnehmen können (z. B. UV– oder IR–Bilder; Ortsfrequenz des Bildrauschens, siehe dazu [*Jähne 1991*, Farbtafel 11]). Sollen jedoch genau die Objekte in einem Bild detektiert werden, die auch ein Mensch visuell wahrnimmt, ist es sinnvoll sich bei der Suche nach klassifizierenden Merkmalen an der visuellen Informationsverarbeitung des Menschen zu orientieren.

Die Klassifikation von optisch wahrnehmbaren Teilobjekten des Holzes und die Klassifikation eines ganzen Holzstücks in Güteklassen müssen unterschieden werden. Da hier Methoden der einfacheren Bildverarbeitung eingesetzt werden und die Klassifikation des Holzes als Ganzes nur thematisch gestreift wird, ist im folgenden, wenn nicht anders vermerkt, von Klassifikation im Zusammenhang mit Teilobjekten des Holzes die Rede. Solche Objekte können aus Sicht der Bildverarbeitung dadurch charakterisiert werden, daß ihre visuellen Merkmale von denen der Normalstruktur (Hintergrund) signifikant abweichen. Mit Hilfe entsprechender Objektbegriffe läßt sich das Bild dann — der Bildverarbeitungsaufgabe angemessen — sprachlich beschreiben.

Zur Detektion von Wurmlöchern, könnten die Eigenschaften "ist schwarz", "ist rund", "Struktur ist kleiner als 2 mm" an jeder Bildposition berechnet werden. Für sich genommen sind dies alles notwendige Bedingungen für ein Wurmloch. Treffen alle zusammen, ist allerdings mit hoher Wahrscheinlichkeit von einem Wurmloch auszugehen. Der Abstraktionsschritt, von — für ein Objekt möglichst charakteristischen — Merkmalen auf das Auftreten dieses Objektes selbst zu schließen, ist die Aufgabe eines Klassifikators. Die bei der technischen Realisierung dafür benötigten charakteristischen Merkmale, die das Ergebnis von Rechenoperationen auf dem Originalbild sind, werden im folgenden zur eindeutigen Unterscheidung *klassifizierende Merkmale* genannt. Im allgemeinen wird die Detektion jeder Objektklasse in einem Bild von einem eigenen Klassifikator abgedeckt, der je eine Teilmenge der berechneten klassifizierenden Merkmale nutzt. Dazu muß gewährleistet sein, daß in der Menge aller klassifizierenden Merkmale genügend Information enthalten ist, um überhaupt Objekte sicher detektieren zu können. Der Aufwand der Realisierung des Klassifikators soll zudem gering sein.

Der Sinn der Berechnung klassifizierender Merkmale liegt darin, den Informationsgehalt des sie umgebenden Bildinhaltes unter einem charakteristischen Aspekt in einem einzigen Wert zu konzentrieren. Zur Detektion von Objekten dient ein Klassifikator dessen Aufgabe es ist, Objekte der für ihn charakteristischen Objektklasse auf der Basis der klassifizierenden Merkmale zu detektieren. Der Klassifikator verarbeitet also Bilder, in denen in jedem Pixel der Wert je eines klassifizierenden Merkmals seiner Umgebung kodiert ist, zu einem Binärbild, das dem Detektionsergebnis Objekte gefunden/ nicht gefunden an der Position dieses Pixels entspricht. Zur Berechnung von statistischen Größen eines klassifizierenden Merkmals im Bereich eines Bildausschnittes, wird eine Verteilung der Merkmalswerte benötigt, die als Histogramm weitergegeben werden kann. Als Bildausschnitt des Histogramms kann das Vollbild, oder eine Partition desselben in Form von Bildkacheln dienen.

Merkmalsberechnung

Zur Lösung von Bildverarbeitungsaufgaben werden gewöhnlich für jede Koordinate im Originalbild mindestens ein, meistens jedoch mehrere klassifizierende Merkmale berechnet. Da das Bild gerastert vorliegt, fallen diese Berechnungen für jede einzelne Pixelposition an.

Die Werte der klassifizierenden Merkmale für jedes Pixel liegen danach kodiert in Form von Bildern vor. Jedes Pixel dieser Bilder repräsentiert nun allerdings den Wert des klassifizierenden Merkmals an seiner Position und nicht — wie im Originalbild — einen Helligkeitswert. Bei der Berechnung können Zwischenergebnisse in Form von Bildern anfallen, die noch kein klassifizierendes Merkmal und nicht mehr das Originalbild repräsentieren. Um sich auf diese Bilder beziehen zu können — meistens tauchen sie nur einmalig auf —, wird z. B. die Rede davon sein, "Pixel mit Koeffizienten zu multiplizieren". Gemeint ist damit, die Werte der Pixel eines Bildes, das nur ein Zwischenergebnis repräsentiert, mit einem Koeffizienten zu multiplizieren und in Pixeln eines neuen Bildes zu kodieren, das wiederum ein Zwischenergebnis darstellt.

Die Menge von klassifizierenden Merkmalen, die zur Detektion von Objekten aus dem Originalbild berechnet werden, bilden kartesisch verknüpft für jedes Objekt einen Merkmalsvektor in einem von den einzelnen klassifizierenden Merkmalen aufgespannten mehr–dimensionalen Merkmalsraum. Da in diesem zu jedem Pixel alle benötigten Merkmale berechnet vorliegen, beinhaltet der Merkmalsraum ebensoviele Merkmalsvektoren, wie das Bild Pixel enthält: Die Berechnung der klassifizierenden Merkmale bezüglich jedes Pixels bildet seine Bildposition auf einen Punkt im Merkmalsraum ab, dessen Koordinaten die Komponenten seines Merkmalsvektors bilden.

Operatoren zur Merkmalsberechnung

Standardoperatoren zur Merkmalsberechnung, die sich hardwareunterstützt problemlos in Echtzeit ausführen lassen, sind Punkt–, Faltungs– und Rangordnungsoperationen. Die Berechnung von Merkmalen kann unter Benutzung dieser Operatorenklassen mehrstufig erfolgen. Zur Analyse und zum Entwurf von Faltungsfiltern gibt es zahlreiche mathematische Verfahren im Orts– und Ortsfrequenzraum, sowie Transformationen zwischen diesen (s. S.476, [*Bronstein 1962*]).

Ein Problem lokaler Operatoren ist allerdings ihr begrenztes Werteinzugsgebiet. Objekte, die größer sind, können nicht als ganzes wahrgenommen werden. Über sie lassen sich somit keine Merkmale berechnen, allenfalls über ihre Randbereiche. Ein Ausweg besteht darin, das Bild in verschiedenen Auflösungsstufen zu verarbeiten, so daß jedes Objekt in der seiner Größe entsprechenden Bildauflösung optimal im Einzugsgebiet seines Merkmalsoperators liegt.

Klassifikation

Die Leistung jedes Klassifikators entspricht formal einer Abbildung von mehreren Bildern, die je ein klassifizierendes Merkmal enthalten, auf ein Binärbild, das die Detektionsantwort zu jeder Bildposition enthält. Die einfachste Klasse von Klassifikatoren detektiert Objekte über den Schwellenwert einer Linearkombination der klassifizierenden Merkmale. Ein solcher linearer Klassifikator sollte zur Detektion angestrebt werden. Komplexere Klassifikatoren berechnen z. B. ein Ähnlichkeitsmaß der klassifizierenden Merkmale zu dem eines typischen Vertreters der zu detektierenden Objektklasse. Dessen klassifizierende Merkmale lassen sich durch Mittelung der Merkmale aller Objekte der Merkmalsklasse, die in einer repräsentativen Stichprobe enthalten sind, berechnen.

Um Objekte an einer Stelle detektieren zu können, muß man ihnen, obwohl sie meist eine Ausdehnung besitzen, eine punktförmige Position zuordnen. Dazu kann z. B. der Flächenschwerpunkt dienen. Das Ergebnis einer Objektdetektion auf einem gerasterten Bild besteht an sich in Bildkoordinaten gefundener Objekte. Das Detektionsergebnis für alle Bildpositionen kann als Binärbild kodiert werden, in dem jedes Pixel die binäre Detektionsantwort enthält, ob an seiner Bildposition ein Objekt gefunden wurde oder nicht. Eine Darstellung der Verarbeitungsstufen maschineller Bildverarbeitungssysteme liefert Abb. 1.1. Nach Berechnung einer Bildpyramide, die noch eingeführt wird, findet die Berechnung klassifizierender Merkmale statt. Die Objekt–Detektionsantwort in Form eines Binärbildes bildet die Ausgabe der “einfacheren Bildverarbeitung”.

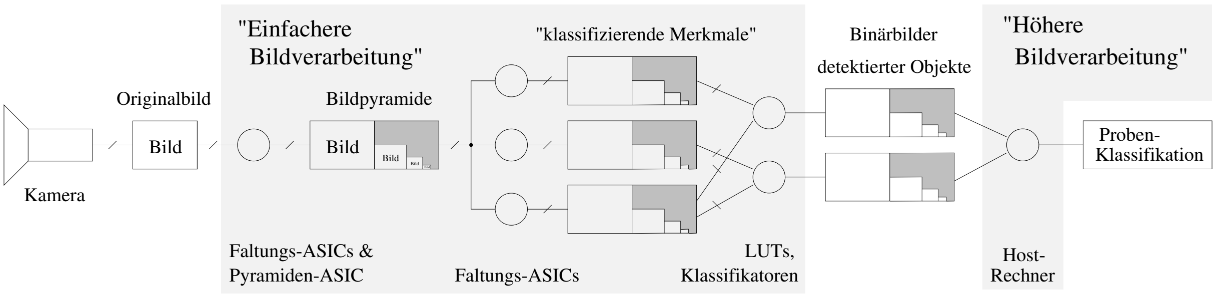


Abbildung 1.1: Bildverarbeitungsstufen

Es gilt dann, das vorliegende Holzstück in Güteklassen einzuteilen. Diese Aufgabe hat einen anderen Charakter als die beiden vorherigen, denn es fallen keine größeren Datenmengen mehr an, da die Verarbeitung nicht mehr pixelorientiert erfolgt. Sie kann durch Software auf einem echtzeitfähigen Betriebssystem realisiert werden. Die hardwareunterstützte Berechnung klassifizierender Merkmale ist jedoch Gegenstand dieser Arbeit — nicht diese Software zu entwickeln.

Zur Detektion von Objekten auf der Basis mehrerer klassifizierender Merkmale kann im einfachsten Fall von den Werten der Merkmale zuerst eine Linearkombination gebildet werden, die über einen Schwellenwertoperator das Detektionergebnis liefert. Ein solcher Klassifikator, der eine Linearkombination bildet und über einen Schwellenwertvergleich das Ergebnis bestimmt, heißt linearer Klassifikator. Er läßt sich problemlos in Hardware realisieren und kann daher vorteilhaft zur Detektion eingesetzt werden. Bei wenigen klassifizierenden Merkmalen kann sogar ein allgemeiner Klassifikator in Form von Funktionstabellen (Look–Up–Tables) durch RAM–Bausteine realisiert werden. Um die Koeffizienten zur Bildung der Linearkombination zu erhalten, kann auf der Basis einer repräsentativen Bildstichprobe zuerst eine Clusteranalyse durchgeführt werden, um herauszufinden, ob die Punkte im Merkmalsraum gleichverteilt oder aber gehäuft auftreten. Anschließend wird mittels einer Diskriminanzanalyse festgestellt, ob und welchen Klassen eventuell gefundene Cluster im Merkmalsraum entsprechen. Die Clusterdurchmesser im Vergleich zur Clusterdistanz und ihre Anordnung erlauben eine Abschätzung über die Sicherheit, mit der sich über die gewählten Merkmale überhaupt oder mittels eines linearen Klassifikators insbesondere, klassifizieren läßt.

Ein anderer Weg zur Klassifikation ist es, statistische Maße als Merkmale heranzuziehen. Diese werden für relativ große disjunkte Bildbereiche, sogenannte Bildkacheln, in Form von Histogrammen oder Co–Occurrenzmatrizen zur weiteren statistischen Auswertung gewonnen. Bildobjekte können hier zwar nicht genauer als eine Kachelgröße lokalisiert werden, dafür ist das Verfahren jedoch relativ einfach. Es muß nicht unbedingt als Alternative zu den o. a. Detektionsverfahren gesehen werden, sondern kann optional auch auf einzelne klassifizierende Merkmale anstatt oder zusätzlich zu einem Klassifikator angewendet werden.

Die Aufgaben der einfachen Bildverarbeitung im Gegensatz zur komplexeren Bildverarbeitung und –interpretation enden dort, wo eine Weiterverarbeitung auf der Ebene einzelner Pixel nicht mehr möglich ist. Da alle Verarbeitungsschritte prinzipiell auch im Rechner erfolgen könnten, kann der Sinn von Hardwareunterstützung in diesem Bereich nur in der durch spezialisierte Hardware möglichen Geschwindigkeitssteigerung der Merkmalsklassifikation liegen.

Datenreduktion mittels Klassifikation

Ziel der "einfacheren Bildverarbeitung" ist es, von *bedeutungslosen* Bildinhalten zu abstrahieren und nur die Koordinaten oder Meßwerte von *interessanten* Bildinhalten zur Verarbeitung weiterzugeben. Dieses ist allerdings eine hochgradig problemspezifische Aufgabe, die individuelle Untersuchungen erfordert. In vielen Bereichen interessieren beispielsweise Bildkanten, da sie Objektgrenzen repräsentieren oder auch Bildskelette, z. B. bei der Schrifterkennung, wo die Liniendicke selbst kein informationstragendes Merkmal ist.

Verschiedene Anwendungen der Bildverarbeitung unterscheiden sich stark in der Komplexität der Verarbeitungsschritte. Der triviale Fall ist der, in dem nur statistische Merkmale des Gesamtbildes oder Merkmale bekannter Bildpositionen als Meßwerte benötigt werden. Klassifikatoren oder kontextabhängige Bildinterpretationen werden hier nicht benötigt. Eine Anwendungsklasse dieser Art wären z. B. Frequenzfilterungen von Signalen zu Meßzwecken.

Damit die komplexere Bildverarbeitung und –interpretation nicht den Gesamtdurchsatz als Engpaß (*bottleneck*) begrenzt, muß eine erhebliche Datenreduktion durch die "einfachere Bildverarbeitung" erfolgen. Ein Mittel für diesen Zweck ist es, nur noch Binärbilder, die für jedes Pixel die Detektionsantwort eines Klassifikators enthalten, weiterzugeben. Dieses kann auch in einer reduzierten Bildauflösung erfolgen.

Bei der Detektion von Objektklassen, die nicht genügend stark abstrahieren, z. B. von Bildkanten, kann das Problem auftreten, daß die Anzahl der detektierten Objekte, bei Hölzern etwa die Kantendetektionsantwort entlang der Maserungslinien, nicht bedeutend geringer als die Pixelanzahl ist. Dann kann die folgende komplexere Bildverarbeitung die Datenmenge nicht mehr bewältigen. In der Objekthierarchie sollte die “einfachere Bildverarbeitung” auf der Ebene Objekte detektieren, die eine deutliche Reduktion der Datenmenge mit sich bringen. Diese können dann durch die komplexere Bildverarbeitung weiterverarbeitet werden.

Bildinterpretation und –auswertung

Bei manchen Verfahren der komplexeren Bildverarbeitung erfolgt danach noch eine Restauration von unvollständig detektierten Kanten und eventuell eine Zuordnung von Pixeln zu bestimmten Objekten (Segmentierung). Solche Aufgaben sind aufwendiger in Echtzeit zu realisieren, als die Merkmalsberechnung mit den oben genannten Standardverfahren, und sollten daher zumindest bei Bildern voller Auflösung möglichst umgangen werden. Softwarelösungen sind in Echtzeit nur möglich, wenn die Anzahl gefundener Bildobjekte, etwa Kanten, von vornherein bekannt und begrenzt ist, z. B. Bestimmung der Lage von bekannten Teilen auf einem Förderband. Ansonsten sind sie wegen der unvorhersagbaren Anzahl gefundener Objekte und der davon abhängigen Laufzeit unbrauchbar.

Die Bildinterpretation schließlich, die Bilder nicht nur als Summe von Objekten, sondern auch unter kontextabhängigen Gesichtspunkten dieser Objekte untereinander beschreibt (z. B. das vier Kanten unter bestimmten Voraussetzungen eine Raute bilden), kann je nach gewünschter Aussage sehr unterschiedlich erfolgen. Hierzu können Modelle der Objektwelt, statistische oder lageinvariante Größen der zu detektierenden Bereiche oder ganzer Bildteile, geometrische Abbildungsbeziehungen u. a. von dreidimensionalen Objekten herangezogen werden. Im Falle der Holzklassifikation wären außer DIN–gerechter Sortierung nach Oberflächengesichtspunkten z. B. auch Analysen der statischen Belastbarkeit von durch Astlöcher mehr oder weniger geschwächtem Bauholz denkbar.

Alternative Wege

Eine völlig unterschiedliche Herangehensweise wäre eine massiv parallele Rechnerstruktur aus vielen Einzelprozessoren, die in der Lage wäre, jede Maserungslinie zu verfolgen. Neben dem Kommunikationsproblem — die Prozessoren müssen ihren Bildbereich verlassende Linienelemente an ihre Nachbarprozessoren melden und die gewonnenen Ergebnisse nicht verteilt, sondern zentral ausgeben — wäre auch der Bewertungsalgorithmus neu zu entwickeln.

Da das menschliche Auge sich nicht daran stört, wenn bei Hölzern Maserungslinien ein wenig unterschiedlich verlaufen, solange nur Dichte und Richtung beibehalten wird, stellt sich die Frage, ob dieser Aufwand nötig ist, oder ob es nicht genügt lokale Merkmale zur Klassifikation heranzuziehen. Als Naturprodukt läßt sich für Holz ohnehin keine Solloberfläche auf der Ebene einzelner Maserungslinien vorgeben, vielmehr wird die Maserung fehlerfreier Hölzer vom Auge als Textur wahrgenommen, deren Eigenschaften — etwa Gleichmäßigkeit oder auch Lebhaftigkeit — über lokale Merkmale quantifizierbar sind.

Ziel der Verarbeitung

Neben Ausschlußgründen für die weitere Verarbeitung, wie etwa Wurmlöcher, gibt es zahlreiche kontextabhängige und quantitative Bedingungen, etwa daß Astlöcher nicht gehäuft auftreten dürfen und die Maserung gleichmäßig sein muß bzw. daß nur eine begrenzte Zahl an Astlöchern zulässig ist. Stehen mehrere Sortierkategorien, wie schlichtes oder lebhaftes Holz zur Verfügung — eventuell dazu in mehreren Güteklassen —, so müssen die Befunde der einfachen Bildverarbeitung in mehrfacher Hinsicht ausgewertet werden. Dabei kann man seriell vorgehen, bis eine passende Sortierkategorie gefunden worden ist, einem hierarchischen Entscheidungsbaum folgen oder auch parallel alle Kategorien testen und je nach Priorität eine auswählen. Diese Aufgaben der komplexeren Bildverarbeitung sind einem nachfolgenden Rechner vorbehalten, der die Objektdetektionsbinärbilder und eventuell Histogramme, die als Ergebnis der einfacheren Bildverarbeitung anfallen, verarbeiten muß. Er muß diese Daten dazu aus Puffern rechtzeitig auslesen, d. h. bevor sie von Daten des nächsten Bildes überschrieben werden, verarbeiten und als Ergebnis der ganzen Bildverarbeitung die Sortierung der Hölzer in Holzkategorien nach DIN–Gesichtspunkten leisten.

1.3 Hardwarelösungen in der Bildverarbeitung

Bildverarbeitungssysteme müssen, wenn aus Kostengründen auf eine Zwischenspeicherung in einem Bildspeicher verzichtet werden soll, zu der Art und Reihenfolge, in der der Bildgeber, meistens Kameras, die einzelnen Pixel liefert, kompatibel sein. Denn nur dann können die Daten direkt ohne Konvertierung oder Speicherung weiterverarbeitet werden.

Technische Grundlagen

Es gibt zwei grundlegend verschiedene Technologien der Bildaufnahme. Bei Kameraröhren wird die Bildinformation seriell abgetastet, bei CCD–Elementen (*charge–coupled–devices*) wird das durch kurzzeitige Belichtung parallel gewonnene Ladungsabbild nach der Belichtung zur Übertragung durch zeilenweise Schiebeoperationen herausgetaktet und somit serialisiert. Aufgrund der hohen Datenmenge kommt eine parallele Übertragung nicht in Frage. Bei identischem Pixeltakt der Kamera und der Verarbeitungseinheit, können diese ohne Bildspeicher direkt gekoppelt und alle Einzelbilder des Bildgebers lediglich mit einer geringen Verzögerung von wenigen Pixeltakten verarbeitet ausgegeben werden.

Videobilder stellen bezüglich ihrer Datenrate hohe Anforderungen an Systeme, wenn sie in

Echtzeit verarbeitet werden sollen. Schon Fernsehkameras liefern 25 mal in der Sekunde eine halbe Million Bildpunkte. Die Bruttopixelrate berücksichtigt auch ungenutzt verstreichende Pixelzyklen, während derer Synchronsignale übertragen werden, so daß kein externer Takt oder Pixeladressen übertragen werden müssen und liegt für Standardvideosignale bei 14.75 Millionen Pixeln/s. Industrielle Bildgeber sind meistens deutlich höherauflösend, aber in der Bildrate nicht auf 25 Bilder pro Sekunde festgelegt. Taktfrequenzen heutiger Mikroprozessoren liegen nur um den Faktor $4-8$ über dem Pixeltakt normaler Kameras. Da es zur Merkmalsvektorberechnung pro Pixel meistens mehrerer Befehle bedarf, scheidet die Möglichkeit der direkten Signalverarbeitung durch einen schnellen Standard– oder auch Signalprozessor aus.

Massiv parallelisierte ICs vs. ASIC

Es bleibt daher nur der Weg über teils massive Parallelisierung von Standardmikroprozessoren oder über anwendungsspezifische ICs, sogenannte ASICs. Der Nachteil der ersten Lösung ist großer Platzbedarf, reduzierte Zuverlässigkeit, Entwicklung brauchbarer parallelisierter Algorithmen mit hohem Synchronisations– und Kommunikationsbedarf. Der Nachteil der zweiten Lösung besteht im ASIC–Entwicklungsaufwand, hohen Fertigungskosten und dem beschränkten, nach dem Entwurf festgelegten Funktionsumfang des ASICs.

Einsatzgebiete für ASICs

Die ersten Schritte der Bildverarbeitung, die Berechnung der klassifizierenden Merkmale (die sich zu Merkmalsvektoren zusammenfassen lassen) laufen meistens sehr uniform ab. Da noch kein Wissen über die Lage eventueller Objekte im Bild vorliegt, müssen alle Pixel zuerst der gleichen Verarbeitungsprozedur unterworfen werden. Viele solcher Schritte lassen sich sehr effizient in Echtzeit mittels ASICs ausführen.

In ihnen werden die Verarbeitungsalgorithmen möglichst parallel ausgeführt, zudem wird auf eine Programmsteuerung und den damit nötigen *Kontrollfluß*overhead verzichtet. Wo jedoch sequentielle Algorithmen eingesetzt werden, laufen eine Reihe von Unterverarbeitungsschritten stattdessen zyklisch oder mikroprogrammiert ab. Ist für jede Operation ein eigenes Rechenwerk realisiert, kann man auf ein Steuerwerk bei der Berechnung völlig verzichten. In so einem *datenfluß*orientierten ASIC sind die Datenpfade im Rechenwerk dem Algorithmus so angepaßt, daß außer dem Takt keine Steuerung mehr benötigt wird. Diese Art der Verarbeitung ist natürlich starr; nur in einigen Registern lassen sich Parameter der Verarbeitung beeinflussen oder zusätzliche Verarbeitungseinheiten in den Datenpfad einschleifen. Es muß also für verschiedene Verarbeitungsalgorithmen verschiedene ASICs geben. Diese Einschränkungen sowie die im Vergleich zu Konsumerprodukten geringe Anzahl von ASIC–basierten Bildverarbeitungssystemen führt zu geringen Fertigungsstückzahlen und somit zu hohen Kosten. Wo komplexere Echzeitbildverarbeitung benötigt wird, sind jedoch ASICs oder Multi–Prozessor–Architekturen unverzichtbar.

Existente Bildverarbeitungssysteme

Aktuelle Standardsysteme weisen deutliche Beschränkungen auf. In der Regel lassen sich Faltungs– und Rangordnungoperationen einer Größenordnung von $8 * 8$ Koeffizienten bzw. Pixeln in Echtzeit durchführen. Die Anzahl von Verarbeitungsmodulen ist auf Bussystemen zudem beschränkt.

Die Bilder werden häufig in Vollbildspeichern zwischengespeichert und pipelineartig bearbeitet, wodurch sich eine pixelgenaue Analyse der Verzögerungen der einzelnen Verarbeitungseinheiten erübrigt. Durch die Bildspeicher werden sie immer auf eine Vollbildverzögerung expandiert. Die Verteilung der Pixelströme ist über das Bussystem meistens frei konfigurierbar. Der Anteil an Komponenten, die nicht unmittelbar an der eigentlichen Bildverarbeitung beteiligt sind — Vollbildspeicher mit Adreßgeneratorlogik, *switching–devices* des Bussystems und ein Hostinterface zur Programmierung — sind hoch. Auch niedrigauflösende Bilder, sehr einfache oder ähnliche Operatoren belegen immer die *worst–case* dimensionierten Ressourcen einer Verarbeitungseinheit. Ebenso erfordern auch die trivialen Rangordnungsfilterungen, Erosion und Dilatation, in Echtzeitanwendungen meistens teure vollwertige Rangordnungsfiltermodule, die es erlauben, jeden Rang auszugeben.

Mit einem solchen System wurden in der Diplomarbeit "Optische Detektion mit Hilfe digitaler Bildverarbeitung" von Olaf Kempendorf [*Kempendorf 1993*] bereits Bildverarbeitungsverfahren zur Segmentierung von Holzfehlstellen erprobt.

Welche Eigenschaften sollte das ASIC haben

Entscheidend beim Entwurf von ASICs ist es, sie im Rahmen dessen, was nicht viel zusätzliche Chipfläche und Entwicklungsaufwand kostet, möglichst universell für eine ganze Klassen von Aufgaben der Bildverarbeitung einsetzbar zu entwickeln, um ihren Hauptnachteil, die hohen Entwicklungskosten, stückzahlmäßig umlegen zu können: *Das* Standardverfahren der Bildverarbeitung ist die Faltung. Mit wenig Mehraufwand sind mit derselben Architektur auch Erosions– und Dilatationsfilter, Spezialfälle der Rangordnungsfilter mit Ausgabe des ersten bzw. letzten Elements der Rangordnung, realisierbar. Entwicklungsrisiken lassen sich zudem dadurch kleinhalten, daß der Entwurf in allen Belangen skalierbar beschrieben wird. Alle wichtigen Parameter, z. B. die unterstützte Bildgröße, die Pixelquantisierung und Faltungsmatrixgröße, lassen sich durch Parameter in der Entwurfsbeschreibung festlegen. So läßt sich die Schaltung nicht nur in kleiner Baugröße testen, sondern es lassen sich auch spätere Änderungswünsche der ASIC–Parameter berücksichtigen, etwa die ASICs in der je nach Integrationsgrad wirtschaftlichsten Größe herzustellen. Um die Entwurfsskalierung nicht am *worst–case* orientieren zu müssen, sollte die verwendete Schaltungsarchitektur kaskadierbar sein. Zur Verarbeitung von Standardkamerasignalen muß das ASIC mit 14.75 MHz Pixeltakt betrieben werden können.

Unter dieser Zielsetzung wurde in der vorliegenden Arbeit die Entwicklung eines ASICs vorgenommen. Dazu wurden zuerst bekannte Bildverarbeitungsoperatoren und Verfahren zur Berechnung von Texturmerkmalen untersucht. Anschließend wurden sie mit einem hierzu entwickelten Bildverarbeitungsprogramm auf der Basis einer Holzstichprobe getestet. Durch graphische Betrachtungen ausgewählter Merkmalsvektoren der verschiedenen Objektklassen im Merkmalsraum wurde belegt, daß über diese Merkmale die Detektion gesuchter Objektklassen, wie Astlöcher und abweichender Maserung, möglich ist. Auf Basis der zur Merkmalsberechnung benötigten Bildverarbeitungsoperatoren wurde schließlich eine Hardware–Architektur entwickelt und diese in der Hardwarebeschreibungssprache VHDL implementiert. Der Entwurf ist über 14 unabhängige Parameter in allen wesentlichen Punkten parametrisierbar. Zur Simulation wird ebenso wie im späteren Betrieb eine Software benötigt, die die ASICs konfiguriert. Nachdem diese entworfen war, wurde der Entwurf in einigen Parametrisierungen und in mehreren Konfigurationen mit synthetischen Testbildern und Realbildern simuliert.

Im folgenden Kapitel wird zuerst auf die technologischen Grundlagen des ASIC–Entwurfs eingegangen.

Kapitel 2

Technologische Grundlagen

2.1 Hardwarebeschreibung

Integrierte Schaltungen können durch ihr äußeres Verhalten, durch ihren inneren strukturellen Aufbau aus Komponenten oder durch die geometrische Elemente, die auf den zu ihrer Fertigung benötigten Belichtungsmasken als Abbild realisiert sind, beschrieben werden. Entsprechend dem Abstraktionsgrad bei der Beschreibung der Schaltung bzw. deren Komponenten findet die Beschreibung auf der Architektur–, algorithmischen, funktionalen, Logik– oder Schaltkreisebene statt. Diesen Zusammenhang veranschaulicht das Gajski– oder Y–Diagramm (s. Abb. 2.1) [*Rosenstiehl, Camposano 1989*]. Jede der drei Achsen — Verhalten, Struktur und Geometrie — repräsentiert eine Sicht. Die konzentrischen Kreise symbolisieren die Abstraktionsebenen der Verhaltens–, Struktur– und Geometriesicht, für die im Diagramm die jeweiligen Beschreibungsprimitiven genannt werden: Strukturelle Beschreibungen setzen sich aus verbundenen Komponenten zusammen, geometrische aus getrennten geometrischen Objekten und Verhaltensbeschreibungen aus Formeln über abstrakte Verhaltensmodelle.

Die verschiedenen Abstraktionsebenen der Struktursicht bilden eine Hierarchie, in der jede Komponente einer Abstraktionsebene aus solchen der nächsttieferen Ebene aufgebaut ist. Das Verhalten der Komponente der oberen Ebene entspricht dabei dem Verhalten des strukturellen Verbundes der Komponenten auf der tieferen Ebene. Während jede strukturelle Beschreibung zusammen mit dem Verhalten der verwendeten Komponenten oder jede Verhaltensbeschreibung für sich eine Schaltung vollständig charakterisiert, muß die Beschreibung auf der Geometriesicht dazu auf der untersten Polygon–Ebene erfolgen. Denn die in den höheren Ebenen durch Abstraktion verlorengegangenen Details lassen sich nicht, wie bei der Strukturbeschreibung, als Verhaltensbeschreibung der Komponenten extrahieren.

Um ein IC schließlich zu fertigen, muß jedoch auf jeden Fall die unterste Geometriesicht eingenommen werden, um den Maskensatz zur Fertigung belichten zu können. Die Struk-

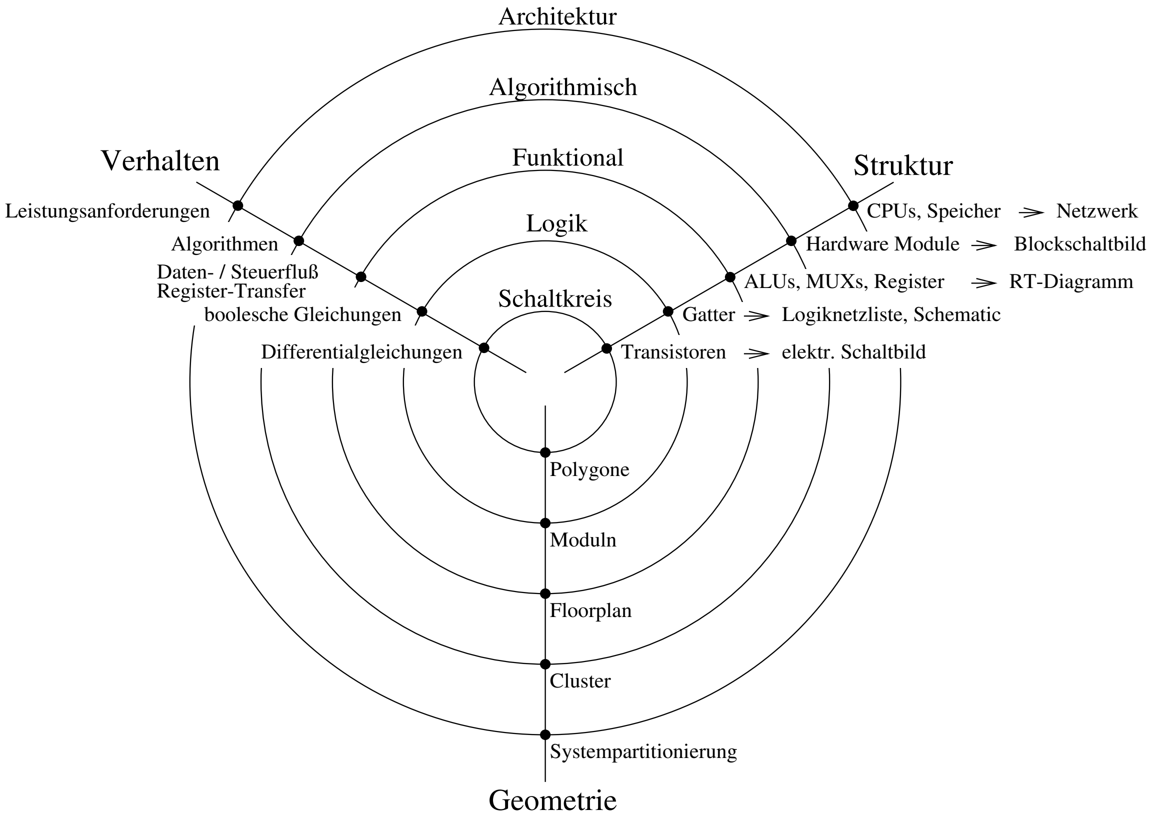


Abbildung 2.1: Gajski-Diagramm

turhierarchie dient dazu, beim Entwurf verschieden stark von den Strukturen der untersten Ebene — den Transistoren — zu abstrahieren, um auch komplexe Entwürfe handhaben zu können. Auf jeder Strukturebene tauschen die Komponenten über Verbindungen Nachrichten aus. Dabei wird auf jeder Ebene sowohl hinsichtlich der Komponenten als auch der Signale hierarchisch weiter abstrahiert. Zum Beispiel werden Spannungen zwischen Transistoren, logische Zustände zwischen Gattern, Integerwerte zwischen ALUs, Instructions zwischen Ausführungseinheiten und schließlich Jobs zwischen vernetzten Prozessoren weitergeleitet. Da dieser Nachrichtenaustausch charakteristisch für jede Ebene ist, werden die Abstraktionsebenen auch Nachrichtenebenen (s. [*Lagemann 1987*, S. 16]) genannt.

2.2 Entwurfsstile

2.2.1 *Full–Custom*

Die direkteste und uneingeschränkteste aber auch komplexeste und fehlerträchtigste Vorgehensweise ist es, direkt auf der untersten Ebene, der Schaltkreisebene, zu entwerfen. Das Verhalten auf der Schaltkreisebene wird durch Differentialgleichungen beschrieben. Für

Digitalschaltungen ist eine genaue Betrachtung der Spannungsverläufe dagegen nicht von Interesse, so daß man hinreichende Bedingungen für das Funktionieren einer Schaltung in Form einfacherer Modelle zur Simulation bevorzugt. Da das Verhalten auf elektrischer Ebene nicht digital ist, ist diese Ebene nicht zur Beschreibung von digitalen Schaltungen geeignet. Neben der Struktur– bietet sich hier jedoch auch die Geometriesicht an, denn auf der untersten Abstraktionsebene ist eine direkte Abbildung zwischen elektrischen Strukturelementen und geometrischen Figuren möglich: Aus dem Layout, dem kompletten Maskensatz, läßt sich automatisch eine Schaltung extrahieren, da die Layoutelemente der verschiedenen Masken, die im Planarprozeß zur Bildung von Transistoren, Verbindungen usw. führen, bekannt sind. Diese Schaltung läßt sich ebenso wieder in ein Layout zurück überführen. Dabei kommt zwar im allgemeinen nicht unbedingt dasselbe Layout heraus, aber ein strukturgleiches mit dem gleichen Verhalten.

Designprozeß Um eine Schaltung aus Geometriesicht zu beschreiben, sind die geometrischen Elemente der Masken für die verschiedenen Diffusionsgebiete, Leiterbahnen und Kontaktierungsstellen mit einem Layouteditor einzugeben. Die Abbildung 2.2 hierzu stammt aus [*Eschermann 1993*]. Die minimalen Abmessungen und Sicherheitsabstände von Strukturelementen sind durch die technologische Beherrschbarkeit des Fertigungsprozesses und die Gesetze der Halbleiterphysik vorgegeben. Sie äußern sich in zahlreichen Designrules, die die möglichen Anordnungen der geometrischen Elemente zur Maskenbeschreibung reglementieren. Der Vorteil liegt darin, jedes Strukturelement, etwa einen Transistor, optimal bezüglich der Lage, Orientierung, und der genauen Positionierung seiner Kontaktstellen beschreiben zu können.

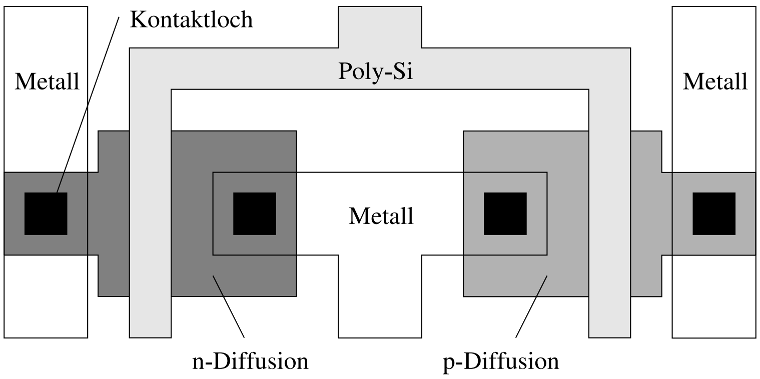


Abbildung 2.2: Layout eines CMOS–Inverters

Bei einem Full–Custom–Entwurf erfolgt aus Struktursicht entweder eine Eingabe von Netzlisten, in denen die Strukturelemente der untersten Abstraktionsebene, Transistoren, instantiiert und verknüpft vorliegen, oder die Strukturelemente werden mit einem graphischen Editor instantiiert und verdrahtet. Ein möglichst dichtes Aneinanderrücken der Komponenten (Kompaktifizieren) läßt sich anschließend automatisch durchführen. Full–Custom–Entwürfe erlauben zwar die größtmögliche Packungsdichte, erfordern aber Entwicklungszeiten in der Größenordnung von Mannjahren. Praktiziert werden sie bei ICs, die mittelfristig in sehr großen Stückzahlen benötigt werden, etwa Mikroprozessoren.

2.2.2 Standardzellen

Eine Abstraktionsebene höher auf der Struktursicht setzen *Standardzellen–Entwürfe* auf ([*Lagemann 1987*, S. 277 ff.]). Die Entwurfsprimitiven sind hier Gatter (Schaltfunktionen von Eingangsvariablen) und Komponenten mit speicherndem Verhalten, Flip–Flops und Latches. Der Umfang solcher Standardzellbibliotheken ist mit den Gattern der bekannten 74xxx IC–Baureihe vergleichbar. Sie stehen als Komponenten in Form von Standardzellen zur Verfügung. Dies sind bereits fertig *Full–Custom* optimierte Entwürfe, deren Einzellayouts im Layout des ICs instantiiert werden. Für jeden Entwurf wird hier nur noch ihre Anzahl, Positionierung und die Verdrahtung individuell berechnet. Auf der Schaltkreisebene wird nur noch vom Anbieter der Standardzellbibliotheken entworfen. Das Ergebnis dieser Arbeit wird von vielen Designs über einen längeren Zeitraum benutzt, so daß die anteiligen Kosten für die Entwicklung der *Full–Custom*–Zellbibliotheken gering sind.

Hardwarebeschreibungssprachen Zur Eingabe gibt es textuelle Hardwarebeschreibungssprachen, die Instantiierungen von Leitungen und Komponenten in Form von Netzlisten erlauben. In den Portlisten der Gatter können zur Verdrahtungsfestlegung zum Beispiel die benötigten Leitungen namentlich aufgezählt werden. Solche Beschreibungen können auch das Produkt von Cross–Compilern anderer Beschreibungsformen (z. B. VHDL s. u.) sein. Alternativ zur textuellen Beschreibung gibt es auch graphische Editoren, die die Standardzellen als *black–box* Komponenten, nur mit ihrem Namen versehen, anzeigen. Die Verbindungsleitungen zwischen ihnen werden durch Linien repräsentiert.

Placement and Routing Zur Fertigung müssen die Standardzellen instantiiert, plaziert und verdrahtet werden. Um sie effizient automatisch plazieren zu können, haben alle Standardzellen die gleiche Bauhöhe (s. Abb. 2.3). Bei der Plazierung werden sie dann dicht an dicht zu Zeilen aneinandergereiht, zwischen denen Platz zur Verdrahtung in Form von Verdrahtungskanälen gelassen wird ([*Kolla et al. 1989*]). Nachdem die Verdrahtung abgeschlossen ist, stellt sie zusammen mit dem inneren Layout aller instantiierten Standardzellen die zur Chipfertigung notwendige Geometrieinformation dar.

Makrozellen Durch die Erweiterung des Standardzellprinzips um komplexe Komponenten kann die Hardwarebeschreibung auf jeder Nachrichtenebene zwischen der Logikebene und der Architekturebene stattfinden. Neben Gattern, ALUs und Modulen, wie seriellen Schnittstellen, werden auch ganze CPUs und Speicher beliebiger Größe in Form skalierbarer Makrozellen angeboten, um nur einige Komponenten der Ebenen zu nennen. Das Layout dieser Makrozellen wird über Layoutgeneratoren und Synthesemechanismen — wiederum in *Full–Custom*–Qualität — erzeugt.

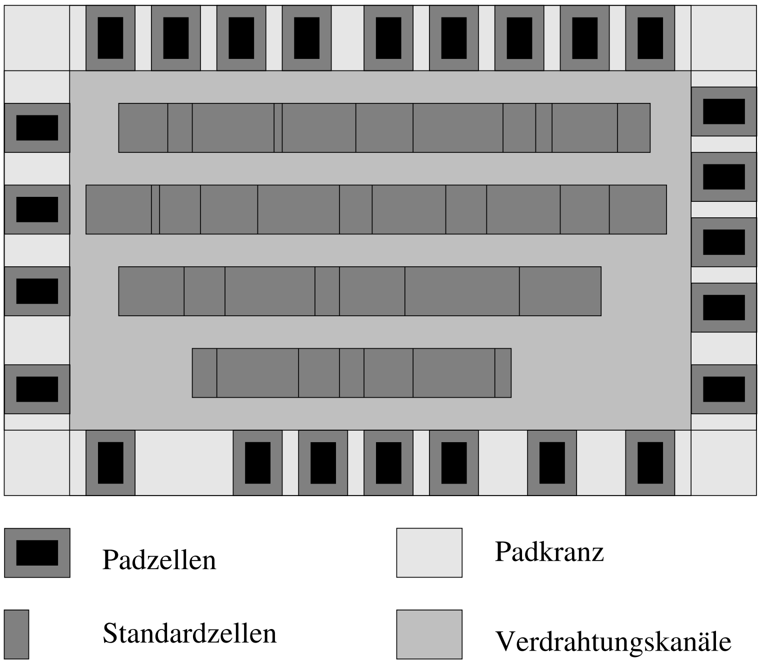


Abbildung 2.3: Schematisches Standardzellenlayouts

2.3 VHDL–Beschreibungen

Moderne Hardwarebeschreibungssprachen, wie das verbreitete VHDL[1], erlauben neben der Strukturbeschreibung als Instantiierung von Komponenten auch die Verhaltensbeschreibungen beliebiger Komponenten auf algorithmischer, funktionaler oder Logikebene. Synthesewerkzeuge ermöglichen die automatische Transformation von Verhaltensbeschreibungen für Komponenten in eine Strukturbeschreibung aus Makro– bzw. Standardzellen.

Verhaltensbeschreibungen sind in VHDL als Prozesse modelliert, die intern aus seriellen Anweisungen bestehen, die permanent zyklisch ausgeführt werden. Innerhalb von Prozessen stehen praktisch dieselben Möglichkeiten offen wie in einer imperativen Programmiersprache. Zudem kann man Prozesse an beliebigen Stellen ihres Verarbeitungszyklus auf bestimmte Ereignisse warten lassen, zum Beispiel auf Takt–Flanken und kann danach abhängig von dem äußeren momentanen Zustand der Schaltung algorithmisch berechnete Nachrichten über Signale ausgeben.

VHDL–Simulator und –Synthese VHDL–basierte Synthesewerkzeuge können zwar alle Prozesse simulieren, aber nur solche auch synthetisieren — also in eine Strukturbeschreibung auf Logikebene überführen (Standardzellenbeschreibung) —, die bestimmte Anweisungen vermeiden oder nur in bestimmten Rahmenbedingungen einsetzen. Der Vor-

[1]
VHDL *V*HSIC *H*ardware *D*escription *L*anguage
VHSIC *V*ery *H*igh *S*peed *I*ntegrated *C*ircuits

teil der nur simulierbaren Anweisungen besteht darin, komplexe verhaltensbeschriebene Spezifikationen in Entwürfen einsetzen zu können (Algorithmendesign), die dann sukzessiv durch synthetisierbaren Code ersetzt werden. Die am Anfang stehende Spezifikation eines ICs oder einer Komponente kann somit problemlos mit dem Endresultat und allen Zwischenstadien verglichen werden, da sich alle mit denselben Werkzeugen — auch gemischt — simulieren lassen. Die synthesefähigen Verhaltensbeschreibungen sind hinsichtlich ihres Abstraktionsgrades auf der Ebene der funktionalen Beschreibungen (Register–Transfer Ebene) einzuordnen. Sie erlauben die Beschreibung endlicher Automaten sowie Komponentenbeschreibungen von ALUs, Multiplexern, Registern usw. unabhängig von der zur Verfügung stehenden Zellbibliothek.

Das Endresultat läßt sich dann mit Hilfe von Syntheseprogrammen auf die gewünschte Zellbibliothek abbilden und kann in Form von strukturellen Hardwarebeschreibungssprachen auf Gatterebene ausgegeben werden. Es ist damit äquivalent zu einem Standardzellenentwurf und kann mit entsprechenden Werkzeugen weiterverarbeitet werden.

Im folgenden Kapitel wird zuerst auf die eingesetzten Klassen von Bildverarbeitungsoperatoren eingegangen. Darauffolgend werden klassifizierende Merkmale zum Schablonenvergleich und solche zur Orientierungsbestimmung von Texturen vorgestellt. Eine Möglichkeit, Merkmale von Bildern in verschiedenen Skalierungen zu berechnen, bietet sich, wenn diese als Bildpyramiden vorliegen, die nun vorgestellt werden. Einige Auswertungsmöglichkeiten aufgrund solcher klassifizierenden Merkmale werden dann am Schluß des nächsten Kapitels aufgezeigt.

Kapitel 3

Theorie der Bildverarbeitung

3.1 Bildanalytische Verfahren

Im folgenden wird von Bildverarbeitungsschritten gesprochen, die ein Quellbild in ein Zielbild überführen. Im allgemeinen erfolgen mehrere solcher Verarbeitungsschritte nacheinander. Am Anfang steht das Originalbild, am Ende ein klassifizierendes Merkmal. Die Bilder, die eventuell dazwischen entstehen, enthalten nur Zwischenergebisse. Da dies für den Operator unwesentlich ist — er kann eine Operation ebensogut auf einem Originalbild wie auf einem Zwischenergebnis vorangegangener Operationen berechnen — und hier Operatoren klassifiziert werden sollen, wird hier nur von Quell– und Zielbildern die Rede sein. Aus dem Quellbild kann aus Sicht des Operators dabei nur gelesen, in das Zielbild können nur Ergebnisse geschrieben werden. Jeder Pixelwert des Zielbildes ergibt sich rein funktional aus Pixelwerten des Quellbildes. Wenn nicht anders vermerkt, handelt es sich bei Pixeln um die des Quellbildes, da solchen zur Klassifizierung der Operatoren unsere Aufmerksamkeit gilt. Es ist allen Operatoren gleich, einen einzigen Ergebniswert als Pixel kodiert zu liefern.

3.1.1 Punktoperationen

Die einfachste Klasse der Bildverarbeitung bilden die Punktoperationen. Das Ergebnis des Punktoperators ist dabei im allgemeinen eine Funktion des Wertes des Pixels und seiner Position im Bild. Ein Spezialfall sind die homogenen Punktoperatoren; bei ihnen hängt die Funktion nicht von der Pixelposition, sondern nur vom Wert des Pixel selbst ab. Solche Funktionen lassen sich als Funktionstabellen realisieren, aus denen das Ergebnis einfach abgelesen werden kann, sogenannten Look–Up–Tables (LUTs). Sie lassen sich mit Hilfe von schnellen ROMs in Echtzeit realisieren.

Zur Berechnung von klassifizierenden Merkmalen, die visuelle Merkmale ausmessen oder zur Detektion von Objekten dienen sollen, die nicht einem isolierten Pixel des Originalbildes zuzuordnen sind, wie Textureigenschaften oder Kanten, eignet sich diese Klasse von

Operatoren jedoch nicht.

3.1.2 Faltung

Eine Erweiterung des Konzeptes besteht darin, nicht nur einen Pixel $P_{x,y}$ als Funktionsargument heranzuziehen, sondern auch die im Eingabebild benachbarten, bis zu r Pixeln entfernten (Schachbrettmetrik) Pixel $P_{x-i,y-j} \mid -r <= i,j <= r$ als weitere Argumente zuzulassen. Bilden solche "lokal" genannten Operatoren aus diesen Pixeln eine Linearkombination, werden sie Faltungsoperatoren genannt. Normalerweise sind die Koeffizienten $C_{i,j}$ unabhängig von der Bildposition konstant. Solche Faltungsoperatoren heißen homogen.

$$F_{x,y} = \sum_{j=-r}^{r} \sum_{i=-r}^{r} C_{i,j} P_{x-i,y-j} \tag{3.1}$$

Trotz all dieser wesentlichen Einschränkungen stellen sie eine funktional sehr mächtige Operatorenklasse dar. Das ursprünglich (Punktoperationen) einzige Pixel $P_{x,y}$, das sich von den benachbarten Pixeln bei der Faltung nur noch dadurch unterscheidet, daß das berechnete Ergebnis seiner Position als Funktionswert zugeordnet wird, heißt "Zentralpixel" (s. Abb. 3.1).

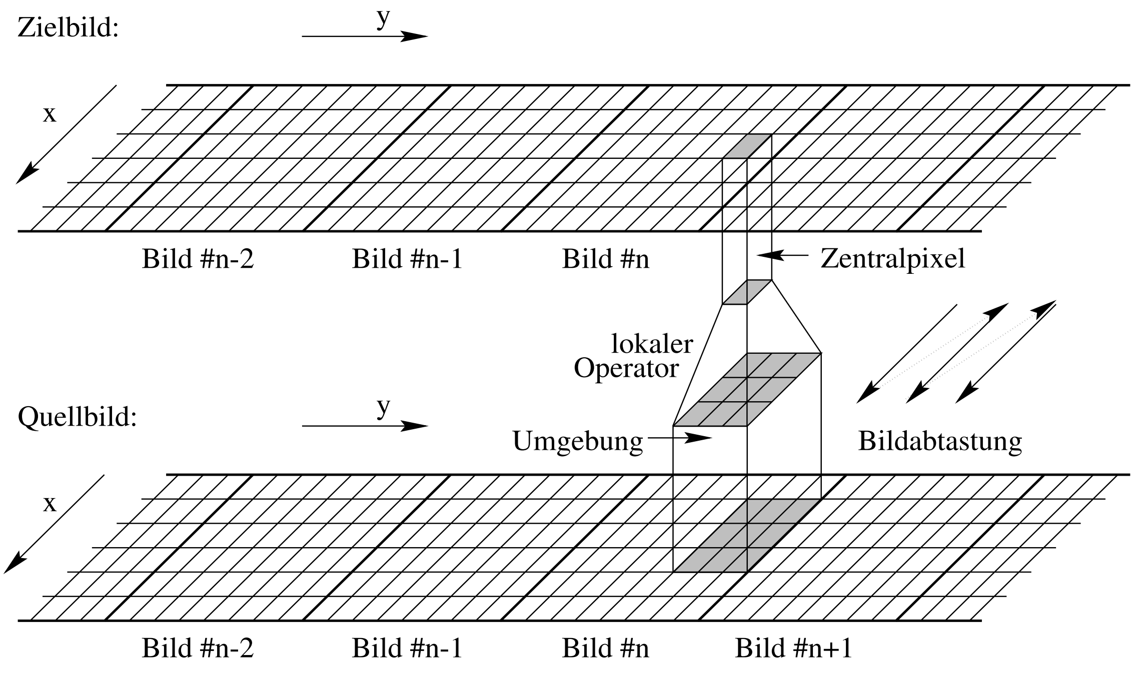


Abbildung 3.1: lokaler Operator

Die Umgebungspixel werden, ebenso wie der Zentralpixel, mit Koeffizienten gewichtet, die von ihrer relativen Position zum Zentralpixel abhängen. Diese Koeffizienten sind aus Sicht des jeweiligen Zentralpixels ortsfest und lassen sich in einer sogenannten Faltungsmatrix zusammenfassen. Um das Faltungsergebnis der Umgebung einer Pixelposition zu erhalten, wird diese Faltungsmatrix so verschoben, daß der Koeffizient des Zentralpixels mit dieser Pixelposition zur Deckung kommt. Nun werden alle Pixel mit den durch die Faltungsmatrix angegebenen Koeffizienten multipliziert und dann zum Ergebnis aufsummiert. Die Koeffizienten können dabei auch negativ sein.

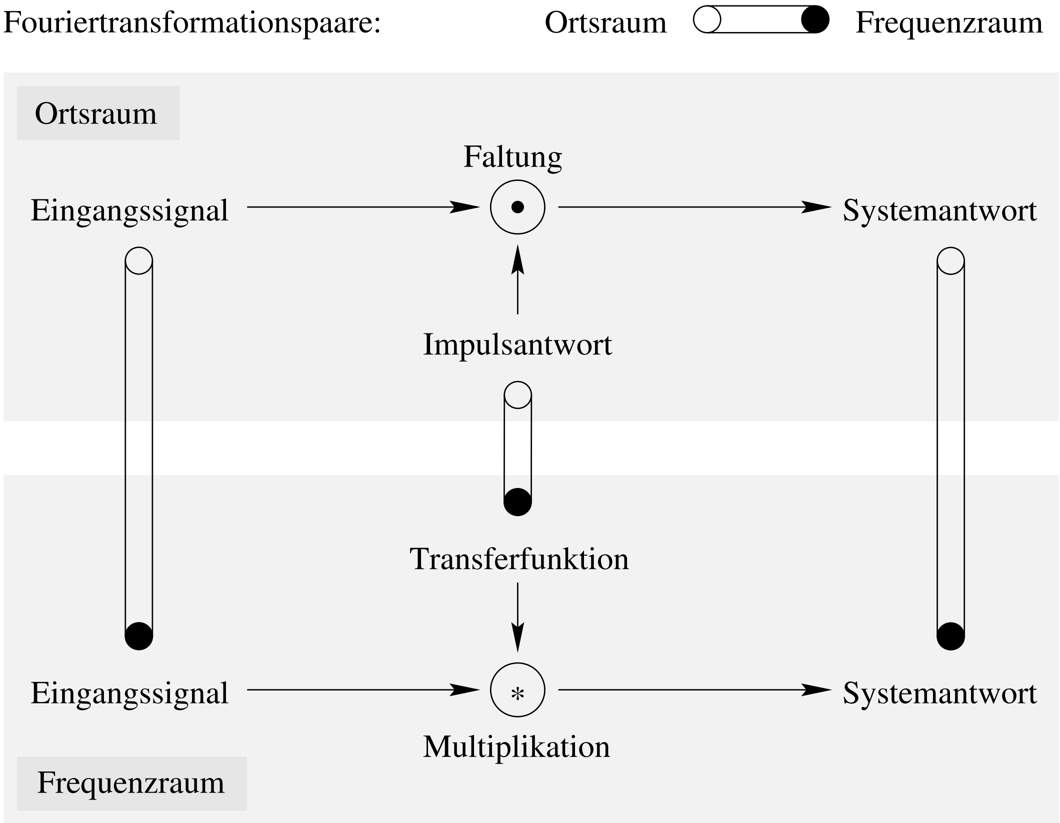


Abbildung 3.2: Systemtheorie linearer Systeme

Bilder lassen sich nicht nur im Ortsraum durch Angabe des Wertes jedes Ortspunktes beschreiben, sondern auch im Fourier– oder Ortsfrequenzraum, wo jeder Bildfrequenz die Intensität ihres Auftretens im ganzen Bild zugeordnet wird. Bilder lassen sich durch die lineare Fouriertransformation jeweils von einem in den anderen Raum überführen (z. B. S.476, [*Bronstein 1962*]). Eine Eigenschaft der Fouriertransformation ist es, daß Faltungen (mit ortsabhängigen Funktionen) im Ortsbereich Multiplikationen (mit frequenzabhängigen Funktionen) im Frequenzbereich entsprechen und umgekehrt (s. Abb. 3.2). Damit läßt sich die frequenzabhängige Wirkung von Faltungsfiltern übersichtlich als sogenannte Transferfunktion im Ortsfrequenzraum in dreidimensionalen Graphen darstellen. Als abhängige Variablen treten hier die Frequenzen in X– und in Y–Richtung auf, als Funktionswerte entweder der Real– und Imaginäranteil der komplexen Frequenzantwort oder der Betrag der Frequenzantwort und seine Phasenverschiebung.

Die Transferfunktion einer Faltungsmatrix erhält man also einfach als Fouriertransformation der durch die Faltungmatrix vorgegebenen Impulsantwort in den Ortsfrequenzraum. Dazu werden die diskreten Koeffizienten der Matrix als entsprechend gewichtete Dirac–Stöße einer kontinuierlichen Faltungsfunktion aufgefaßt, die das Bild nur an diskreten Werten abtasten. Deren Fouriertransformierte sind komplexe Wellenfunktionen der Form $\exp(-i\,2\pi\,f\,t)$. Die Transferfunktion ergibt sich nun unter Ausnutzung der Linearität der Fouriertransformation als Linearkombination dieser Terme.

Binomialoperatoren

Zur Frequenzfilterung sind Filter mit hoher Steilheit, geringem Überschwingen im Frequenzraum und möglichst kleiner Matrixgröße im Ortsraum gesucht. Die dritte Forderung ergibt sich aus dem Rechenbedarf für solche Filter. Eine Funktion, die ohne Überschwingen schnell abklingt, ist die Normalverteilung, die durch Fouriertransformation in sich selbst überführt wird und als "Gaußsche Glockenkurve" bekannt ist. Im Frequenzraum sind nur positive Frequenzen von physikalischer Bedeutung, so daß die Normalverteilung als Transferfunktion einem Tiefpaßfilter entspricht (s. Abb. 3.3).

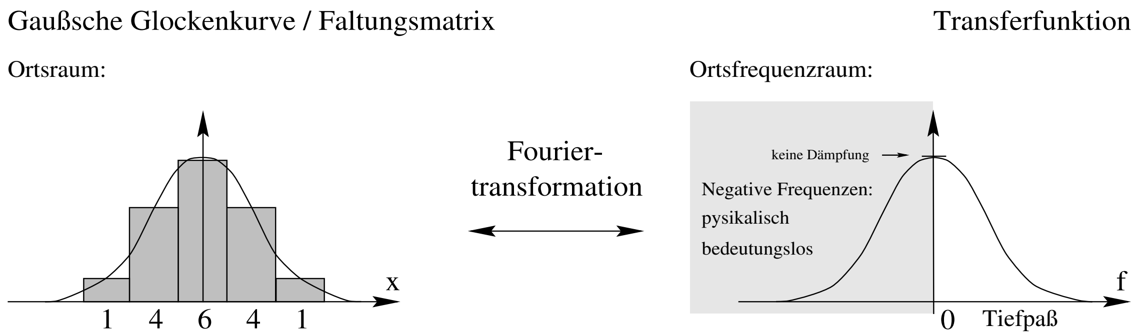


Abbildung 3.3: Tiefpaßfilterung mit Binomialfiltern

Für den diskreten Fall konvergieren die Binomialkoeffizienten mit wachsender Koeffizientenzahl, also der "Ordnung" der Binomialkoeffizienten, schnell gegen die Normalverteilung ([*Jähne 1991*, S. 78]). Sie bilden die Klasse der Binomialmatrizen. Eine Binomialmatrix nullter Ordnung ist die Identitätsmatrix (1). Man erhält die höheren Ordnungen, jeweils durch Faltung der vorhergehenden mit der Binomialmatrix erster Ordnung: $(1,1)\bullet$. (Die Binomialmatrix tritt dabei sowohl als Operator als auch als zu faltetendes Objekt auf. Im letzteren Fall stelle man sie sich dazu eingebettet in einem Bild vor, das ansonsten nur Pixel mit dem Wert 0 besitzt.) Die eindimensionale Binomialmatrix vierter Ordnung lautet demnach: $(1,1)\bullet(1,1)\bullet(1,1)\bullet(1,1)\bullet(1) = (1,4,6,4,1)$. Eine zweidimensionale Gaußfilterung läßt sich durch zwei eindimensionale in X– und Y–Richtung faltende Binomialmatrizen, die sequentiell ausgeführt werden, realisieren, beispielsweise in folgender Form:

$$\frac{1}{4}\begin{pmatrix}1\\2\\1\end{pmatrix} * \frac{1}{4}\begin{pmatrix}1 & 2 & 1\end{pmatrix} = \frac{1}{16}\begin{pmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{pmatrix} \tag{3.2}$$

Operatoren, die diese Eigenschaft besitzen, heißen separabel. Während eine Filterung in X– und Y–Richtung mit eindimensionalen Faltungsmatrizen bei n Koeffizienten je Pixel $2 * n$ Multiplikationen und $2 * (n-1)$ Additionen benötigt, werden ohne Ausnutzung der Separierbarkeit n^2 Multiplikationen und $n^2 - 1$ Additionen benötigt.

Differentialoperatoren

Durch Faltungen lassen sich auch Differentialoperatoren realisieren. Man kann diese auch ohne Umwege über den Frequenzraum einführen. Eine übliche Darstellung des Differentialquotienten ([*Jähne 1991*, S. 101]) ist:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{3.3}$$

Die Nullfolge h kann hier im Diskreten einen minimalen Abstand von einem Pixel haben. Statt der Grenzwertbildung bleibt die Möglichkeit, sich für $h = 1$ oder $h = -1$, den rechts– oder linksseitigen Grenzwert, zu entscheiden. Die Differenzenquotienten lauten dann $f(x+1) - f(x)$ bzw. $f(x) - f(x-1)$, die zugehörigen Faltungsmatrizen also $(0, -1, 1)$ bzw. $(-1, 1, 0)$, wobei der Zentralpixel jeweils in der Mitte liegt. Da die Matrixkoeffizienten asymmetrisch zum Zentralpixel besetzt sind, handelt es sich hier um den asymmetrischen Differentialoperator $D = (-1, 1)$. Dabei stellt sich nicht nur bei dieser kompakteren Schreibweise die Frage, welcher Koeffizient der des Zentralpixels ist. Man kann nämlich über die Eigenschaften eines Differentialquotienten im Frequenzraum zeigen ([*Jähne 1991*, S. 102]), daß mit dieser Matrix Differentialquotienten für Zwischengitterplätze berechnet werden. Die Position des Zielbildes, für die das Ergebnis den Differentialquotienten repräsentiert, liegt also jeweils in der Mitte zwischen den beiden Koeffizienten -1 und 1 der Matrix.

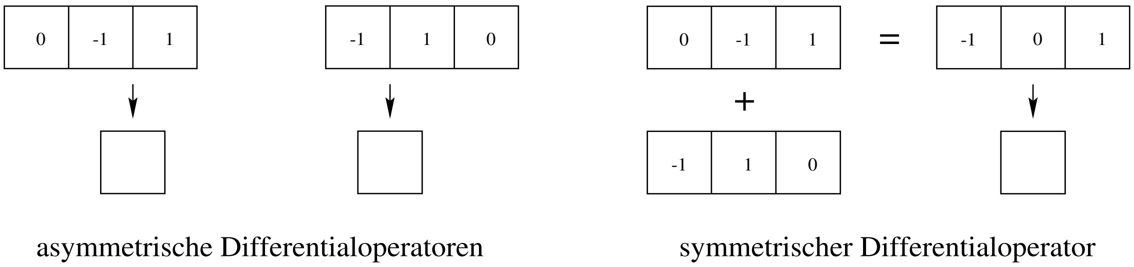


Abbildung 3.4: Zentralpixelposition von Differentialmatrizen

Aus der asymmetrischen Faltungsmatrix $(1, -1)$ kann eine symmetrische durch Faltung mit der Binomialmatrix $(1, 1)$ erzeugt werden (s. Abb. 3.4). Dies entspricht genau der Aufsummierung der beiden erstgenannten Matrizen, also der Differentialkoeffizienten der Positionen $1/2$ und $-1/2$ Pixel bezüglich des Zentralpixels:

$$(0, -1, 1) + (-1, 1, 0) = (1, 0, -1) = (1, -1) \bullet (1, 1) \tag{3.4}$$

Der Nachteil ist offensichtlich: Eine Binomialmatrix ist ein Tiefpaß, der eine Differentialbildung bei hohen Frequenzen verfälscht. Sowohl symmetrische als auch asymmetrische Faltungsoperatoren lassen sich jedoch noch optimieren. Dazu wird man versuchen, möglichst genau den Eigenwert k des Differentialoperators (3.5) im Frequenzraum zu approximieren:

$$\frac{d}{dx}\, e^{kx} = k\, e^{kx} \tag{3.5}$$

Wählt man als allgemeinen Ansatz eine ungerade Faltungsmatrix mit beliebigen Koeffizienten, berechnet von dieser, wie oben skizziert, die Transferfunktion, entwickelt sie in eine Taylorreihe und stellt ein Gleichungssystem auf, so daß sich möglichst viele Koeffizienten für die höheren Potenzen in k wegheben, so lassen sich optimierte Differentialmatrizen berechnen. Das Verfahren ist ausführlich in [*Jähne 1991*, S. 116 – 118] beschrieben.

3.1.3 Schablonenvergleich

Faltungsoperatoren lassen sich auch für näherungsweise Schablonenvergleiche einsetzen ([*Dreschler–Fischer 1994*, Teil II, S. 3: Schablonenvergleich (2)]). Dabei ist die Schablone eine Bildstruktur, deren Vorkommen im Bild durch ein Merkmal detektiert werden soll. Um nicht auch unbedeutend kleine Variationen mitzudetektieren, ist es notwendig, ein Ähnlichkeitsmaß einzuführen, das dann als klassifizierendes Merkmal zur Objektdetektion dienen kann. Ein bekanntes Ähnlichkeitsmaß stellt der euklidische Abstand dar — die Wurzel aus der Summe der quadrierten Differenzen zwischen Schablone $S_{i,j}$ und Operatorumgebung $P_{i,j}$ (im Abschnitt 3.4 werden weitere Ähnlichkeitsmaße eingeführt).

$$\begin{aligned} d_{euklidisch}(x, y) &= \sum_{i,j=-r}^{r} (S_{i,j} - P_{x-i,y-j})^2 && (3.6) \\ &= \sum_{i,j=-r}^{r} (S_{i,j}^2 - 2 * S_{i,j} P_{x-i,y-j} + P_{x-i,y-j}^2) && (3.7) \end{aligned}$$

Mittels einer Faltung der Schablone mit dem Bild läßt sich folgendes Merkmal berechnen, das den gemischten Produkten beim Quadrieren der Differenz bei der Berechnung des euklidischen Abstands entspricht:

$$d_{gefaltet}(x,y) = \sum_{i,j=-r}^{r} S_{i,j} P_{x-i,y-j} \tag{3.8}$$

Die Differenz von (3.7) und zweimal (3.8) entspricht dabei dem Fehler, der bei der Näherung des Schablonenvergleichs über den doppelten Wert der Faltungsmatrix auftritt:

$$d_{euklidisch}(x,y) - 2 * d_{gefaltet}x, y = \sum_{i,j=-r}^{r} \left(S_{i,j}^2 + P_{x-i,y-j}^2 \right) \tag{3.9}$$

Der erste Term ist dabei konstant und bekannt, da er nur von der Faltungsmatrix, also der Schablone, abhängig ist. Er kann konstant abgezogen werden, stört aber auch ansonsten die Suche maximaler Ähnlichkeit nicht. Der zweite Term ist problematischer; er ergibt sich aus der Summe der Quadrate der Umgebung des Operators. Unter der Annahme von mittelwertfreien Bildern, die z. B. bei der Laplacepyramide vorliegen, kann immerhin noch eine Aussage über das Quadrat der Summe der Umgebung des Operators getroffen werden — es ist null. Aus der Statistik ist außerdem zur Berechnung der Varianz einer Zufallsvariablen X bekannt:

$$\sigma^2(X) = Mittelwert(X^2) - Mittelwert^2(X) \tag{3.10}$$

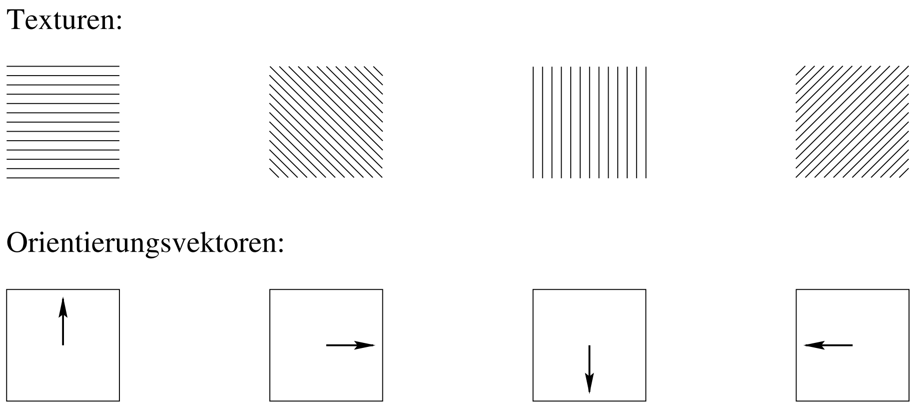
So entspricht der Störterm auf mittelwertfreien Bildern der Varianz der Umgebung des Operators. Ist die Varianz über das Bild überall annähernd gleich, kann eine Faltung einen Schablonenvergleich bezüglich eines euklidischen Ähnlichkeitsmaßes gut approximieren. Bei nicht–mittelwertfreien Bildern kann die Summe über die Quadrate der Umgebungspixel mit einer lokalen Energie der Bildumgebung o. ä. umschrieben werden (vgl. $E = U^2 \, R \, t$), die zum Schablonenvergleich mittels Faltungen konstant sein muß.

3.1.4 Orientierungsbestimmung

Dem Buch [*Jähne 1991*, S. 133 – 137] entnommen ist ein Verfahren zur Orientierungsbestimmung von Texturen. Es liefert als Ergebnis einen Orientierungsvektor ($\vec{O}$). Zur Berechnung werden nur sehr einfache Operatoren benötigt: Differentialoperatoren in X– (D_x) und Y–Richtung (D_y), zwei zweidimensionale Binomialoperatoren (B) und drei Multiplikationen als Punktoperationen. Die Differentialoperatoren in X– und Y–Richtung im Inneren der Gleichung (3.11) detektieren mit sich selbst multipliziert Helligkeitssprünge im Bild in ihrer Richtung und gemischt je nach Vorzeichen auch in den beiden Diagonalen. Die erste Komponente des Orientierungsvektors ergibt sich aus der tiefpaßgefilterten Differenz der Ableitungsquadrate in X– und Y–Richtung, der zweite aus dem tiefpaßgefilterten Quadrat ihrer gemischten Ableitung. Die Tiefpaßfilterung ist so auszulegen, daß Frequenzen ab der halben Wellenlänge der größten Textur–Wellenlänge, deren Orientierung noch bestimmt werden soll, ausgefiltert werden. Da diese Filterung auch die Ortsauflösung des Orientierungsvektors einschränkt, sollte ihre Grenzfrequenz nicht tiefer als nötig gewählt werden. Binomialfilter finden hier als Tiefpaßfilter Anwendung:

$$\vec{O} = \begin{pmatrix} B(D_y \bullet D_y - D_x \bullet D_x) \\ 2B(D_x \bullet D_y) \end{pmatrix} \tag{3.11}$$

Eine kurze Betrachtung spezieller Orientierungen soll hier das Verfahren plausibel machen: Für horizontale Strukturen ist nur das Y–Differential verschieden von null, die erste Vektorkomponente ist daher positiv, die zweite null. Für vertikale Strukturen dagegen ist nur das X–Differential verschieden von null, die erste Vektorkomponente somit negativ, die zweite ebenfalls null. Für die Diagonalen ist abhängig von ihrer Richtung das Produkt immer positiv oder negativ: Je nach ansteigender oder nachlassender Helligkeit an Texturkanten sind die Vorzeichen beider Differentiale vertauscht. Das Vorzeichen ihres Produktes hingegen hängt nur von der Richtung der Diagonalen ab. Bei der Berechnung der ersten Komponente löschen sich die betragsmäßig gleichen X– und Y–Differentiale bei der Differenzbildung aus, sie ist bei Diagonalen gleich null. Im folgenden sei die Stellung des Orientierungsvektors für die besprochenen Texturorientierungen dargestellt:



Der Orientierungsvektor einer Textur dreht sich also mit doppelter Geschwindigkeit, wie diese selbst; nach einer Drehung der Textur um 180° nimmt er wieder dieselbe Richtung ein. Wie Jähne über Betrachtungen im Frequenzraum nachweist, besteht zwischen dem Orientierungsvektor und dem Winkel der Texturorientierung ϕ folgender Zusammenhang:

$$\tan(2\phi) = \frac{B(D_y \bullet D_y - D_x \bullet D_x)}{2B(D_x \bullet D_y)} \tag{3.12}$$

Neben der Richtung kann man auch eine Aussage über die Intensität der Texturierung machen, die im allgemeinen nicht dem Betrag des Orientierungsvektors entspricht, der für isotrop (ohne Vorzugsrichtung) texturierte Strukturen ebenfalls $\vec{0}$ wird. Zur Berechnung dieses Merkmals werden die quadrierten X– und Y–Ableitungen aufsummiert und ebenfalls tiefpaßgefiltert:

$$B(D_y \bullet D_y + D_x \bullet D_x) \tag{3.13}$$

Dieses Merkmal ist neben dem Orientierungsvektor für Texturen dann informativ, wenn nicht nur ideal orientierte Texturierung auftritt. Ist dies jedoch der Fall, läßt sich die Texturintensität auch am Betrag des Orientierungsvektors ablesen.

3.1.5 Rangordnungsfilter

Rangordnungsoperatoren entsprechen keiner arithmetischen Funktion auf den Pixeln der Umgebung des Zentralpixels, sondern selektieren das p.–größte Pixel und geben es als Funktionswert zurück. Für den allgemeinen Fall müssen dazu die Pixel ihrer Wertigkeit oder Rangordnung nach sortiert werden. Eine Ausnahme liegt vor, wenn nach dem ersten oder letzten Wert der Rangordnung gesucht wird. Hier genügen $n-1$ Vergleiche zwischen den n Pixeln der Umgebung, wobei immer der größere bzw. kleinere Pixel weiterverglichen und am Ende ausgegeben werden. Diese Spezialfälle von Rangordnungsfiltern heißen Dilatations– und Erosionsfilter, die allgemeine Bezeichnung ist Rangordnungsfilter p–ter Ordnung.

Eine Eigenschaft der Rangordnungsfilter besteht darin, daß sie angewandt auf Bildkanten, die von Flächen konstanter Helligkeit umgeben sind, oder auf Bilder mit gleichmäßig steigenden (also auch konstanten) Grauwerten zu keiner Signaländerung durch die Filterung führen. Dies ist einfach zu zeigen, denn solche Bildinhalte bilden von vornherein eine geordnete Rangfolge, so daß es in Abhängigkeit von der gewählten Ordnung höchstens zu einer Verschiebung kleiner dem Abstand des Zentralpixels zum Rand seiner Umgebung kommt. Sollten die Pixelwerte in umgekehrter Rangfolge angeordnet vorliegen, entspricht dies einem gewählten Rang $p' = n - p$ auf dieser umgekehrten Rangordnung.

Die Funktionsweise von Rangordnungsfiltern mit $p \approx n/2$ beruht darauf, daß statistisch stark streuende Werte mit hoher Wahrscheinlichkeit nur vereinzelt auftreten. Damit werden sie am Anfang oder Ende der Rangfolge eingeordnet und beeinflussen den mittleren Bereich, aus dem selektiert wird, nicht. So können solche Filter zum wirksamen Ausfiltern von statistischem Rauschen eingesetzt werden. In diesem Zusammenhang ist es auch von Vorteil, daß das Ausgabepixel ausgewählt und nicht berechnet werden müssen. So können auch Pixel mit mehrdimensionalen Werten anhand eines eindimensionalen Merkmals sortiert und dann wieder mehrdimensional ausgegeben werden (z. B. RGB–Pixel nach ihrem Luminanzwert gegen Bildrauschen rangordungsgefiltert oder allgemeiner: Filterung von Mehrkanalspektralaufnahmen).

Das andere Extrem ($p = 1$ oder $p = n$) sind Dilatations– und Erosionsfilter. Sie können dazu dienen, kontrastmäßig abgehobene helle Bildstrukturen, die kleiner der Umgebung des Filters sind, verschwinden zu lassen und größere zu verkleinern (Erosion) bzw. in jedem Fall zu vergrößern (Dilatation) (für dunkle Bildstrukturen gilt entsprechend Umgekehrtes). Eine sequentielle Anwendung von Erosions– und Dilatationsfiltern führt also dazu, daß kleinere Strukturen gänzlich aus dem Bild verschwinden, größere dagegen unverändert bleiben. Diese Operationsfolge wird daher auch morphologisches Schließen genannt. Ein Vergleich des Originalbildes mit einem morphologisch geschlossenen Bild offenbart damit alle Strukturen, die kleiner als die Umgebung des Filters sind.

Die mathematische Handhabung von Rangordnungsfiltern innerhalb der Signaltheorie ist nicht möglich. Auch auf statistischer Basis sind Aussagen über Rangordnungsfiltern nicht trivial.

3.2 Statistische Verfahren

Neben pixelorientierten Verfahren kommen zur Klassifikation auch statistische Verfahren in Betracht. Dabei gibt es die Möglichkeit, alle Pixel unabhängig von ihrer Position im Bild zu bewerten oder in Abhängigkeit ihrer Nachbarn. Ein allgemeines Verfahren zur Vorverarbeitung von Bildern besteht darin, zuerst ihr Histogramm, d. h. die Anzahl der Pixel jedes möglichen Wertes, zu berechnen. Das kann hardwareunterstützt ablaufen, indem jedes Pixel über seinen Wert eine Speicherzelle im Histogrammspeicher adressiert. Sie wird ausgelesen, ihr Wert inkrementiert und in sie zurückgeschrieben. Ein solches ASIC wurde bereits entwickelt [*Dwersteg 1995*]. Da dies keine zeitaufwendige Aktion ist, kommen prinzipiell auch sehr schnelle Signalprozessoren, die einen Tabelleneintrag mittels indizierter Adressierung inkrementieren, in Betracht. Das Auslesen und Neuinitialisieren der Tabelle kann in der Austastlücke oder während einer ungenutzten Bildzeile geschehen.

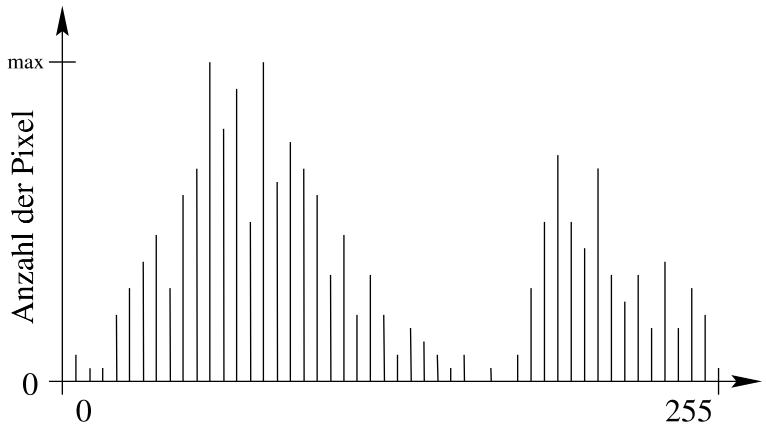


Abbildung 3.5: Histogramm

Das Ergebnis läßt sich bei einfachgelagerten Fällen direkt zur Segmentierung über Schwellenwerte nutzen: Sind die Helligkeitsverteilungen verschiedener Objektklassen und des Hintergrundes im Histogramm disjunkt, können die Objekte sicher über einen Schwellenwert detektiert werden, der zwischen beiden Verteilungen liegt. Im allgemeinen lassen sich aus Histogrammen Mittelwerte, Varianzen oder beliebig höhere statistische Momente berechnen, die ein Merkmale des Gesamtbildes darstellen. Damit Bildstrukturen bei großen Bildern nicht in statistischen Schwankungen (Rauschen) der Histogramme untergehen, kann man Einzelhistogramme über Teilbilder — sog. Bildkacheln — berechnen. Das Gesamthistogramm steht dabei auch als Summe der Einzelhistogramme zur Verfügung.

Co–Occurenzmatrizen verfeinern Histogramme dahingehend, daß nicht nur der Wert eines Pixels, sondern auch der seines Nachbarn berücksichtigt wird. Ein Eintrag an der Position (x, y) einer Co–Occurenzmatrix sagt aus, wieviele Pixel den Wert x und einen Nachbarn des Wertes y hatten. Werden alle Zeilen der Matrix aufsummiert, ergibt sich wieder das Histogramm. Zur Histogrammberechnung wird nun anstatt eines Pixels, der Wert des Pixels und seines Nachbarn konkateniert benutzt. Es muß dann eine Wertetabelle mit doppelt so breiten Indizes geführt werden. Alternativ kann stattdessen auch die Quantisierung der Pixel halbiert werden. Statt sich für eine Nachbarschaft in horizontaler oder vertikaler Richtung zu entscheiden, können beide Matrizen simultan berechnet werden oder man verallgemeinert das Konzept auf einen zweiten Nachbarschaftspixel und erhält so dreidimensionale Matrizen. Ein Zusammenhang beider Alternativen besteht darin, daß eine Summation über je einen Index der Nachbarschaftspixel der dreidimensionalen Matrix zu den erstgenannten beiden simultan berechneten zweidimensionalen Matrizen führt. Bei mehreren Nachbarpixeln voller Quantisierung wird die Datenmenge eines Tabelleninhalts jedoch mit der des Bildes vergleichbar, so daß das Auslesen der schwach besetzten Matrix problematisch wird.

Der Vorteil von Co–Occurenzmatrizen besteht darin, daß homogene Bilder, die in ihrer Helligkeit lokal nicht stark variieren, zu einer Füllung der bezüglich der Indizes winkelhalbierenden Diagonalen der Matrix beitragen, während Bilder mit lokal stark schwankenden Helligkeiten zu einer punktsymmetrische Füllung der Matrix ohne Vorzugsrichtung führen. Manche Klassifikationsaufgaben lassen sich über dieses Merkmal des Gesamtbildes erstaunlich gut lösen.

3.3 Bildpyramiden

Die Größe der benötigten Umgebung zur Merkmalsberechnung richtet sich danach, über einen wie großen Bildbereich das berechnete Merkmal eine Aussage machen soll. Um unabhängig von der Größe eines Objektes ein Merkmal relativ zu seiner Größe, also skalierungsunabhängig, berechnen zu können, muß es zuerst auf eine konstante Größe gebracht werden. Dazu kommt die kleinste Umgebung in Frage, in der sich noch alle Objekteigenschaften von Interesse auflösen lassen. Da diese Umgebung vollständig im Einzugsbereich der anschließenden Operatoren zur Merkmalsberechnung liegen muß, schlägt sich ihre Größe direkt im Hardwareressourcenbedarf nieder.

Analog zu dem Vorgehen, zu allen Pixeln Merkmalsvektoren zu berechnen, um an diesen Stellen auftretende Objekte zu ermitteln, kann man anstatt Objekte an unbekannten Stellen auch Objekte unbekannter Größe suchen. Man muß nur Merkmalsvektoren zu dem in alle möglichen Größen skalierten Originalbild detektieren. Kombiniert man beide Verfahren, also detektiert man ein bestimmtes Objekt, indem man klassifizierende Merkmale zu jedem Ort und in jeder Größenskalierung des Bildes berechnet, erhält man als Zusatzinformation zum Ort auch die Größenskalierung unter der das Objekt vom Detektor erkannt wurde — und damit seine Größe selbst.

Vom Rechenaufwand her ist es allerdings nicht praktikabel, das Bild von der Originalgröße über alle beliebigen Verkleinerungen bis zum Einpixelbild herunterzuskalieren. Die Anzahl der Pixel stiege — vom Runden auf ganzzahlige Werte für die Bildauflösung einmal abgesehen — bei einem $x_r * y_r$ großen Bild auf astronomische Werte an:

$$\sum_{n=1}^{x_r} n^2 * \frac{y_r}{x_r} = \frac{y_r(x_r+1)(2x_r+1)}{6} \tag{3.14}$$

Für ein CCIR–Videonorm–Bild von $768 * 576$ Pixeln Größe betrüge die Gesamtpixelzahl aller Auflösungsbilder über 100 Millionen Pixel. Eine gemeinsame Skalierung jeweils um den Faktor zwei ist dagegen sowohl vom Rechenzeitbedarf her kein Problem als auch technisch einfach lösbar. Die Pixelanzahl ergibt sich mathematisch als geometrische Folge; bei auf ganzzahlige Werte abgerundeten Bildauflösungen ist sie kleiner als:

$$\sum_{n=0}^{\infty} (1/4)^n * x_r * y_r = 4/3 * x_r * y_r \tag{3.15}$$

Für ein CCIR–Bild fallen jetzt nur noch Auflösungsbilder mit weniger als 600 Tausend Pixeln an. Die Gesamtpixelanzahl aller Pyramidenstufen steigt hier, im Gegensatz zum ersten Ansatz, proportional mit der Bildgröße. Der Auflösungsverlust beträgt, relativ zur Bildgröße gesehen, für jeden Übergang auf eine höhere Ebene der Bildpyramide konstant Faktor zwei in beiden Bilddimensionen (s. Abb. 3.6).

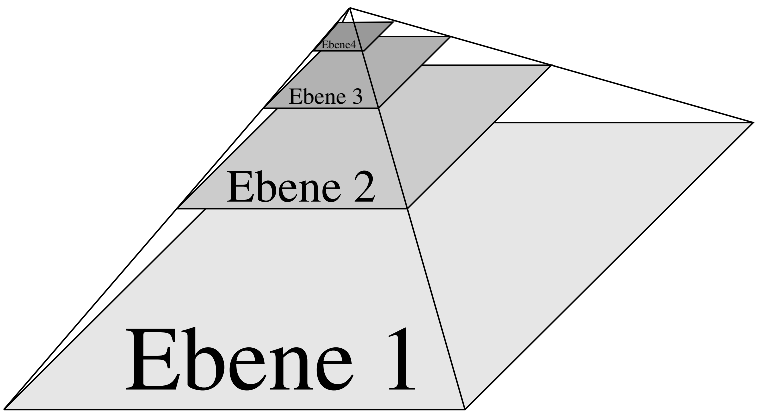


Abbildung 3.6: Pyramide

Ein Herunterskalieren der Bildgröße durch Unterabtastung — der Reduktionsoperator R selektiert dazu in beiden Bilddimensionen jeden zweiten Pixel — ist nach dem Abtasttheorem nur dann ohne Interferenz–Effekte zwischen der Abtastfrequenz und den Bildstrukturen möglich (Moiré), wenn im Bild keine Frequenzanteile größer der halben Abtastrate mehr vorhanden sind. Das Bild der i–ten Pyramidenstufe muß also vor jeder Unterabtastung tiefpaßgefiltert werden, was man zum Beispiel mit einer Gaußfilterung $B^{(i)}$ realisieren kann. Die resultierende Bildpyramide nennt sich daher auch "Gaußpyramide". Der Index i in den nächsten beiden Gleichungen, die [*Jähne 1991*] entnommen sind, dient dazu, zwischen Filterungen auf den verschiedenen Ebenen der Pyramide zu differenzieren. Es handelt sich zwar immer um die gleiche Gaußfilterung, aber wenn auch die Indizes übereinstimmen, wie zum Beispiel bei der Berechnung von $G^{(i+1)}$ und L^i (vgl. 3.16 u. 3.17), wird diese auf dem *selben* Bild der Gaußpyramide durchgeführt, was natürlich nur einmal realisiert werden muß.

$$G^{(i+1)} = RB^{(i)} \bullet G^{(i)} \tag{3.16}$$

Nach der Filterung kann das Bild unterabgetastet werden. Iterativ lassen sich so alle Pyramidenstufen $G^{(i+1)}$ berechnen, wobei die Pyramidenstufe G^0 das Originalbild darstellt. Werden alle Bilder der Pyramide wieder auf die Originalgröße interpoliert, erhält man eine Reihe von Tiefpaßfilterungen mit fortlaufend halbierter Grenzfrequenz. Schon eine direkte Berechnung eines einzigen stark tiefpaßgefilterten oberen Pyramidenbildes würde ein Tiefpaßfilter mit sehr großem Einzugsgebiet erfordern — wenn auch bei entsprechend stark reduziertem Pixeltakt —, worin der Vorteil der iterativen Pyramidenberechnung beim Hardwareeinsatz, also bei konstant vorgegebener Umgebungsgröße der Operatoren, liegt.

Es fällt allerdings auf, daß niederfrequente Bildanteile nicht nur im Bild der letzten Pyramidenstufe vertreten sind, auf der sie noch nicht ausgefiltert worden sind, sondern auch auf allen tieferen Pyramidenbildern. Dabei erschweren sie Merkmalsberechnungen von kleineren, auf den tieferen Ebenen zu detektierenden Objekten. Abhilfe ist durch einen Hochpaß möglich, der alle Bildinhalte entfernt, die auch in der nächsthöheren Gaußpyramidenstufe noch enthalten sind, also jene, die das Tiefpaßfilter passieren können. Dazu kann die Antwort des Tiefpaßfilters einfach von dem Gaußpyramidenbild abgezogen werden (dazu wird in Formel (3.17) in Operatorschreibweise der Tiefpassoperator $B^{(i+1)}$ vom Indentitätsoperator I abgezogen). Die Pyramide, die dabei entsteht, ist die "Laplacepyramide":

$$L^{(i+1)} = (I - B^{(i+1)}) \bullet G^{(i+1)} \tag{3.17}$$

Vergliche man die Bilder der beiden mit einem idealen Tiefpaßfilter berechneten Pyramiden, so würde man bei der Laplace– im Gegensatz zur Gaußpyramide feststellen, daß in jedem Bild nur die Objekte dargestellt sind, die sich in der jeweiligen Rasterung der Pyramidenstufe gerade noch, in der der nächsten aber nicht mehr auflösen lassen. Auch mit einfachen Filtern zeigen die Transferfunktionen Bandpaßcharakter mit jeweils halbierter Durchlaßfrequenz. Mit geeigneten Interpolationsfiltern auf die Ursprungsbildgröße interpoliert, kann man die Bilder der Laplacepyramide, falls das Abtasttheorem nicht verletzt wurde, wieder zum Ursprungsbild aufsummieren.

Die Frequenzselektivität bzw. Güte der Bandpässe beider Pyramiden hängt von der Steilheit der verwendeten Tiefpässe ab. Bei der mittels $9*9$ Gaußfilterung gewonnenen Laplacepyramide sind Objekte in der Ebene unter ihrer Auflösungsgrenze am stärksten vertreten, ober– und unterhalb dieser Ebene auch noch mit je halber Amplitude. Noch eine Ebene höher mit weniger als 5 % Amplitude, was bei manchen Anwendungen kritisch sein mag, da sie sich hier unterabgetastet als Moiré zeigen.

Mit größeren Einzugsbereichen der Filteroperatoren lassen sich anstatt besser dämpfender auch zunehmend steilere Filter realisieren, die die Objektdarstellung stärker auf eine Pyramidenebene konzentrieren. So gibt es Binomialfiltern höherer Klassen mit zunehmend steileren Transferfunktionen, als sie die normalen Binomialfilter der ersten Klasse aufweisen. Allerdings besitzt die kleinste Binomialmatrix n–ter Klasse schon $2n+1$ Koeffizienten. Man kann die zur Verfügung stehenden Koeffizienten also entweder für eine hohe Selektivität der Pyramide oder für geringe Interferenzfehler in Form von Moiré — hervorgerufen durch unzureichende Dämpfung der Signalanteile über der halben Abtastfrequenz — nutzen.

Neben anwendungsspezifischen Genauigkeitsforderungen setzt das Quantisierungsrauschen Maßstäbe für die maximal sinnvolle Filterdämpfung. In [*Jähne 1991*, S. 147] sind Diagramme der Transferfunktionen von Binomialfilter mit $5*5$, $9*9$ und $17*17$ Koeffizienten dargestellt. Soll der Fehler selbst gegenüber dem Quantisierungsrauschen noch klein sein, sind auch Binomialfilter erster Ordnung der letzten genannten Größe noch berechtigt.

3.4 Euklidischer Merkmalsraum

Bei der Bildklassifikation über Merkmale werden gewöhnlich für jeden Pixel einige Merkmale berechnet, die, kartesisch zu einem Merkmalsvektor verknüpft, in einem durch die einzelnen Merkmale aufgespannten mehrdimensionalen Merkmalsraum liegen. Einer Klassifizierung dieser Merkmalsvektoren in verschiedene von vornherein festgelegte Klassen dient die Diskriminanzanalyse ([*Deichsel, Trampisch 1985*]). Davon zu unterscheiden ist die Clusteranalyse, die Häufungen von Merkmalsvektoren im Merkmalsraum, sogenannte Cluster, untersucht. Ein Zusammenhang ergibt sich jedoch insofern, als clusterartige Häufungen (im Gegensatz zu im Raum gleichverteilten Merkmalsvektoren) Voraussetzungen sind, eine erfolgreiche Diskriminanzanalyse durchführen zu können.

Für beide Verfahren ist es jedoch nötig, ein Ähnlichkeitsmaß im Merkmalsraum zu definieren. Es ist dabei naheliegend, die Differenz zweier Merkmalsvektoren $\vec{x}$, $\vec{y}$ in Form einer Metrik, d. h. einer Abbildung $(\vec{x},\vec{y}) \mapsto d(\vec{x},\vec{y})$ mit folgenden Eigenschaften, zu wählen:

$$\begin{array}{llcl} Identit\ddot{a}t: & d(\vec{x},\vec{y}) & = & 0 \;\; genau\ dann\ wenn\ \; \vec{x}=\vec{y} \\ Symmetrie: & d(\vec{x},\vec{y}) & = & d(\vec{y},\vec{x}) \\ Dreiecksungleichung: & d(\vec{x},\vec{z}) & <= & d(\vec{x},\vec{y})+d(\vec{y},\vec{z}) \end{array} \tag{3.18}$$

Das euklidische Distanzmaß ist die Wurzel des kanonischen Skalarprodukts der Differenz zweier Vektoren mit sich selbst und erfüllt die Eigenschaften einer Metrik, allerdings ist es wegen der Wurzel aufwendig in Echtzeit zu berechnen. Zusammen mit dem Merkmalsraum bildet es einen euklidischen Vektorraum.

$$d(\vec{x}, \vec{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} \tag{3.19}$$

Es gibt jedoch auch zahlreiche andere Metriken, z. B. die Schachbrettmetrik. Hier ist die Distanz durch die maximale Distanz zweier Komponenten der Vektoren $\vec{x}$, $\vec{y}$ gegeben (vgl. Zugmöglichkeiten eines Königs auf einem Schachbrett).

$$d(\vec{x}, \vec{y}) = \max\left((x_1 - y_1), (x_2 - y_2), \cdots, (x_n - y_n)\right) \tag{3.20}$$

Besonders verbreitet ist jedoch die Cityblock–Metrik. Hier ergibt sich die Distanz aus der Summe der Differenzbeträge der Komponenten der Vektoren (vgl. Distanz zwischen 2 Orten, bezüglich eines rechtwinkligen Straßennetzes):

$$d(\vec{x}, \vec{y}) = \sum_{i=1}^{n} |x_i - y_i| \tag{3.21}$$

Anhand solcher Distanzmaße lassen sich nun auch in höherdimensionalen Merkmalsräumen Aussagen über Distanzen zwischen Vektoren machen und sich ein Ähnlichkeitsmaß zwischen ihnen einführen. Von den zu untersuchenden Merkmalsvektoren können nun für alle Kombinationen die Distanzen berechnet werden.

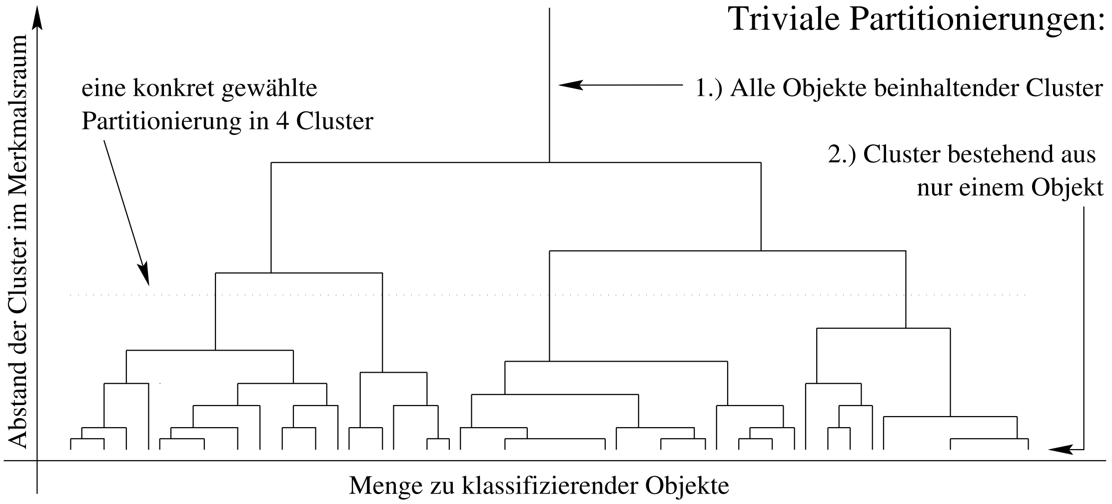


Abbildung 3.7: Hierarchiebaum

Hierarchische agglomerative Verfahren der Clusteranalyse fassen nun zuerst die beiden bezüglich der gewählten Metrik am dichtesten beieinanderliegenden Merkmalsvektoren von Objekten zusammen ([*Deichsel, Trampisch 1985*, S. 17]). Damit bilden die bezüglich der Metrik und der klassifizierenden Merkmale ähnlichsten Objekte im Bild einen Cluster. In einer Tabelle von Distanzen zwischen Objekten führt diese Verschmelzung dazu, daß bei allen Distanzen zu einem Objekt des Clusters, die kleinste Distanz zu irgendeinem Objekt des Clusters eingetragen wird. Dieses Verfahren wird fortgesetzt bis schließlich nur noch ein Cluster existiert. Die Reihenfolge der Clusterbildung und der dabei jeweils aktuellen Distanz wird während der Prozedur protokolliert. Nun kann man feststellen, ab welchem Schritt die Distanzschwelle so deutlich zunimmt, daß man die vorherigen Cluster als Lösung auffaßt. Ohne sich entscheiden zu müssen, kann man auch zu jeder Distanz die Cluster angeben. Die Clusterbildung kann durch einen Binärbaum illustriert werden, wobei die Höhe der Knoten der aktuellen Distanz entspricht. Objekte in unserem Fall sind Paare von Bildkoordinaten im Originalbild einerseits und den berechneten Merkmalsvektoren andererseits, wobei nach letzteren und einer gewählten Metrik die Clusterbildung erfolgt. Bei geeigneten klassifizierenden Merkmalen wird die Ähnlichkeit der Merkmalsvektoren im Merkmalsraum mit einer Ähnlichkeit der visuellen Bildstrukturen bzw. mit dem Auftreten von Objekten der gleichen Objektklasse im Originalbild an den entsprechenden Koordinaten einhergehen.

Dies ist nur ein Verfahren, andere berechnen etwa die Zunahme der Summe aller Distanzen zwischen den Objekten eines Clusters und fügen das Objekt ein, das sie am geringsten vergrößert. Die meisten Verfahren neigen unter konstruierten Fällen zu paradoxem Verhalten, so daß man die Ergebnisse mehrere clusterbildenden Algorithmen vergleichen sollte, wenn man bei höherdimensionalen Räumen auf diese Verfahren angewiesen ist.

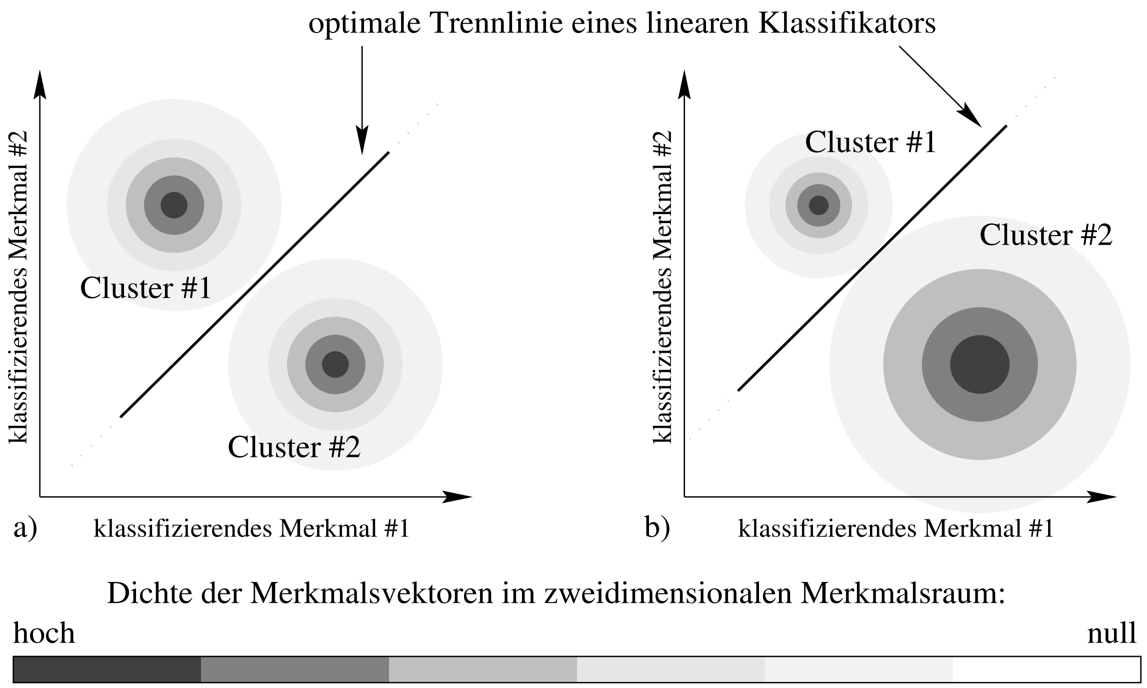


Abbildung 3.8: Klassifizierungstrennlinien

Durch die Angabe von Clustern läßt sich nun auch die Streuung um ihren Mittelwert berechnen. Damit lassen sich nun auf die Streuung normierte Distanzangaben (Mahalanobisdistanz) zu den Clustern einführen. Davon kann auch die Diskriminanzanalyse profitieren, denn die Zuordnung von Merkmalsvektoren zu Clustern nach der geringsten Mahalanobisdistanz führt in vielen Fällen zu weit weniger Klassifizierungsfehlern, als nach der absoluten Distanz (s. Abb. 3.8).

Es lassen sich zur Diskriminanzanalyse aber auch Wege einschlagen, die ohne vorherige Clusteranalyse auskommen. Das Vorwissen über die Strukturierung des Merkmalraums bezüglich Klassenzugehörigkeit muß dann durch eine bereits klassifizierte Stichprobe eingebracht werden. Ein Verfahren besteht darin, in der näheren Umgebung des zu klassifizierenden Merkmalsvektors nach bereits klassifizierten Merkmalsvektoren der Stichprobe zu suchen. Die Klassifizierung erfolgt dann entsprechend der am häufigsten vorgefundenen Klassenzugehörigkeit der Stichprobenvektoren.

Anstatt eine gewisse Umgebung des Merkmalraums abzusuchen kann eine Abwandlung dieses Verfahrens auch darin bestehen, diese Umgebung so weit zu ziehen, bis man eine bestimmte Anzahl an klassifizierten Stichprobenmerkmalsvektoren gefunden hat, und dann wie oben fortzufahren. Eine extreme Auslegung ist dabei die Nächste–Nachbar–Regel. Es wird die Klassifizierung des Stichprobenmerkmalsvektors einfach übernommen, der, bezüglich der gewählten Metrik, am dichtesten bei dem zu klassifizierenden Merkmalsvektor liegt.

Die zuletzt genannten zwei Verfahren sind in Echtzeit nicht realisierbar. Allenfalls die Mahalanobisdistanz als mit einem Normierungsfaktor gewichtete euklidische Distanz zu wenigen Clustern ließe sich noch berechnen. Eine Suche auch nur nach dem nächsten Nachbarn benötigt zahlreiche Distanzberechnungen und Vergleiche, die nicht in Echtzeit realisierbar sind. Eine völlig andere Möglichkeit tut sich allerdings auf, wenn es gelingt, die Dimensionalität des Merkmalraums auf drei bis vier Merkmale mit eventuell eingeschränkter Quantisierung zu beschränken. Hier lassen sich noch komplette Funktionstabellen realisieren, so daß im Vorfeld für alle denkbaren Merkmalsvektoren eine Klassifizierung vorgenommen werden könnte, die dann nur noch abgerufen wird. Bei so einer Realisierung stehen einem selbst komplizierteste Verfahren wieder zur Verfügung. Eine adaptive Steuerung wäre in Form periodischer Updates der LUT vom Hostrechner möglich, der zu diesem Zweck eine Stichprobe der aktuellen Klassifizierungen nehmen und die LUT je nach berechneten Bewegungen der Cluster im Merkmalsraum modifizieren könnte.

Bei bis zu drei Merkmalen pro Klassifikator lassen sich die Merkmalsvektoren im Merkmalsraum auch in dreidimensional animierten Graphen als, je nach Klassenzugehörigkeit eingefärbte, Punkte darstellen. Dabei erschließt sich die Clusterverteilung dem Betrachter ohne weitere Hilfsmittel.

Kapitel 4

Material und Methoden

In diesem Kapitel werden zuerst die charakteristischen visuellen Objektmerkmale von Hölzern beschrieben, die anhand einer Stichprobe erhoben wurden. Dabei werden die Eigenschaften der zur Detektion einsetzbaren klassifizierenden Merkmale abgegrenzt und beanstandungsfreie und zur Aussortierung führende Ausprägungen visueller Merkmale beschrieben. Das Ziel dieser Betrachtung tangiert alle Stufen des Bildverarbeitungssystems. Im einzelnen dient diese Betrachtung dazu, festzustellen, welche Anforderungen an das Bildaufnahmesystem gestellt werden müssen, damit alle zur Klassifikation wichtigen Bildstrukturen erfaßt werden. Dazu gehört es dann auch festzustellen, was für Objekte über die klassifizierenden Merkmale zu detektieren sind und unter welchen Gesichtspunkten die komplexere Bildverarbeitung die Detektionsergebnisse unter eventuell statistischen Aspekten auszuwerten hat. Anschließend werden Bildaufnahmesysteme und –vorgehensweisen unter technischen Gesichtspunkten untersucht, und es wird die konkrete Bildaufnahme des Probenmaterials beschrieben. Schließlich folgt eine kurze Einführung der zum Experimentieren mit den gewonnenen Bilddaten entwickelten Programme.

Das Ziel der Untersuchung der visuellen Merkmale besteht darin, diese Merkmale durch möglichst genau angepaßte klassifizierende Merkmale im automatisierten System zu ersetzen. Ob mit diesen dann eine Objektdetektion möglich ist und ob alle berechneten klassifizierenden Merkmale dazu überhaupt einen Beitrag leisten, kann aus der Verteilung der Merkmalsvektoren im Merkmalsraum geschlossen werden.

Eine Besonderheit ist vielen visuellen Merkmalen von Holz gemeinsam. Sie können in um Größenordnungen verschiedenen Skalierungen bezüglich ihrer Größe auftreten. Um dieses Problem nicht jedesmal aufgreifen zu müssen, wird hier zuerst ein Lösungsansatz der Skalierungsthematik vorgestellt.

Er basiert auf den in [*Fleischer 1993*, S. 15] vorgeschlagenen Bildpyramiden, die das Bild in einem Vorverarbeitungsschritt vor der Berechnung klassifizierender Merkmale in verschiedenen Auflösungen berechnen und diese als neue Originalbilder weitergeben. Die Auflösung zwischen benachbarten Ebenen der Pyramide muß sich dabei, um sie einfach berechnen zu können, jeweils um den Faktor zwei in beiden Dimensionen unterscheiden. Sollte eine feinere Auflösungsstufung erforderlich sein, so kann auf der höherauflösenden Pyramidenebene dasselbe klassifizierende Merkmale zusätzlich noch mit Bildoperatoren, skaliert von der einfachen bis zur doppelten Größe, parallel berechnet werden. Das Ergebnis entspräche dann effektiv einer einzigen Berechnung klassifizierender Merkmale auf einer Pyramide mit feinerer Auflösungsabstufung der Ebenen.

Neben dem Ort von Objekten wird auch ihre Größe durch ein Detektionsergebnis auf der entsprechenden Ebene der Pyramide bzw. von dem für die Größe zuständigen Klassifikator ermittelt. Damit die gerade noch aufgelösten Objekte nicht von größeren Objekten bei der Merkmalsberechnung gestört werden, kommt als Bildpyramide auch die bandpaßfilternde Laplacepyramide zum Einsatz.

4.1 Arbeitsmaterial

4.1.1 Stichprobenumfang

Zur Verfügung stand eine Auswahl von 23 Hölzern aus Eiche zur Parkettgewinnung mit typischen Fehlern, wegen derer die Hölzer von der weiteren Verarbeitung ausgeschlossen worden waren. Leider stand keine repräsentative fehlerfreie Stichprobe zur Verfügung, anhand derer man sich ein Bild der zulässigen Merkmalsstreuung hätte machen können. Die fehlerfreien Bereiche der vorhandenen aussortierten Hölzer mußten als fehlerfreie Stichprobe verwendet werden, die somit zwangsläufig subjektiv idealisiert war. Insbesondere eine Begutachtung von Grenzfällen, die nicht zur Aussortierung führen, war damit nicht repräsentativ möglich. Die Hölzer hatten Abmessungen von 37 bis 40.5 cm, bei einer Breite von exakt 7.1 cm und einer Stärke von etwa einem Zentimeter.

4.1.2 Klassen von Holzartefakten

Typische Objektklassen, nach denen Hölzer klassifiziert werden, sind fehlerhafte Maserung, Wurmlöcher, Astlöcher, Verarbeitungsfehler, Verfärbungen und Risse:

Die Holzmaserung äußert sich bekanntermaßen in abwechselnden helleren und dunkleren Holzbereichen. Da die Übergänge meist kontinuierlich verlaufen, kann man sie mathematisch als Wellenfunktionen einer Ebene auffassen. Die Amplitude entspricht dabei der Helligkeitsschwankung im Bild und die Wellenlänge der Breite eines hellen und angrenzenden dunklen Holzbereiches.

Eine Messung der niederfrequentesten, und damit für den optischen Eindruck relevanten, Maserungswellenlängen an den Probehölzern führt zu folgender Verteilung. (Gemittelt durch Zählung der Maserungslinien der gröbsten Kategorie quer zum Brett an einer fehlerfreien Stelle):

Maserungslinien	6	7	12	14	17	18	20	21	24
Wellenlänge [mm]	12	10	6	5	4	4	4	3	3
Probenanzahl	1	1	1	2	2	1	1	1	1

Maserungslinien	25	28	30	33	34	36	40	55
Wellenlänge [mm]	3	3	2	2	2	2	2	1.3
Probenanzahl	3	1	2	1	2	1	1	1

typische Maserung Die mit dem Auge wahrnehmbare "Maserungswellenlänge" betrug bei subjektiv für fehlerfrei gehaltenen Hölzern 1 bis 5 mm typisch 3 mm, wobei bei genauerem Hinsehen innerhalb der Maserung Submaserungen bis 0.3 mm Wellenlänge auszumachen waren. Solche Feinstrukturen sind vom Betrachter allerdings erst ab wenigen Zentimetern Distanz zum Holz wahrnehmbar und für den optischen Gesamteindruck und somit die Klassifikation unbedeutend. Das Verhältnis der Strukturbreite von dunklem zu hellem Holzbereichen beträgt allerdings entgegen dem Wellenmodell nicht eins zu eins sondern eher eins zu zwei. Das war zu erwarten, da witterungsbedingt (Jahresringe) nicht nur die Holzfärbung sondern auch die Wuchsgeschwindigkeit variiert.

extreme Maserung Je nach Schnittwinkel bezüglich der Wuchsrichtung kann die Maserungswellenlänge beliebig größer werden. Durch die Art der Zerlegung von Stämmen zu Hölzern könnte der Schnittwinkel und mit ihm die maximale Maserungswellenlänge beschränkt sein, jedenfalls traten in der Stichprobe un– oder grobgemaserte Bereiche nur bis maximal 2 cm Breite auf. Dabei wurde besonders bei den grobgemaserten Bereichen ein Breitenverhältnis der dunklen zu den hellen Holzbereichen von bis zu 1 : 10 gemessen.

Bei der Berechnung von Merkmalen der Maserung sind also Wellenlängen von 1 mm bis 20 mm abzudecken. Die bei grober Maserung prinzipiell grobe Ortsauflösung bei der Bestimmung der Maserungsorientierung ist bei lebhaft fein gemaserten Hölzern dabei nicht akzeptabel. Ein Ausweg besteht darin, für die verschiedenen Wellenlängen getrennte Merkmalsberechnung vorzunehmen. Eine Berechnung auf einer Bildpyramide ermöglicht dies mit ein– und denselben Rechenoperatoren, deren Ergebnis, je nach verarbeiteter Stufe der Bildpyramide, das Merkmal Maserung für verschiedene Wellenlängen repräsentiert. Die verringerte Ortsauflösung der oberen Stufe der Pyramide ist hierbei kein Nachteil, da sie mit der inhärenten Ortsunschärfe von Merkmalen grober Texturen einhergeht.

Wurmlöcher Wurmlöcher hatten in der Stichprobe Durchmesser von 0.4 mm bis 1.5 mm. Ebenso wie die Maserung können sie bei schrägem Verlauf zum Anschnitt in einer Richtung beliebig länger werden. Wenngleich dies ihre Detektion wesentlich erschwert, bieten sich auch neue Ansatzpunkte. Denn sie lassen sich, im Unterschied zur Maserung und festen Astlöchern, als Objekte mit einer Ausdehnung in der dritten

Dimension, auch durch besondere Beleuchtungstechniken, zum Beispiel schräges Anleuchten oder Durchleuchten des Holzes hervorheben oder gar mechanisch abtasten.

Astlöcher Zur Beurteilung, in welcher Größe überhaupt Astlöcher auftreten, um entsprechende Merkmale ihres Auftretens berechnen zu können, wurden die Astlochdurchmesser der 23 Stichprobenhölzer vermessen. Es ergab sich folgende Verteilung:

Durchmesser [mm]	1	1.5	2	2.5	3	4	7
Anzahl	4	4	6	4	11	1	3

Rein optisch sind Astlöcher dabei nicht unbedingt von Wurmlöchern zu unterscheiden. Rechnet man diese hinzu, ergeben sich zu detektierende, kontrastartig abgehobene Kreise von 0.4 mm bis 7 mm, wobei sich ein Astloch natürlich — auch wenn es in der Stichprobe nicht der Fall war — über die ganze Holzbreite von 7 cm erstrecken kann. Der mögliche Bereich der Objektgröße der Klasse Astloch erstreckt sich also über 2 Zehnerpotenzen. Eine Detektion mit einer einzigen Faltungsmatrix, die auf kreisförmige Strukturen anspricht, kann damit nur auf einer Bildpyramide gelingen, auf der ein entsprechendes Objektmerkmal, eventuell mit mehrfach skalierten Operatoren (s. o.), berechnet worden ist.

Verarbeitungsfehler Durch fehlerhaft eingestellte Verarbeitungsmaschinen kann es zum Beispiel zu Hobelschlagstreifen kommen. Sie traten nur bei einer Probe mit 3 mm Breite auf. Ihr optisches Erscheinungsbild kommt normaler Maserung nahe, nur daß sie hier rechtwinklig zu dieser verlief, so daß das Holz in beiden Richtungen "gemasert" erschien.

Verfärbungen Bei einer Probe trat an einem Ende eine Verfärbung auf. Charakteristisch war dabei der Übergang der gelb–blonden Holzfarbe zu einem bläulich–grauen Holzton. Im Grauwertbild tritt die Farbänderung lediglich als Helligkeitsgradient in Erscheinung. Eine Detektion auf einer sehr hohen Ebene der Gaußpyramide, bei der jedes Pixel Aussagen über die mittlere Helligkeit eines größeren Bereiches des Holzes macht, kann an tieffrequenten Maserungslinien scheitern, die die mittlere Helligkeit großer Bereiche ebenfalls stark beeinflussen. Eventuell tritt der Effekt in nicht sichtbaren Spektren, etwa im Infraroten oder UV–Licht verstärkt auf, was durch Betrachten über Kameras mit entsprechender spektraler Empfindlichkeit zu erproben wäre. Eine detailliertere Untersuchung ist auf einer Stichprobenbasis von einem Exemplar nicht möglich.

Risse Die Detektion von Rissen als Objekte mit Hilfe von Merkmalsberechnungen ist problematisch, da Risse wegen ihrer länglichen Struktur nicht als Ganzes in den Einzugsbereich eines Operators gelangen. Eine Unterstützung bei der Detektion ist aber dadurch möglich, Risse in Binärbildern herauszuarbeiten und zur Weiterverarbeitung zur Verfügung zu stellen. Das muß nicht in der vollen Bildauflösung geschehen. Nach vorangegangener Erosion, die dunkle Strukturen verbreitert, gehen Risse auch in unterabgetasteten Bildern mit geringerer Auflösung nicht verloren. Eine algorithmische Echtzeit–Auswertung über einen Mikroprozessor ist dann denkbar. Verfahren wurden in [*Kempendorf 1993*] schon vorgestellt, die benötigten Berechnungen zur Vorverarbeitung unterstützt das hier vorgestellte ASIC.

4.2 Bildaufnahme

4.2.1 Bildauflösung von CCIR–Kameras

Zur sicheren Erfassung der Maserung eines Holzes bei der Bildaufnahme sind mindestens vier Abtastungen pro Wellenlänge der Maserung notwendig. Daß lediglich zwei Abtastungen nicht ausreichen, ist offensichtlich, da man zum Beispiel fortlaufend die Nulldurchgänge einer Schwingung abtasten könnte, ohne sie selbst zu bemerken.

Die feinsten zu detektierenden Gebiete haben bei den Wurmlöchern 0.4 mm Strukturgröße. Zum sicheren Treffen der Extrema der Strukturen und einer anschließenden Rauschfilterung — ohne die Extrema zu verlieren — sind 3 Abtastwerte pro 0.4 mm wünschenswert. Bei 7.1 cm Holzbreite entspricht dies 533 Pixel. Diese Forderung wird sowohl von der Horizontal– als auch der Vertikalauflösung einer nach der europäischen Videonorm CCIR arbeitenden Kamera erfüllt (576 Zeilen und deutlich mehr als 300 Linien, also 600 Pixel in der Horizontalauflösung). Die feinsten Maserungen von 0.3 mm Wellenlänge, also 0.15 mm Strukturbreite, benötigten bei 4 Abtastwerten pro Wellenlänge 945 Pixel, sind also nicht mehr aufzulösen. Eine Unterabtastung feiner Strukturen mit Kameras ist jedoch unkritisch, da immer durch einen optischen Tiefpaß — unscharf stellen — vermeidbar. Darüber hinaus sind solch feine Holzstrukturen nicht regelmäßig genug, um ausgedehnte Interferenzen im Bild zu verursachen.

Für eine CCIR–Kamera ergeben sich unter Berücksichtigung der horizontalen und vertikalen Strahlrücklaufzeit, die zu erhöhten Bruttopixelraten führen, folgende Frequenzen und Taktraten:

576 von den 625 Zeilen eines Vollbildes sind sichtbar, die restlichen 49 dienen der Synchronisation und dem Strahlrücklauf. Von den 64 μs Zeilendauer stellen nur 52 μs Bildinhalte da, den Rest belegen Synchronsignale und der Strahlrücklauf, währenddessen zudem Klemmpegel und bei Farbbildern die Farbhilfsträgerreferenz (Burst) übertragen wird. Bei 25 Vollbildern/s ergibt sich eine Zeilenfrequenz von:

$$25\, Vollbilder/s * 625\, Zeilen/Vollbild = 15625\, Zeilen/s$$

Das Bildseitenverhältnis beträgt vier zu drei. Der Pixeltakt sollte bei der Abtastung so gewählt werden, daß sich ein Pixelseitenverhältnis von eins zu eins ergibt. Bei 576 Zeilen entspricht dies 768 Pixeln pro Zeile. Daraus ergeben sich die Bruttopixel pro Zeile zu:

$$\frac{768\,sichtbare\ Pixel/Zeile * 64\,\mu s\ Länge/Zeile}{52\,\mu s\ Bildlänge/Zeile} = 945\,Pixel/Zeile$$

Der für die Echtzeitvideoverarbeitung benötigte Pixeltakt beträgt dann:

$$945\,Pixel/Zeile * 15625\,Zeilen/s \;\approx\; 14.75\,Millionen\ Pixel/s$$

Die entsprechende Videobandbreite ist etwa halb so groß, also 7.5 MHz. Da dies bereits der Videobandbreite nachgeschalteter Videoverstärker nahekommt, muß mit 3 dB Verlust der hochfrequenten Anteile gerechnet werden. Dieses ist nicht mit einer erwünschten Eingangsfilterung vergleichbar, da das Verstärkerrauschen zu hohen Frequenzen hin stark ansteigt. So sinkt der Signalrauschabstand durch gegenläufige Signaldämpfung und Rauschanstieg soweit, daß zwar starke Pegelsprünge noch eindeutig bleiben, etwa Wurmlöcher, aber schwach gemaserte Holzstrukturen, an der Auflösungsgrenze abgetastet, im Rauschen untergehen.

Die Signalrauschabstände von Kameras werden über eine nach hohen Frequenzen abfallenden Bewertungskurve gewichtet angegeben. Das unbewertete Signal–Rauschverhältnis ist je nach Frequenzband bis zu 10 dB schlechter. Für ein dem Quantisierungsrauschen der 8 Bit Abtastung entsprechendes Videorauschen sind 6 dB pro Bit * 8 Bit + 10 dB = 58 dB bewerteter Signalrauschabstand nötig, ein Wert den nur teure Kameras erreichen. Zumindest bei der Bild AD–Wandlung ist daher eine 8 Bit–Quantisierung völlig ausreichend.

Da kleinste Strukturen in Richtung der Maserung nur bei Wurmlöchern auftreten, die bezüglich ihres Kontrastes unkritisch sind, sollte beim Einsatz einer normalen Kamera das Holz in Richtung der Maserung abgetastet werden. Denn obwohl die vertikale Auflösung 1/4 weniger Pixel enthält, weist sie doch bis zur Grenzfrequenz keine Signaldämpfung durch den Videoverstärker auf. Damit wird ein verringerter Rauschabstand des Signalanteils der schwächeren Maserung längs des Holzes vermieden.

4.2.2 Einzelbildanzahl

Eine Abtastung mit den 533 gewünschten Pixeln mittels einer CCIR–Kamera, die 576 Zeilen liefert, ist noch bei bis zu insgesamt acht Prozent Bildrand ober– und unterhalb des horizontal liegend aufgenommenen Holzes gewährleistet. Dabei wird dann gemäß dem Bildseitenverhältnis von 4 zu 3 das Holz in einer Länge von

$$4/3 * 1.08 * 7.1\,cm = 10.2\,cm$$

abgetastet. Es werden dann also gerade vier Vollbilder benötigt, um eines unserer Probenhölzer vollständig abzutasten. Entweder wird die Kamera alle 40 ms um ein Bild weitergeschwenkt, was nicht ohne größere Verzögerung und mechanische Probleme gelingen dürfte, oder man montiert vier fix ausgerichtete Kameras. Sind diese fremdsynchronisiert, geht keine Zeit zur Synchronisation beim Umschalten verloren. Der Zeitbedarf für die Abtastung beträgt somit:

$$4\,Bilder * 1/25\,s/Bild = 160\,ms$$

Eine interessante Alternative wäre eine Zeilenkamera, die das in Längsrichtung mit einigen Metern pro Sekunde vorbeilaufende Holz währenddessen quer abtastet. Unter Verringerung der Abtastrate wären damit sogar beliebig höhere Auflösungen möglich. Bei einer 1024 Pixelzeile und unverändertem Pixeltakt können

$$\frac{14.75\,MillionenPixel/s}{1024\,Pixel/Zeile} \approx 14400\,Zeilen/s$$

pro Sekunde verarbeitet werden. Bei einem Millimeter Überlapp in beiden Richtungen haben die Hölzer eine maximale Größe von 72 mm * 406 mm, also ein Seitenverhältnis von 1 : 5.639 . Somit sind

$$1024\,Pixel/Holzbreite * 5.639\,Holzbreiten/Holzlänge = 5774\,Pixel/Holzlänge$$

abzutasten, so daß

$$\frac{14400\,Zeilen/s}{5774\,Zeilen/Holzlänge} \approx 2.5\,Holzlängen/s$$

verarbeitet werden könnten. Dazu würden 2.5 Hölzer/s der Länge nach unter der Zeilenkamera durchgeschoben. Bei etwa 40 cm ergibt sich eine Geschwindigkeit von einem Meter pro Sekunde, also 3.6 km/h — langsames Fußgängertempo. Dabei ist die Auflösung allerdings in beiden Richtungen auch fast doppelt so groß wie gefordert. Bei Verwendung einer 512 Pixel–Zeilenkamera ergibt sich die vierfache Abtastrate von 10 Hölzer/s bei einer Geschwindigkeit von 14.4 km/h. Neben der 5% geringeren Auflösung in beiden Richtungen und somit einer um 10% verringerten Pixelanzahl gegenüber der einführenden Betrachtung von CCIR–Kameras, ist der Hauptgrund für die deutlich größere Geschwindigkeit beim Einsatz von Zeilenkamera, daß keine Verluste durch Synchronsignale und Strahlrücklaufzeiten entstehen und somit mit jedem Pixeltakt auch Pixel verarbeitet werden können.

Die Bilder haben bei Zeilenkameras dasselbe Seitenformat wie das Holz, was gewöhnlich keiner Videonorm entspricht, dafür aber bei Bildfaltungen keine künstlich geschaffenen Randwertprobleme aufwirft: Alle Bildränder sind auch Holzränder, das Randwertproblem ist dann nicht mehr künstlich geschaffen sondern systeminhärent. Ein konstanter Ersatzwert an den Bildrändern ist der Vorstellung äquivalent, das Holz vor einem neutralen Hintergrund einer mittleren Bildhelligkeit aufgenommen zu bewerten. Andere Lösungen wie Bildspiegelungen mögen im Spezialfall brauchbar sein, können im allgemeinen jedoch zu unerwünschten Artefakten führen (zum Beispiel plötzliche Orientierungsumkehr bei Spiegelung diagonaler Texturen).

4.2.3 Testbilderaufnahme

Die Probeaufnahmen wurden mit einer RGB–Studiokamera und einem 512 * 512 Pixel Framegrabber gemacht. Es wurde neben der diffusen Deckenbeleuchtung auch ein starker Scheinwerfer probeweise eingesetzt. Er führte zu keiner Verbesserung des Bildrauschens, offensichtlich waren die Kameraröhren auch bei Raumbeleuchtung schon voll ausgesteuert. Es wurde der blaue Farbkanal digitalisiert, da er gegenüber dem grünen und roten 50% höhere Streuung aufwies. Von 9 Holzproben mit typischen Fehlern wurden jeweils 6 Bilder eingescannt, das letzte Bild wurde nur zu 1/3 genutzt. Weitere 6 Einzelbilder einzelner markanter Fehler der restlichen Probenhölzer deckten die interessanten Objekte der restlichen Stichprobe von 23 Hölzern ab.

Vor der Bildverarbeitung wurde der — zu geringem Teil sichtbare — weiße Hintergrund durch den mittleren Grauwert der Bilder ersetzt, um Randwerte nachfolgender Verarbeitungen möglichst gering zu verfälschen. Nach der Aufnahme stellte sich heraus, daß der Quantisierungsbereich nur zu knapp 80% ausgenutzt worden war, so daß der Kontrast um 30% angehoben werden konnte, ohne mehr als einige Dutzend Pixel im Holz, die jedoch keinen besonderen Objekten zuzuordnen waren, sondern statistisch streuten, auf den Maximalwert 255 zu begrenzen.

4.3 Verwendete Programme

4.3.1 PC–Bildverarbeitung

Zur Erprobung von Merkmalen wurde ein C–Programm geschrieben, das alle benötigten Bildverarbeitungsfunktionen bietet, um Merkmalsberechnungen von eingescannten Holzbildern auf einem PC durchführen zu können. Die Leistungsmerkmale dieses Programms werden im folgenden aufgeführt, eine Bedienungsanleitung einzelner Befehle findet sich im Anhang. Die Implementationsentscheidung, ob bzw. wieviele Bilder im Hauptspeicher gehalten werden sollen (Architektur), welche Bildgröße möglich ist, welche Pixelquantisierung gewählt und wie Sonderfälle wie Überläufe und Randwerte bei Faltungen gehandhabt werden sollten, wurden folgendermaßen getroffen:

Bildspeicher Das Vorgehen entspricht dabei dem einer 2–Adress–Maschine mit zwei Registern: Es stehen im Hauptspeicher zwei Bildspeicher zur Verfügung, die jeweils das Ziel– und das Quellbild der letzten Bildoperation enthalten. Bei Operationen auf zwei Quellbildern wird eines vom Ergebnis überschrieben. Die beiden Bilder im Speicher können vertauscht werden, um ohne Ein–/ Ausgabeoperationen mehrfache probeweise Operationen auf demselben Quellbild durchführen zu können. Bilder können mit ihrem Zustand (Größe, Ursprungsoffset, signed–/ unsigned–Kennung, Anteil geclippter Werte und undefinierten Bildbereichen bei Pyramiden) oder nur als Bytestrom abgespeichert und geladen werden.

Bildgröße Die mögliche Bildgröße für jedes der beiden Bilder beträgt 2.3 Millionen Pixel (705 * 3264). Die Bildbreite ist (um Seitenwechsel des 16 KB–weise eingeblendeten EMS–Speicher während der Faltungsberechnung ausschließen zu können) auf maximal 3264 Pixel Bildbreite begrenzt.

Wertinterpretation Die Pixelquantisierung ist auf 8 Bit festgelegt. Es werden signed– und unsigned–Darstellungen der Pixel unterstützt. Bezüglich arithmetischer Operationen beträgt das Wertintervall bei der signed–Darstellung [0 ... 1[und bei unsigned–Darstellung [-1 ... 1[, bei 8 Bit linearer Quantisierung. In letzterer sind die absoluten Quantisierungsschritte nur halb so groß wie bei der signed–Darstellung. Bei Berechnungen werden die Ergebnisse daher immer im unsigned–Format abgelegt. Sobald ein negativer Wert auftritt, werden weitere Werte im signed–Format gespeichert und die vorigen Werte nachträglich entsprechend konvertiert. Somit liegen Ergebnisse von Berechnungen immer in der optimal angepaßten Darstellung vor.

Überläufe Überläufe werden auf die jeweils möglichen Extremwerte geclippt. Nach jeder Operation sind die Anteile der geclippten Pixel von den ungeclippten Extremwerten im Histogramm farblich abgehoben markiert. Sowohl die Anzahl übergelaufener Pixel, als auch der Maximalwert der geclippten Werte werden angezeigt.

Randwertbehandlung Bei der Bildverarbeitung werden optional Randersatzwerte eingeblendet, ansonsten ist das Zielbild nur für die Pixel definiert, deren Umgebung nicht über den Bildrand ragt. Der Randersatzwert kann vor Operationen numerisch oder als Linearkombination von Mittelwert und Streuung des Bildes angegeben werden. Zur Berücksichtigung der Verschiebung von Bildern nach Operationen, die die Randwertoption nicht nutzen, in Richtung Ursprung wird eine Offset des Bildes mitgeführt, der eine pixelgenaue Verknüpfung jedes gefilterten Bildes mit seinem Ursprungsbild ermöglicht. Die Bilder werden vor Operationen auf zwei Bildern zueinander passend verschoben, und die Operatoren nur auf in beiden Bildern existierende Pixel angewandt.

Die folgenden Bildverarbeitungoperationen stehen nicht nur auf normalen Bildern, sondern jeweils auch auf den in einem Bild eingebetteten Ebenen der Gauß– und Laplacepyramide (s. Abb. 5.1 und Abb. 5.2) zur Verfügung. (Zur Erzeugung dieser Darstellung dienen die beiden zuletzt genannten Bildoperationen.)

Faltungs– und Rangordnungsoperatoren Die lokale Umgebung von Faltungsoperatoren beträgt zehn Pixel für eindimensionale Binomialfilter und 9 * 9 Koeffizienten für frei definierbare Faltungsmatrizen. Koeffizienten werden als Integerwerte mit einem gemeinsamen Integerteiler eingegeben. Rangordnungsoperationen werden ebenfalls bis zu 9 * 9 Pixel Einzugsfeldgröße unterstützt; zur Sortierung dient das Heapsortverfahren, dessen Implementation aus ([*Wirth 1986*]) übernommen wurde. Dilatation und Erosion sowie eindimensionale Faltungsmatrizen sind zeitoptimiert implementiert.

Arithmetische Punktoperationen Einige Pixeloperationen benötigen zwei Quellbilder und überschreiben mit ihrem Ergebnis dann das zweite Bild. Neben Summe, Differenz, Multiplikation, Division, Logarithmierung und der Exponentialfunktion kann auch eine Rangordnungoperation zwischen je zwei korrespondierenden Pixeln der Bilder gebildet oder ein Binärbild über einen Schwellenwertvergleich erzeugt werden. Das zweite Bild kann durch eine Konstante ersetzt werden. Über einen Reduktionsoperator lassen sich die Bilder durch Unter– und Überabtastung in ihrer Größe skalieren und über einen Clippingoperator auf jeder Seite Ränder beliebiger Größe abschneiden oder anfügen.

Pyramidenimplementation Bei der Bildoperation zur Berechnung der Gauß– oder Laplacepyramide werden alle Bilder der berechneten Ebenen der entsprechenden Pyramide in einem besonderen Bildformat zusammengefaßt. Ungenutzte Bildbereiche werden dabei mit dem Randersatzwert gefüllt und aus Histogrammbetrachtungen und statistischen Größen herausgerechnet. Alle Teilbilder der Pyramide werden untereinander getrennt behandelt, Umgebungen von Operatoren ragen nicht in andere Bildebenen der Pyramide: An den Teilbildern der Pyramiden können ebenso wie bei Einzelbildern Randverluste durch Ersatzwerteinblendung vermieden werden. Pyramidenbilder können als Laplace– oder Gaußpyramide berechnet werden. Die Gaußfilterung erfolgt separiert über Binomialfilter wahlfreier Ordnung.

Laplacepyramiden Bei der Laplacepyramide ist noch ein Verstärkungsfaktor vorgesehen, der eine Signaldämpfung oder –verstärkung um einen konstanten Faktor zwischen den Laplacepyramidenbildern ermöglicht, so daß alle Pyramidenbilder ihren Wertebereich voll ausschöpfen und das Bild als Ganzes mit denselben Operatoren weiterbearbeitet werden kann. Bei der Gaußpyramide fehlt er, da niederfrequente Bildstrukturen auf allen Ebenen auftauchen und exponentiell verstärkt bei jedem Faktor ungleich eins zur Übersteuerung der oberen Ebenen führen würden.

Die berechneten Bilder lassen sich statistisch als Histogramm auswerten, als Merkmalsvektoren von Objekten im Merkmalsraum oder als Bild darstellen. Die Bilder selbst als auch die zu ihrer Berechnung erforderlichen Rechenschritte lassen sich sichern und als Batchjob wiederholen.

statistische Auswertung Ein Histogramm ist für statistische Auswertungen die ideale Aufbereitungsform, da noch alle benötigten Informationen enthalten sind, obwohl die Darstellungsform sehr kompakt ist. Neben der Ausgabe des Mittelwertes und der Streuung der Bilder wird deswegen auch das Histogramm angezeigt. Damit die wenigen Pixel, die nicht in die Verteilung der Masse fallen, nicht durch Rundung in der linearen Darstellung verloren gehen, obwohl sie meist den entscheidenden Informationsgehalt besitzen, wird neben der linearen auch eine logarithmische Darstellung geboten.

Bilddarstellung Bilder können in VGA sowie in VESA–Modi angezeigt werden. Die Anzeige kann linear oder über eine das Histogramm einebnende Gammafunktion erfolgen: In diesem Fall wird jeder Grauwert von der gleichen Anzahl von Pixeln angenommen. Dazu werden die Histogrammlinien so verschoben — ohne ihre Reihenfolge zu vertauschen —, daß gleichgroße horizontale Intervalle des Histogramms die gleiche Anzahl von Pixeln enthalten. Erhalten bleibt dabei die Monotonie, hellere Pixel in der Darstellung entsprechen solchen im Bild. Das Ergebnis ist ein Darstellung, die auch Bilder, deren Histogramm kaum um den Mittelwert streut, mit maximalem Kontrast anzeigt. Histogrammverteilungen mit deutlich getrennt besetzten Intervallen werden durch dieses Verfahren allerdings zusammengerückt und auf volle Breite gedehnt. Ob ein Bild eindeutig über Schwellenwerte segmentierbar ist, ist bei eingeebneter Darstellung nicht mehr zu entscheiden. Wahlweise werden nebeneinander beide Bildspeicher dargestellt. Sind Bilder zu groß, kann der sichtbare Bildausschnitt mit dem Cursor verschoben werden. Mit dem Cursor lassen sich zudem einzelne Pixel selektieren und assoziierte Sollwerte eingeben, die sequentiell in ein Datei geschrieben werden. Mit einem Zusatzprogramm lassen sich die berechneten klassifizierenden Merkmalsvektoren dieser Pixel und ihre Sollklassifikation im Merkmalsraum anzeigen.

Ein–/ Ausgabe Bilder können in einem eigenen Format, das alle Zusatzinformationen abspeichert, oder als reiner Bytestrom gespeichert und geladen werden. Im letzteren Fall muß dazu vorher die erwartete Bildgröße angegeben werden. Wird von einem mit zu wenig Speicher und damit beschränkter Bildgröße gestarteten System aus versucht, ein zu großes Bild des eigenen Formates zu laden, wird der überschüssige rechte und untere Rand abgeschnitten.

Batchverarbeitung Um eine Verarbeitungsoperation eines Bildes, die aus vielen sequentiellen Operationen auf mehreren Zwischenbildern bestehen kann, als Ganzes anstoßen zu können, wird batch–Verarbeitung der Einzeloperationen unterstützt. Ganze Bildfolgen können durch Dateinamen mit inkrementellen Namensbestandteilen und rekursive batch–Aufrufe ohne Benutzereingriff verarbeitet werden. Parameterstudien werden durch einen variablen Integerbezeichner unterstützt. Batches lassen sich durch Ausführen der gewünschten Verarbeitungsschritte, die mitprotokolliert werden, generieren und abspeichern. Optional kann die eigentliche Bildverarbeitung dabei unterbleiben, so daß ein Batchjob, ohne die Verarbeitungszeit abzuwarten, eingegeben und syntaktisch geprüft werden kann. Die Bildparameter folgen dabei der scheinbaren Berechnung, der Bildinhalt ist jedoch undefiniert.

4.4 Entwurfsvorgehen

4.4.1 Arbeitsstadien

Zur Erkundung brauchbarer Merkmale zur Klassifikation wurden zuerst zwei C–Programme für PCs entworfen. Das erste erlaubt, Grauwertbilder von Hölzern mit den in der Literatur geläufigen Operatoren zu bearbeiten, um Merkmale zu berechnen. Das zweite gestattet die Betrachtung von bis zu drei klassifizierenden Merkmalen, die kartesisch verknüpft einen Merkmalsraum aufspannen, in den alle Pixel des Originalbildes als dreidimensionale Merkmalsvektoren abgebildet werden. Der Merkmalsraum wird dazu in Form eines in Echtzeit rotierbaren Raumwürfels dargestellt, in dem die Merkmalsvektoren, farblich kodiert nach einer vorgegebenen Sollklassifikation, als Punkte dargestellt werden. Da Pyramidenebenen hier eingebettet in einem Bild vorliegen, wurde die Entscheidung von welcher Ebene der Pyramide ein Merkmalsvektor dargestellt werden soll, bereits mit der Festlegung der anzuzeigenden Pixelkoordinaten getroffen.

Nachdem die Art der zu unterstützenden Operatoren bekannt war, wurde eine ASIC–Architektur, die sich an Faltungsaufgaben orientiert, entworfen. Zuerst wurden alle benötigten Hardwaremodule einzeln realisiert. Dazu wurden die zu ihrem strukturellen Aufbau benötigten Komponenten auf der funktionalen Ebene in Struktur bzw. Verhalten beschrieben und simuliert. Anschließend wurde auf der algorithmischen Strukturebene das Gesamtdesign aus diesen Komponenten aufgebaut, durch Simulationsläufe ausgetestet und optimiert.

Dabei stellte sich heraus, daß wegen des Zeitversatzes bei der Kaskadierung, aufgrund einiger Optimierungen des Designs, die die Regularität störten, und wegen der bitorientierten nicht an Wortgrenzen liegenden Register eine manuelle Erstellung einer ASIC Konfigurationsdatei viel zu komplex und fehlerträchtig wurde. Parallel dazu mußte daher noch Software zur Programmierung des ASICs entwickelt werden. Bild D.2 gibt einen Überblick über die Einzelarbeiten und den Stand des Gesamtprojekts.

Zuletzt wurden die Komponenten modifiziert, bis alle synthetisierbar waren. Wegen der Komplexität des Designs war ein Herunterskalieren des ASIC–Entwurf auf die halbe Größe und die getrennte Synthese aller Komponenten der algorithmischen Ebene notwendig. Nur so ließ sich die VHDL–Beschreibung der funktionalen Ebene in vertretbarer Zeit (innerhalb eines Tages) auf Strukturbeschreibungen der Logikebene (Mapping auf Standardzellbibliotheken) abbilden.

4.4.2 VHDL–Entwurf

Zur Formulierung des ASICs wurde VHDL ([*Synopsys 1995*]) eingesetzt. Verhaltensbeschreibungen wurden zur Abstraktion von der Zellbibliothek auf der Logikebene (Multiplexer, Flipflops, Adder) und für einfache Komponenten der funktionalen Ebene (Schieberegister, serielle Shiftregister und Barrelshifter) eingesetzt. Außerdem wurde das Steuerwerk für den Zeilenspeicher, die Registerinitialisierung, die Ansteuerung der LUT, die Randbereichszähler und für die Randwerteinblendung aus Verhaltensbeschreibungen synthetisiert. Die Beschreibung der Rechenwerke erfolgte auf der Strukturebene. Der Entwurf wurde auf Gatterzellen abgebildet und im EDIF–Format gesichert. Mit dem Programm Cadence ([*Cadence 1994*]) wurde der Standardzellenwentwurf dann weiterverarbeitet.

Kapitel 5

Voruntersuchung

Das folgende Kapitel befaßt sich mit den ausgewählten Verfahren zur Berechnung der Bildpyramide, den klassifizierenden Merkmalen auf den Bildern der Pyramidenebenen und der Ermittlung der konkreten Parameter einzelner Operatoren. (Im Anhang A erfolgt eine Kommentierung der Batchjobs zur automatischen Berechnung der klassifizierenden Merkmale der Testbilder, die mit der entwickelten Bildverarbeitungssoftware abgearbeitet wurden.)

Den Abschluß dieses Kapitels bildet eine Betrachtung der Cluster von Merkmalsvektoren der Objektklassen Astlöcher und lebhafte Maserung im Merkmalsraum. Dies erfolgte als räumliche Darstellung der Merkmalsvektoren einiger vorklassifizierter Objekte und der Merkmalsvektoren von zufällig ausgewählten Bildpositionen, an denen keine Objekte auftauchten. Die Merkmalsvektoren wurden dazu aus den Bildern der klassifizierenden Merkmale, mit Hilfe eines zu diesem Zweck implementierten Programms, gewonnen.

5.1 klassifizierende Merkmale

Als erster Schritt der Bildverarbeitung wurde die Laplace– und Gaußpyramide berechnet. Zu deren Berechnung wurden in beiden Koordinatenrichtungen separierte Gaußfilter achter Ordnung, also Binomialfilter mit neun Koeffizienten, verwendet. Die Anzahl ergab sich daraus, daß zwei ASICs mit je 40 Koeffizienten und der Verknüpfungsmöglichkeit des Zentralpixels Matrizen mit bis zu 81 Koeffizienten realisieren und somit das Ergebnis des Bildverarbeitungsprogramm genau reproduzieren können.

Die zusätzlichen Ebenen der Pyramide wurden mit dem Bild in Originalauflösung zu einem gemeinsamen Bild zusammengefaßt. Dabei wurden die Einzelbilder wie getrennte Bilder behandelt, die im weiteren jedoch alle gleichzeitig von jedem Operator bearbeitet wurden (single instruction — multiple data). Auf diesem Bild wurden alle weiteren Berechnungen klassifizierender Merkmale vorgenommen. Dabei stellte sich jedoch als gravierender Unterschied der Pyramidenebenen heraus, daß in allen oberen Ebenen der Pyramiden kaum

Anteile an höchstmöglichen Frequenzen auftraten. Diese waren bei der Berechnung der jeweiligen Pyramidenstufe stark gedämpft worden. Im Bild voller Auflösung war dies jedoch nicht der Fall. Um dieselben Operatoren auf allen Ebenen der Pyramide — auch der untersten — einsetzen zu können, war eine Dämpfung der höchsten Frequenzen im Originalbild erforderlich.

Als initiale Filterung des Originalbildes wurden nun versuchsweise Binomial– und Rangordnungsfilter eingesetzt. Zur Abschätzung der danach möglichen Detektionsleistung wurde anschließend die erreichbare Detektionsicherheit über einen angepaßten Schwellenwert auf der Laplacepyramide erprobt. Binomialfilter größer $2*2$ Pixel sind hierbei kritisch, da dies bereits die Größe praktisch vorkommender 0.4 mm Holzwurmlöcher ist. Bei einer $4*4$ Pixel Binomialmatrix waren bereits die meisten Wurmlöcher verschwunden, bei $2*2$ Pixeln waren sie noch besser als bei $3*3$ Pixeln detektierbar. Rangordungsfilter als Dilatations– oder Erosionsfilter solch kleiner Größe dämpften das starke hochfrequente Pixelrauschen nicht, so daß 0.4 mm Wurmlöcher undetektierbar wurden. Nach Medianfilterung wurde schon dunklere, pixelbreite Feinmaserung fälschlicherweise mitdetektiert und die 1 bis 2 tiefschwarzen Pixel pro Astloch herausgefiltert, so daß diese nicht mehr zu detektieren waren. In den Versuchsreihen lieferte ein — mit Hinblick auf die quer zur Maserung feineren Strukturen — in Maserungsrichtung drei und quer dazu, zwei Pixel großes Binomialfilter die besten Ergebnisse.

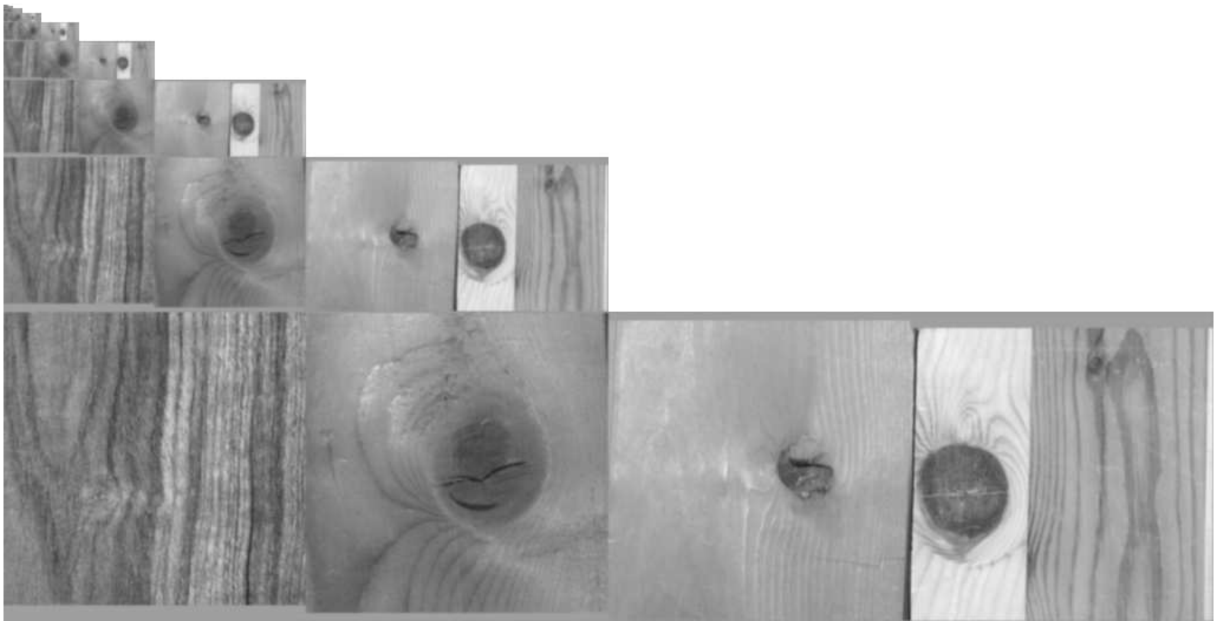


Abbildung 5.1: Gaußpyramide eines Testbildes, berechnet mit dem Programm *bild_ver*

Abb. 5.1 zeigt die nach der initialen Filterung mit dem Bildverarbeitungsprogramm berechnete Gaußpyramide. Um die Auswirkung auf möglichst viele verschiedene Objekte zeigen zu können , wurde statt des Vollbildes eines Holzes, ein aus vier interessanten Holzbereichen verschiedener Hölzer bestehendes Testbild als Originalbild verwendet. Die Abbildung der Originalbildes erübrigt sich, es ist (bis auf die initiale Binomialfilterung) mit dem Bild der Basisebene der Gaußpyramide identisch.

Bei der Bestimmung des Orientierungsvektors auf der Laplacepyramide fiel zudem auf, daß zumindest bei der gewählten Holzsorte die feine Maserung viel kontrastreicher war, als die gröbere. Bei der gemeinsamen Berechnung verschwand entweder der Orientierungsvektor bei den gröberen Maserungsstrukturen, wie sie sich in den oberen Ebenen der Pyramide fanden, im Quantisierungsrauschen, oder derjenige der unteren Ebenen entartete durch Übersteuerte infolge einer Kontrastanhebung bis zur Signumfunktion. Abhilfe schafft hier ein konstanter Faktor, mit dem die Koeffizienten des Gaußtiefpaßfilters multipliziert werden, was den Kontrast bei der Berechnung der nächsten Pyramidenebene jeweils um einen bestimmten Betrag anhebt. Für die Berechnung der Laplacepyramide (s. Abb. 5.2) wurde der Faktor 1.33 gewählt. Da die Ergebniswerte nur einen geringen Teil des Wertebereichs ausnutzten, wurden sie zudem um den Faktor vier verstärkt.

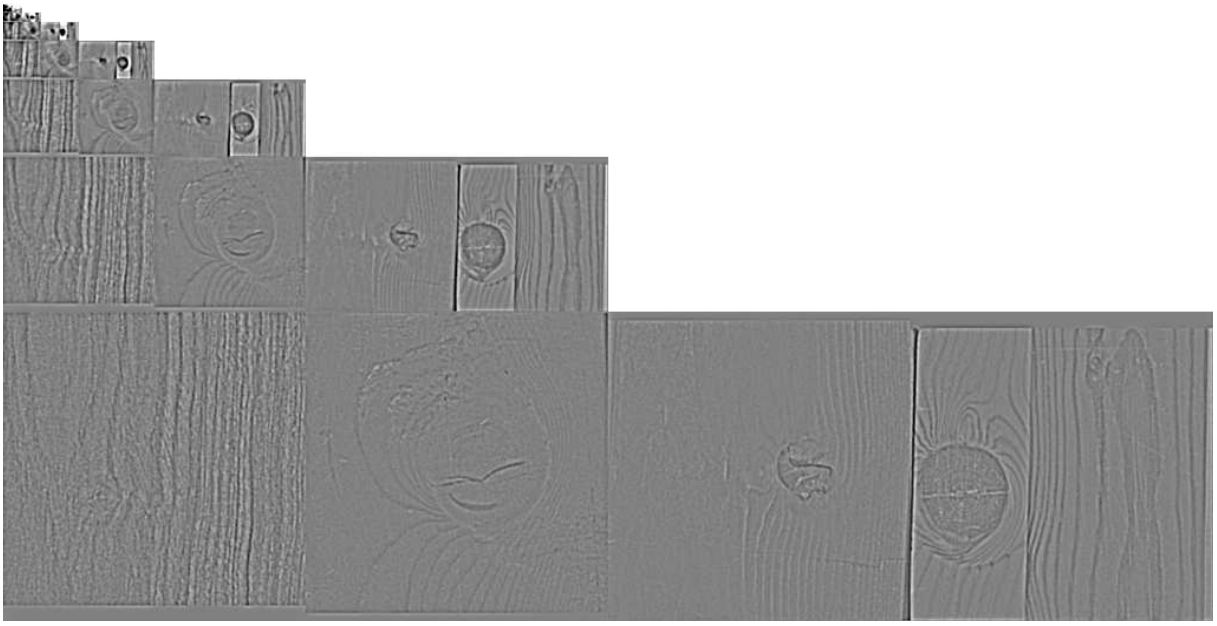


Abbildung 5.2: Laplacepyramide des Testbildes, berechnet mit dem Programm *bild_ver*

Wie Abbildung 5.1 zeigt sind die Bilder von Astlöchern sehr verschieden. Für eine allgemeine Klassifikation kann es Vorteile bringen, mehrere Merkmale zu berechnen, die auf verschiedene Ausprägungsarten von Astlöchern optimiert sind und sich ergänzen.

Objekte verlieren in bandpaßgefilterten Bildern, die ihrer Größe angepaßt sind, scharfe Randkonturen, da scharfe Bildkanten im Frequenzraum starke hochfrequente Spektralanteile besitzen, die ausgefiltert wurden. Das typische Ergebnis ist ein "Überschwingen" der Randkonturen. Zuerst wurde versucht, dieses visuelle Merkmal nach einer Laplacepyramidenfilterung nach dem Prinzip des *pattern–matching* mit einer Faltungsmatrix zu detektieren. Um die Koeffizienten dieser, als *mexican–hut*–Filter bekannten, hier benötigten

Filterform zu erhalten, wurde die radialsymmetrische Faltungsfunktion $\cos(r) * \exp(-r)$ an den diskreten Feldern einer $9*9$ Matrix ausgerechnet ($-\pi <= x, y <= \pi$). Das Ergebnisbild dieses klassifizierenden Merkmals unterschied sich jedoch kaum von dem der Laplacepyramide, die als Quellbild diente. Die Erklärung besteht darin, daß auch der *mexican–hut*–Operator im Frequenzbereich eine Bandpaßfiltercharakteristik besitzt und so nur die Güte der Bandpaßfilterung der Laplacepyramide verbesserte. Daher wurde stattdessen das Pyramidenbild der Laplacepyramide als klassifizierendes Merkmal verwendet. Es reagiert wegen der Bandpaßfilterung besonders sensibel auf die Objektgröße.

Die absolute Helligkeit des Zentralpixels im Originalbild bietet zur Astlochdetektion ein wichtiges klassifizierendes Merkmal, das bei Bandpaßfilterungen durch Dämpfung der niederfrequenten Strukturen negativ verfälscht werden kann. Dies erfolgt immer dann, wenn die Bandpaßfilterung nicht genau auf die Objektgröße abgestimmt ist, was im allgemeinen der Fall ist. Die absolute Helligkeit wurde daher ebenfalls in Form der Gaußpyramide als klassifizierendes Merkmal zur Astlochdetektion verwendet.

Der Einsatz von Rangordnungsverfahren in Form einer Dilatationsfilterung wurde schon durch [*Kempendorf 1993*] mit Erfolg erprobt und ausführlich diskutiert. Die grundlegende Idee bestand darin, daß — im Vergleich zur Operatorumgebung — kleine dunkle Astlöcher im dilatierten Bild verschwinden, da diese jeweils die Werte der maximalen Helligkeit der lokalen Umgebung des Operators enthalten. Bei der Differenzbildung zum Originalbild tauchen nun diese verschwundenen dunklen Bildstrukturen als einzige Objekte signifikant auf (s. Abb. D.10). Da das Originalbild hier auch als klassifizierendes Merkmal vorliegt, kann die Differenzbildung auch bei einem linearen Klassifikator durch einen negativen Koeffizienten des Merkmals Gaußpyramide geschehen. Das neu hinzukommende klassifizierende Merkmal wäre dann lediglich das Dilatationsergebnis. Da das Ergebnis dieses Verfahrens jedoch eventuell zu Segmentierungszwecken von Rissen genutzt werden soll, ist eine Differenzbildung schon bei der Berechnung vorzuziehen, was mit dem entwickelten ASIC ohne Mehraufwand möglich ist.

Zur Klassifizierung stehen somit die drei klassifizierenden Merkmale “Laplacepyramide”, “Gaußpyramide” und “Dilatation auf der Laplacepyramide” zur Verfügung. Dabei werden hier die Originalbilder (Pyramiden) zusammen mit den Berechnungsverfahren genannt, da sowohl zwei verschiedene Pyramiden verwendet wurden als auch zweimal die Berechnung des klassifizierenden Merkmals aus dem Identitätsoperator bestand — also entfiel. Die Eigenschaft, daß Objekte in Pyramiden bei Filtern üblicher Größenordnung in bis zu drei benachbarten Ebenen signifikant auftauchen, könnte dazu führen, daß mindestens ein Merkmal die Größe der Objekte stärker berücksichtigen müßte, um eine verbesserte Detektionssicherheit zwischen den Ebenen der Laplacepyramide zu erreichen. Dazu böte sich bei Bedarf dann doch wieder die *mexican–hut*–Filterung auf der Laplacepyramide an, die die Selektivität der Bandpaßfilterung verbessert (zumal, wenn der Pixelstrom ohnehin von einem ASIC zur Kompensation der Berechnungsdauer der Dilatation verzögert würde).

Eine qualitative Betrachtung der klassifizierenden Merkmale einzelner Objektklassen führt zu folgender erwarteter Verteilung der Merkmalsvektoren im Merkmalsraum.

Merkmale	Gaußpyramide (Helligkeit)	Laplacepyramide (Bandpaß)	Dilatationsverfahren auf der Gaußpyramide)
normal	mittel	null	klein
Astloch	dunkel	negativ	groß

Zur Maserungsbewertung dienen die drei klassifizierenden Merkmale, die unmittelbar bei dem in Abschnitt 3.1.4 vorgestellten Verfahren zur Texturorientierungsanalyse anfallen: Der Orientierungsvektor, der die "Vorzugsrichtung der Maserung" angibt liefert mit seinen beiden Komponenten zwei Merkmale und die "Texturintensität", die zur Unterscheidung zwischen isotroper (zum Beispiel wegen quer zur Maserung verlaufender Hobelschlagstreifen) oder fehlender Maserung dient. Über die Berechnung auf der Pyramide fällt auch noch die Maserungsfrequenz als Verteilung der Maserungsintensität auf den verschiedenen Pyramidenebenen ab.

Orientierungs– Merkmale	vertikal – horizontal	diagonal	große Textur– intensität in
normal	positiv	null	einer Laplace–Ebene
streifig	positiv	null	zwei Laplace–Ebenen
lebhaft	null	$\neq$ null	einer Laplace–Ebene
Hobelschlag	negativ	null	zweiter Laplace–Ebene

5.2 Merkmalsberechnung

Als Ausgangsbilder dienten die i in einem Bild eingebetteten Ebenen der Gaußpyramide (G^i), die alle mit einem $9 * 9$ Gaußfilter B (vgl. $3 * 3$ Gaußfilter Abb. 3.2) nach der iterativen Berechnungsvorschrift der Gleichung 3.16 berechnet wurden. Das Eingabebild wurde vorher jedoch noch, wie bereits besprochen, einer initialen Tiefpaßfilterung durch eine $3 * 2$ Binomialmatrix unterworfen.

Zur Berechnung der klassifizierenden Merkmale wurden nun zur Astlochdetektion folgende Bildoperationen auf der eingebetteten Gaußpyramide durchgeführt (I ist der Identitäts–, D der $9 * 9$ Pixel Dilatationsoperator):

Astlochdetektion: Klassifizierende Merkmale

- Originalbild: I
- Laplacefilterung: $I - B$
- Dilatation: $I - D$

Die klassifizierenden Merkmale zur Detektion fehlerhafter Maserung wurden auf einer Laplacepyramide berechnet, deren Ebenen jeweils um den Faktor 1.33 gegenüber der vorherigen angehoben wurden. Die folgenden Operatoren beziehen sich also auf das Ergebnisbild des Operators $1.33^i(I - B)$ angewandt auf das Bild der eingebetteten Gaußpyramide. Um die Merkmalsbilder voll auszusteuern, wurde die Ergebnisse zusätzlich noch um einige Faktoren verstärkt. (D_x ist der symmetrische Differentialoperator in X–Richtung (rechts in Abb. 3.4), D_y derselbe Operator in Y–Richtung angewandt. Das Symbol "$\bullet$" steht hier anstelle der Faltung für das Punktprodukt zwischen zwei Bildern und B wiederum für eine $9 * 9$ Pixel große Binomialmatrix):

Detektion fehlerhafter Maserung: Klassifizierende Merkmale

- X–Orientierungsvektor: $4 * B(3 * (D_y \bullet D_y - D_x \bullet D_x))$
- Y–Orientierungsvektor: $4 * 2B(3 * D_y \bullet D_x)$
- Texturintensität: $16 * B(3 * (D_y \bullet D_y + D_x \bullet D_x))$

Die Berechnungen wurden mit dem erstellten Bildverarbeitungsprogramm vorgenommen. Das genaue Vorgehen dabei kann im Anhang A kommentiert nachgelesen werden. Die Berechnung laßt sich mit einigen Exemplaren des hier entwickelten ASICs ganz genauso durchführen, da alle hier verwendeten Operatoren zur Verfügung stehen.

5.3 Merkmalsraum

Aus den nun in Form von sechs Merkmalsbildern vorliegenden klassifizierenden Merkmalsvektoren wurde mit Hilfe des Bildverarbeitungsprogramm (s. Anhang B, Befehl v) Stichproben für die zu detektierende Objektklassen Astloch, fehlerhafte Maserung und fehlerfreies Holz gezogen. Die vorgegebene Merkmalsvektoren wurden von dem Programm *cluster* im Merkmalsraum graphisch dargestellt. Der einzelne Merkmalsvektor, der ein Objekt einer vorgegebenen Objektklasse war, wurde — entsprechend dieser Klasse farbig kodiert — als Punkt dargestellt. Um nicht nur zwei– sondern auch dreidimensionale Merkmalsräume abbilden zu können, wurden die Merkmalsvektoren im Merkmalsraum in einer in Echtzeit wählbaren Projektionsrichtung auf die Bildebene projeziert. Ebenso wie man ein dreidimensionales Objekt durch Schwenken aus mehreren Richtungen betrachtet, um einen räumlichen Eindruck zu gewinnen, läßt sich hier der dargestellte Raumwürfel mit den Cursortasten beliebig schwenken.

Eine komplette Darstellung aller 1.5 Millionen Merkmalsvektoren pro Bild auf einem Bildschirm ist jedoch weder übersichtlich, noch ist seine Projektion in Echtzeit zu verändern. Es wurden daher etwa ein Dutzend zu detektierender Objekte ausgewählt, und ebensoviele Punkte blind gezogen (wobei im nachhinein überprüft wurde, daß man nicht zufällig wieder ein Objekt getroffen hatte). Die Prozedur bestand darin, mit dem Graphikprogramm das Originalbild als Gaußpyramide zu laden, im Displaymodus einen Pixel mit einem Cursor zu markieren und für ihn anschließend den Wert der Sollklassifikation einzugeben. Die Liste der Pixel und Sollklassifikationen wurde dabei in eine Datei geschrieben.

Anschließend wurden im Fileheader die Dateinamen der klassifizierenden Merkmalsbilder eingetragen. Das Programm *cluster* liest nun die klassifizierenden Merkmalswerte an den gewünschten Positionen aus den als Dateien abgespeicherten Bildern ein und stellt sie unter der gewünschten Blickrichtung auf dem Bildschirm dar. Findet man bei einer beliebigen Blickrichtung eine Gerade, die die Punkte farblich — also nach Klassenzugehörigkeit — trennt, so ist auch eine Trennung mit einem linearen Klassifikator möglich. Das Verfahren bietet den Vorteil, daß man beliebige Clusterformen sofort erkennt, während man bei Anwendung mathematischer Verfahren implizit mit angibt, nach welchen Clusterarten man überhaupt sucht. Bei den Versuchen bezüglich der Astloch und Maserungsdetektion gelang eine Trennung über einen linearen Klassifikator auf Anhieb. Zur Absicherung wären natürlich deutlich mehr Merkmalsvektoren verschiedenster Hölzer zu betrachten, was allerdings eine sehr zeitaufwendige Arbeit darstellt.

Eine nach Klassenzugehörigkeit farbliche Darstellung der Merkmalsvektoren im Merkmalsraum, die das Programm *cluster* geliefert hatte war bereits im Vortrag zur Vorstellung dieser Arbeit gezeigt worden. Im einzelnen erlaubt dies Programm die Drehung der Clusterdarstellung um alle drei Achsen in Echtzeit (Cursortasten und Seitentasten) eine Größenskalierung ("%" und "*") und ein Tauschen der zur Colorierung herangezogenen vorgegebenen Merkmale ("+" und "-"), ferner können zu Testzwecken in den Merkmalsbildern die Pixelpositionen der dargestellten einzelnen Merkmalsvektoren markiert werden (Option *–mark*), und Grauwert–*screendumps* ausgegeben werden ("w"). Wie schon im Programm *bild_ver* wurde die eingesetzte Heapsortimplementierung aus Wirth übernommen.

5.4 Stellung des ASICs im Systemverbund

Im Vorgriff auf die technische Realisierung soll hier in Abb. 5.3 das Strukturbild eines kompletten Bildverarbeitungssystems gezeigt werden, um die Stellung des ASICs im Systemverbund zu verdeutlichen. Bei diesem Entwurf wird im oberen Verarbeitungszug das Bild der Pyramidenbasis, im unteren dagegen werden alle höheren Ebenen der Bildpyramide, in einem einzigen Bild eingebettet, verarbeitet. Die dunkler dargestellten Komponenten sind aus konfigurierten ASICs des hier entwickelten Typs aufgebaut, die weißen Kästen symbolisieren nur das Umfeld der "einfacheren Bildverarbeitung".

Die gesamte Berechnung klassifizierender Merkmale ist hier in zwei gleichen *black–box*–Komponenten mit der Bezeichnung "Merkmalsberechnung" zusammengefaßt, die von mehreren ASICs realisiert werden, ansonsten fällt pro dunklem Kasten je ein ASIC des entwickelten Typs an. Die heller gezeichneten Kästen stehen für nicht realisierte noch fehlenden Komponenten. Auf der rechten Seite sind dies die linearen Klassifikatoren, auf der linken ein Pyramiden–ASIC, das eine Berechnung und Einbettung der Bildpyramide in Echtzeit unterstützt. Die randlos gezeichneten Kästen entsprechen formal einem Operator zur Unterabtastung, was jedoch durch das noch fehlende Pyramiden–ASIC durch entsprechende Ansteuerung der beiden Bildspeicher (links im Bild) realisiert werden könnte. Im letzten Kapitel wird das zur Systemrealisierung noch zu entwickelnde Pyramiden–ASIC kurz charakterisiert.

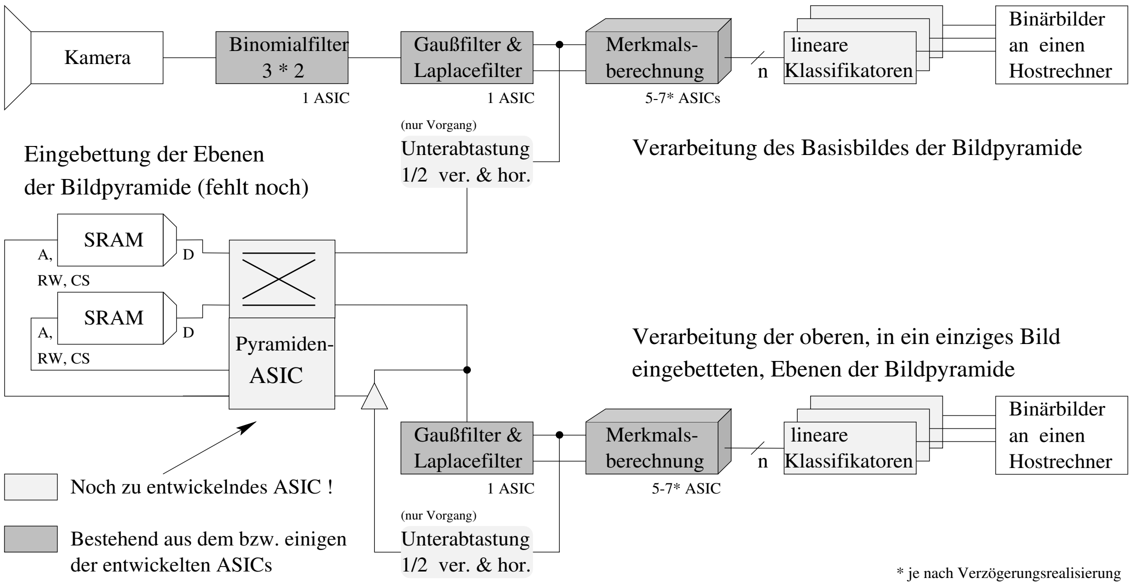


Abbildung 5.3: Pyramidenberechnung im Systemverbund

Das ASIC sollte also Faltungs– und Dilatationsoperatoren unterstützen, andere wurden hier nicht eingesetzt. Um eine $9 * 9$ Matrix zur Bildpyramidenberechnung oder noch wesentlich größere, wie in [*Jähne 1991*, S. 146] vorgeschlagen wird, realisieren zu können, sollte die Verarbeitung der Faltungsmatrix zudem auf beliebig viele kleinere ASICs aufgeteilt werden können. Da ansonsten nur kleinere Matrix–Faltungen anfallen, würden die ASICs ansonsten nicht voll ausgenutzt. Die fixen dominierenden Herstellungskosten des ASICs zur Erstellung des Maskensatzes etc., die von der Chip–Fläche abhängen, würden so deutlich geringer ausfallen. Da schon bevor die Parameter feststanden mit der Erstellung des ASIC–Entwurf begonnen wurde, wurde er voll parametrisierbar beschrieben.

Im folgenden Kapitel soll die Parametrisierung einzelner Verarbeitungseinheiten besprochen werden, die sich so oder ähnlich in allen Faltungs–ASICs wiederfinden. Es ist dazu noch kein Wissen über die konkrete Implementation erforderlich, die erst ein Kapitel später beschrieben wird. Im zweiten Teil des nächsten Kapitels werden dann einige prinzipielle Architekturen und Vorgehensweisen der einzelnen Komponenten der Hauptblockebene besprochen. Obwohl der entwickelte ASIC in allen Bereichen parametrisierbar ist, wird im folgenden nur auf die konkret gewählte Parametrisierung eingegangen. (Sonst stände anstatt jeder Zahl im Text eine lange interpretationsbedürftige Formel.)

Kapitel 6

ASIC: Parametrisierung und Architekturen

Nach Vorüberlegungen, die die internen Parametrisierung der arithmetischen Einheiten betreffen wird im folgenden zuerst eine Anforderungsliste an den ASIC–Entwurf erstellt werden, in der die wesentlichen äußeren Betriebsparameter und –arten, wie Geschwindigkeit, Busbreite, zu unterstützende Operatorumgebungen usw. festgelegt werden. Ein wichtiger Aspekt betrifft die Rechengenauigkeit. Einsparungsmöglichkeiten durch Reduktion der internen Wortbreiten sind bei diesem ASIC, das zum Großteil aus Rechenwerken besteht, besonders zu beachten. Der meiste Platz wird von den Multiplikationswerken der Koeffizienten belegt, die deshalb nur mit den Wortbreiten instantiiert werden sollten, die tatsächlich benötigt werden (also nicht unbedingt nur Zweierpotenzen). Dies kompliziert die rekursiv angelegte VHDL–Beschreibung und macht eine vorgeschobene Betrachtung der zu erwartenden Instantiierungsvarianten nötig. Nach der Auswahl der verbliebenen noch freien Parameter erfolgt eine Vorstellung grundsätzlich möglicher Verzögerungsarchitekturen und schließlich die Entwicklung der implemetierten ASIC–Architektur.

6.1 Anforderungsparameter

Um Videosignale nach CCIR–Norm verarbeiten zu können, ist ein Pixeltakt von mindestens 14.75 MHz nötig (4.2.1). Um auf die Pixel aller Bildzeilen der Zentralpixelumgebung zur Berechnung der Faltung zeitgleich zugreifen zu können, obwohl sie nur einmal angeliefert werden, bedarf es eines Zeilenspeicher mit einer Kapazität von mehreren Bildzeilen. Auf dem Chip ist dieser Speicher wegen seiner Größe schlecht zu realisieren, als externe Komponente wird man ihn aus Kostengründe in Form von Standardspeicherbaustein realisieren und keine Schieberegister o. ä. wählen wollen. Wie im nächsten Abschnitt gezeigt eignen sich externe statische RAMs als kostengünstiger Zeilenspeicher. Alle anderen Komponenten einer Bildverarbeitungseinheit lassen sich auf dem ASIC integrieren.

6.1.1 Unabhängige Parameter

Die Unterstützung höherer Bildauflösungen verursacht bei konstantem Pixeltakt kaum Kosten, da sie zu geringeren Bildwiederholraten führt und somit keine Steigerung der zu erbringenden Verarbeitungsleistung bewirkt. Es müssen lediglich breitere Zähler und Vergleichstabellenregister für die aktuelle Bildposition bzw. die Bildbereichsaufteilung des Gesamtbildes (eingebettete Bildpyramiden) und größere Zeilenspeicher vorgesehen werden. Da der Zeilenspeicher hier extern realisiert wurde und daher nur bei der tatsächlichen Verwendung hoher Auflösungen entsprechende Kosten verursacht, wurde die maximale Bildgröße auf 4096 * 16384 Pixel festgelegt. Dies erlaubt z. B. quadratische oder rechteckige Bilder bis zu einem Seitenverhältnis von sechzehn zu eins mit einer Auflösung von 16 Millionen Pixeln zu verarbeiten. Die Bildwiederholrate liegt dann allerdings nur noch bei etwa einem Hertz. Bei dem Anwendungsgebiet der Holzklassifikation kann z. B. die höhere Vertikalauflösung von 16384 Pixeln zur Längsabtastung des Holzes bei einer Horizontalauflösung von 1024 Pixeln zur Abtastung quer zum Holz mittels einer Zeilenkamera genutzt werden. Als Pixelwortbreite wurde die für die Kamerasignale AD–Wandlung ausreichende acht Bit–Quantisierung gewählt (vgl. 4.2.1).

Benötigt, und von diesem ASIC unterstützt, werden Faltungs–, Dilatations– und Erosionsoperatoren. Wie im folgenden deutlich wird, scheint eine Parametrisierung von vierzig Koeffizienten pro ASIC günstig zu sein. Sie können zur unabhängigen Berechnung zweier Operatoren mit kleinerer Matrix mit je bis zu zwanzig Koeffizienten oder eines einzigen Operators mit einer Matrix von bis zu vierzig Koeffizienten dienen, wobei über den zweiten Ausgang dann noch eine Linearkombination mit dem Zentralpixel ausgegeben werden kann, etwa um gleichzeitig Gauß– und Laplacepyramiden, die sich nur in der Gewichtung des Zentralpixels unterscheiden, berechnen zu können. Im Verbund mit weiteren ASICs kann jedes ASIC auch vierzig Koeffizienten eines Operators beliebiger Größe realisieren. Eine $9*9$ Operatormatrix ist so mit zwei ASICs realisierbar (der 81. Koeffizient kann vom letzten ASIC als Zentralpixel hinzuverknüpft werden). Bei radialsymmetrischen Matrizen ist durch nullsetzen von Koeffizienten in jeder Ecke der Matrix und Nutzung des Zentralpixels eine $11 * 11$ Matrix mit zwei ASICs realisierbar (s. Abb. 6.7).

6.1.2 Rechengenauigkeit

Betrachtet man die Faltung als mathematisches Problem ganzzahliger Werte, lassen sich beliebig genaue Koeffizienten fordern. Werden etwa gleichgroße positive und negative Koeffizienten einer Matrix mit dem gleichen Pixelwert multipliziert und zu null aufsummiert, kann auch ein im Vergleich zu ihnen beliebig kleiner Koeffizient noch die Summe dominieren. Die Pixelwerte von Originalbildern sind jedoch durch die AD–Wandlung oder die Ausgabeformatierung anderer Verarbeitungsstufen zuvor aus dem analogen in einen diskreten Wertebereich abgebildet worden. Die Differenz dabei macht den Quantisierungsfehler aus, der sich in einem der Bildinformation überlagerten Rauschen äußert. Die analogen Werte, die dabei auf ein und demselben diskreten Abtastwert abgebildet werden, streuen gleichverteilt in einem Intervall mit der Breite der Abtastauflösung um den Abtastwert. Für die Koeffizienten und die folgende Verarbeitung lassen sich nun relative Genauigkeiten angeben, bei deren Einhaltung die Streuung des Ergebnisses durch die Rechnung nicht stärker als durch das Quantisierungsrauschen der Eingabedaten verfälscht wird.

Welche Rechengenauigkeit bei der internen Berechnung und Gewichtung der Pixel sinnvoll ist, hängt von den Operatoren ab. Für Dilatations– und Erosionsoperatoren ist diese Frage schnell beantwortet, da das Ergebnis nur einen Selektionsprozeß darstellt. Dieselbe Genauigkeit, mit der die Pixel zum ASIC übertragen werden, genügt auch für die internen Berechnungen. Bei Faltungsoperationen liegt der Fall komplizierter. Dadurch, daß mehrere Pixel zum Ergebnis beitragen, verringert sich statistisch die Streuung des Quantisierungsrauschens des Ergebnisses, was zu einer erhöhten Genauigkeit des Ergebnisses im Vergleich zu den Werten der einzelnen Eingabepixel führt und eine interne Verarbeitung mit erhöhter Genauigkeit rechtfertigt.

Volle Ergebniswortbreite

Eine hohe interne Rechengenauigkeit kann besonders gefragt sein, wenn viele Koeffizienten betragsmäßig gleichstark gewichtet zum Ergebnis beitragen, was vor allem bei größeren Matrizen der Fall ist, deren Koeffizienten auf mehrere kaskadierte ASICs verteilt sind. Schon bei der Multiplikation mit dem Koeffizienten erreicht man als Wortbreite des Ergebnisses die Summe der Wortbreiten des Eingabepixels und des Koeffizienten, obwohl das Ergebnis nur dieselbe relative Genauigkeit wie das Eingabepixel besitzt. Um die Summe solch breiter Datenworte mit jedem Pixeltakt zwischen ASICs zu übertragen, sind jedoch viele Anschlußpins nötig, die teurere Gehäusebauformen und einen höheren Verdrahtungsaufwand zwischen den ASICs erfordern. 8–Bit quantisierte Pixel und 256 ebensobreite Koeffizienten führen bei Faltungen bereits zu Wortbreiten von 24 Bit, die nur aufwendig zu übertragen sind. Bei 8–Bit breiter Ausgabe stellt sich die Frage, wie genau oder mit welcher Wortbreite quantisiert die internen Teilergebnisse überhaupt berechnet und zwischen kaskadierten ASICs übertragen werden müssen. Wird das interne Ergebnis stets mit der vollen Dynamik ausgegeben — was bedeutet, daß auch die möglichen Extremwerte noch im Ausgabeintervall liegen — wird von dem Ergebnis der internen Berechnung auch nur 8–Bit Genauigkeit gefordert werden müssen. Dies ist jedoch nicht der Fall, wenn die Extrema mit an Sicherheit grenzender Wahrscheinlichkeit nie auftreten werden. Dann kann das Intervall, in dem überhaupt mit dem Ergebnis zu rechnen ist, vergrößert ausgegeben werden, und man könnte auch von einer erhöhten relativen Genauigkeit des internen Ergebnisses profitieren.

Verteilung von Ergebniswerten

Ob mit den theoretisch möglichen Ergebniswertextrema überhaupt zu rechnen ist, hängt von den Eingabepixeln ab. Die Verteilung der Faltungssumme kann als Dichte einer Zufallsvariable betrachtet werden, die sich aus der Faltung der Wert–Verteilungen der einzelnen Pixel aus der Operatorumgebung mit den ihnen zugeordneten Koeffizienten ergibt. Die Verteilung der Summe sehr vieler, linear unabhängiger Zufallsvariablen führt als Grenzwert zu Gaußverteilungen [*Jähne 1991*, S. 77], die sich durch Mittelwert und Varianz charakterisieren lassen. Es ergeben sich schon bei einer geringen Anzahl von Zufallsvariablen ähnlicher Varianz Gaußverteilungen mit einer sehr geringen Streuung. Das Ergebnis einer Faltung kann also immer dann, wenn die lineare Unabhängigkeit ihrer Eingabewerte erfüllt ist durch Streckung des besetzten Ausgabeintervalls genauer ausgegeben werden, als die Werte selbst eingegeben wurden. Die den Eingabepixeln linear überlagerten Quantisierungsfehler erfüllen die Bedingung, linear unabhängig zu sein. Die Streuung ihrer Summe, die das von den Eingabpixeln fortgepflanzte Quantisierungsrauschen des Ergebnisses ausmacht, ist daher viel geringer als das Quantisierungsrauschen der Eingabepixel — das Ergebnis liegt damit im ASIC mit erhöhter Genauigkeit vor. In welchem Falle diese auch bei der Ausgabe genutzt werden kann, wird im folgenden geklärt.

Bei hinreichend linearer Unabhängigkeit der Eingabepixel selbst kann die Streuung des Ergebnisses so klein werden, daß auch bei Übertragung eines eingeschränkten Ergebnisintervalls symmetrisch um den Ergebniserwartungswert und mit der vielfachen Breite der Ergebnisstreuung die Wahrscheinlichkeit von Überläufen praktisch null wird. Allerdings sind die Verteilungen von Pixeln einer Umgebung in folgender Hinsicht fast immer linear abhängig. Bei Bildern mit geringem hochfrequenten Spektralanteilen, wie sie meistens vorkommen, sind die Grauwerte benachbarter Pixel stark korreliert. Ob dies zu einer breiten Ergebnisverteilung führt, hängt vom Operator ab. Ergebnisse von Binomialoperatoren (Tiefpaß) streuen aufgrund der Korrelation beispielweise ebenso stark, wie die lokale Bildhelligkeit. Ergebnisse von Differentialoperatoren (Hochpaß) sind dagegen, unabhängig von der lokalen Bildhelligkeit, dicht um das Ergebnis null verteilt.

Neben dieser Korrelation in den Bildern, treten Verteilungen von Pixeln einer Umgebung typischerweise genau dann auf charakteristische Weise korreliert auf, wenn ein visuelles Merkmal vorliegt, oder ein Objekt im Bild zu erkennen ist. Fällt die Faltungssumme in diesen Fällen nicht mit der Verteilung für sonstige Bildbereiche zusammen, kann man dies nutzen, um Objekte zu detektieren. Die Ergebnisse, gewichtet mit der Wahrscheinlichkeit ihres Auftretens, überlagern sich nun mit der eingangs erwähnten Gaußverteilung. Man erhält sie als Histogramm des Ergebnisbildes. Aus Histogrammen lassen sich geeignete lineare Ergebnistransformationen zur optimalen Ausnutzung der beschränkten Ausgabewortbreite durch tatsächlich vorkommende oder benötigte Ergebnisse abschätzen.

Reduzierte Ergebniswortbreiten

Für die quantitative Betrachtung eines typischen *worst–case* Falls zur Dimensionierung der geforderten Verarbeitungsgenauigkeit, sei hier von vier kaskadierten ASICs, die einen Faltungsoperator realisieren, ausgegangen. Die Koeffizienten von Operatoren, deren Umgebung aus mathematischer Sicht meist unbegrenzt ist, fallen zum Rand ihrer Umgebung soweit ab, daß alle außerhalb der Umgebung befindlichen Koeffizienten ohne signifikanten Fehler nullgesetzt werden können. Dominante Beiträge liefern also nur die zentral gelegenen Koeffizienten. Von den 160 Koeffizienten, die die vier ASICs enthalten, mögen größenordnungsmäßig etwa nur 64 die Summe gleichstark dominieren, die restlichen spielen bei dem Überlagerungsprozeß der Einzelfehler keine signifikante Rolle. Die 64 Koeffizienten sind durch Multiplikation mit den Eingabepixeln mit einem relativen Quantisierungsfehler von maximal 2^{-9} behaftet. Die als Wurzel der Varianz definierte Streuung des im Intervall (a,b) gleichverteilten Quantisierungsfehlers, beträgt $\sqrt{(b-a)^2/12}$ und ist somit um den Faktor $\sqrt{3}$ geringer als der Quantisierungsfehler $(b-a)/2$ [*Hübner 1990*], also in der Größenordnung 2^{-10}. Durch die Mittelung der 64 Pixel der Operatorumgebung verringert sich der Quantisierungsfehler des Ergebnisses. Der Vorgang ist mit mehrfachem Messen und Mitteln der Ergebnisse in Experimenten zur Genauigkeitssteigerung vergleichbar: Wir wollen hier den gleichverteilten Quantisierungsfehler durch einen normalverteilten mit demselben Erwartungswert und derselben Varianz ersetzen, da bei deren Faltungen die Auswirkungen auf die Varianzen trivial, nämlich quadratisch gewichtet additiv, sind. Die Streuung des Fehlers der Summe nimmt dann um den Faktor $\sqrt{64 * (1/64)^2} = 8$ ab. Dabei sollte noch erwähnt werden, daß im Gegensatz zum vereinfachten normalverteilten Modell hier ein maximaler Quantisierungsfehler angegeben werden kann. Mathematisch exakt müßte die Varianz und Dichte einer 64–fach gefalteten Rechteckverteilung berechnet werden. Da die Varianz jedoch nur abzuschätzen ist, die Extremwerte der Verteilungen mit dem 64–fachen Quantisierungsfehler trivialerweise bekannt sind und die Verteilungen der Summen beliebiger unabhängiger Zufallsvariablen bekanntermaßen schnell gegen Normalverteilungen konvergieren, kann hier darauf verzichtet werden. Der Quantisierungsfehler der Eingabedaten äußert sich bei den Ausgabedaten also in einer 1/8 so großen Streuung von etwa 2^{-13}. 11–Bit quantisierte Daten haben etwa dieselbe Streuung, zusammen mit dem Vorzeichenbit enthalten also 12 Bit des Ergebniswertes signifikante Daten, die niederwertigeren Bits gehen im Quantisierungrauschen der Eingabepixel unter. Durch die angebotenen arithmetischen Shiftoperationen zum Skalierung von Zwischenergebnissen ist jedoch eine volle Ausnutzung des 12–Bit Wertebereichs durch das Ergebnis nicht zu erreichen, so daß 13 signifikante Bits für das Ergebnis zu veranschlagen sind. Da nun bei allen vier ASICs die internen Ergebnisse auf die gesuchte Zwischensummenwortbreite gerundet werden — bei den zwei mittleren ASICs, bei denen die Summe der Verknüpfung externer und interner Daten ein Bit breiter geworden ist, sogar zweimal — bietet sich eine Übertragung mit der doppelten Datenbreite der Pixel, also mit 16–Bit Wortbreite an. Damit kann die von den Bilddaten her maximal zu erwartende Genauigkeit des Ergebnisses bei jedem Operator für bis zu vier kaskadierten ASICs ohne signifikanten Verlust ausgeschöpft werden.

Koeffizientenmultiplikation

Neben dem Quantisierungsrauschen der Pixel selbst trägt auch die Quantisierung der Koeffizienten zu Fehlern bei. Man kann sich die Koeffizienten als rationale Zahlen in Festkommadarstellung im Wertebereichs $[-1, 1[$, also mit führendem Komma vorstellen, die mit Pixeln multipliziert wiederum zu rationalen Zahlen im Festkommaformat führen. Dies erlaubt den Begriff des Rundens bzw. Abschneidens von Nachkommastellen anzuwenden. Spricht man dagegen bei ganzzahliger Darstellungen vom Abschneiden niederwertige Stellen, führt dies zusätzlich zu einer unbeabsichtigten Shiftoperation des verbliebenen Wertes, was eine zusätzliche Interpretation des Ergebnisses erfordert. Quantisierungsfehler der Koeffizientenmatrix unterscheiden sich von solchen der Pixelquantisierung darin, daß sie zu systematischen Fehlern, nämlich schlicht falschen Operatoren, führen. Um solche Fehler gegenüber den nicht–systematischen streuenden Fehlern klein zu halten, wurde die Koeffizientenquantisierung hier ein Bit größer als die der Pixel gewählt. Zusammen mit dem Vorzeichenbit ergibt sich für die Koeffizienten so eine 10–Bit Integerdarstellung im 2–er Komplement, die zu Quantisierungsfehlern der Beträge von 2^{-10} führen.

Neben den Quantisierungsfehlern der Eingabedaten, der Pixel und der Faltungsmatrix, kommen noch diejenigen hinzu, die eventuell bei der weiteren Verarbeitung bis zum Endergebnis, der Faltungssumme, begangen werden. Um den Benutzer nicht mit diesen unerwarteten Fehlerquellen zu konfrontieren, rechnen viele existente Systeme intern mit voller Präzision. Die Multiplikation liefert dann ein Ergebnis mit der Wortbreite der Summe der Wortbreiten des Multiplikanten und des Multiplikators, also des Pixels und des Koeffizienten. In unserem Fall wären das 18–Bit Integerwerte. Bei der Summenbildung aller 40 gewichteten Pixel pro ASIC kommt man dann auf 24–Bit Werte. Erst bei der Ausgabe erfolgt dann eine Abbildung der internen 24–Bit Darstellung auf die 8–Bit Ausgabe. Der Vorteil liegt darin, daß zum Beispiel die Anwendungen der Faltungsmatrizen $2^{-1} * (-1, 0, 1)$ und $2^{-9} * (-256, 0, 256)$ dasselbe Ergebnis liefern, der Nachteil im — vom eigentlichen Problem her ungerechtfertigten — Berechnungsaufwand.

Einsparung bei der Multiplikation

Das hier entwickelte ASIC bietet je nach Parametrisierung die Möglichkeit, eine beliebige Anzahl niederwertigster Bits des Produkts gar nicht erst zu berechnen und dafür entsprechend viel Logik pro Koeffizientenzelle einzusparen. Die hier instantiierte Parametrisierung liefert als Ergebnis des Produkts des 8–Bit Pixels mit dem 10–Bit Integerkoeffizienten ebenfalls ein 10–Bit Integerergebnis. Die sechs niederwertigsten Stellen werden unterdrückt, und die folgenden zwei nur berechnet, um einen Rundungsfehler des Ergebnisses von höchstens einem Bit garantieren zu können. Der Nachteil dabei besteht darin, darauf achten zu müssen, daß die Koeffizienten ihren Wertebereich voll ausschöpfen und Rundungsfehler des Ergebnisses akzeptieren zu müssen, die allerdings kleiner als die unvermeidlichen Quantisierungsfehler sind. Die erste der im vorherigen Abschnitt genannten Faltungsmatrizen ist absolut ungeeignet, da die Pixelwerte nach der Multiplikation genau die abgeschnittenen acht niederwertigsten Bits einnehmen; das Ergebnis wäre konstant null. Die zweite Matrize würde hier sogar zum exakten mathematischen Ergebnis führen, da alle acht abgeschnittenen niederwertigsten Bits konstant null sind. An Volladdierern benötigt man für die 8 und 9 Bit (excl. Vorzeichen) quantisierten Produktterme statt 46 $(0+1+2+\ ...\ +6+7+7+6+\ ...\ +2+1+0)$ nun 15 $(5+4+3+2+1+0)$ Volladdierer weniger, also 31 (vgl. Abb. 6.1 und Abschnitt 6.1.3, Rekursive Multiplikationszerlegung). Die Berechnung der Summe erfolgt nun durch 39 Additionen von 10–Bit statt 18–Bit Integers, was nur 425 Volladdierer (20 * 10–Bit, 10 * 11–Bit, 5 * 12–Bit, 2 * 13–Bit, 14– und 15–Bit Addierer) statt 737 (39 * 8–Bit breitere Addierer) erfordert. Bei 40 Koeffizienten werden also nur 1665 gegenüber 2577 Volladdierern benötigt. Diese Komponenten dominieren den ASIC–Flächenbedarf, da der Zeilenspeicher nicht *on–chip* realisiert ist, so daß die Einsparungsmöglichkeiten den Komfortverlust bei der Koeffizienteneingabe für eine Matrix und geringe Rundungsfehler nach Meinung des Autors rechtfertigen. (Das ASIC ist auch auf volle Rechengenauigkeit um–parametrisierbar.)

Als letzte Einsparungsmöglichkeit wäre eine Reduktion der Wortbreite bei der Summation zum Endergebnis selbst in Erwägung zu ziehen. Mit der Summation von jeweils vier Summanden steigt die Wortbreite des Ergebnisses um je zwei Bit. Die Streuung des Quantisierungsfehlers wird jedoch bestenfalls um $\sqrt{2}$ geringer, d. h. die relative Genauigkeit steigt höchstens um ein Bit, trotzdem ergibt sich kein Einsparungspotential. Da z. B. bei einer Matrix mit nur einem Koeffizienten die höherwertigen Bits gar nicht genutzt werden, bleibt relativ zum genutzten Wertebereich das niederwertigste Bit immer signifikant (selbst wenn die relative Genauigkeit nicht bei jeder zweiten Addiererstufe um ein Bit zunähme). Das häufige Runden würde außerdem zu signifikanten Fehlern führen. Bei der aktuellen Parametrisierung des Designs liefert jeder Koeffizientenmultiplizierer 10–Bit signed–Werte. Auf dem Chip werden 40 solcher Werte zu einem 16–Bit Integer aufaddiert. Mit einem Offset der vollen Wortbreite ist eine Verschiebung von unsigned–Werten zu mittelwertfreien Ergebnissen oder umgekehrt möglich. Intern kennt das ASIC zwar nur positive Pixel, mit der LUT und dem Offset sind jedoch auch vorzeichenbehaftete Pixelwerte durch eine *displaced*–Darstellung der Pixel und eine matrixabhängige Offsetkorrektur möglich. Da ab der Offsetaddition bis zur Ausgabe intern nur im *signed*–Format gerechnet wird, wird eine 17–Bit Darstellung benötigt, um auch den 16–Bit *unsigned*–Bereich voll abzudecken.

Kaskadierungsschleife

Zur Kaskadierung von ASICs und auch zur Ausgabe wird der Wert mit einer Genauigkeit von 16 Bit weitergereicht. Bei der Kaskadierung muß der Wert, um interne Überläufe auszuschließen, auch noch heruntergeteilt werden können (s. u.). Ein arithmetischer Shifter, der die Summe arithmetisch nach rechts schieben und somit durch beliebige Zweierpotenzen teilen kann, erfüllt beide Aufgaben. (Der Shifter ermöglicht zudem die Entkopplung der Wortbreiten der Faltungssumme von der Pixelwortbreite und sichert damit eine unabhängige Wahlmöglichkeit der ASIC–Parameter). Pro Pixeltakt muß je ein Wert vom Nachbar–ASIC gelesen und einige Takt später, mit dem eigenen Ergebnis verknüpft, an das nächste ASIC weitergeben werden. Dies geschieht zum Einsparen von Gehäusepins im *time–division–multiplex* Verfahren. Zuerst wird das *high*–Byte, einen halben Pixeltakt später das *low*–Byte ausgegeben. Bei kaskadierten ASICs muß der Wertebereich so dimensioniert werden, daß bei Kaskadierungsverknüpfungen keine Überläufe stattfinden können. Denn Clippen von Teilsummen führt natürlich zu unsinnigen Ergebnissen. Obwohl sie intern geclippt wurden, liegt der Ergebniswert der Kaskade u. U. mitten im Wertebereich. Um den Wertebereich der Kaskadierungsschnittstelle voll ausnutzen zu können, sollte der Offset so gewählt werden, daß der Mittelwert der Teilsumme null beträgt. Mit einem zweiten Offset direkt vor der Ergebnisausgabe kann unabhängig von dem Integerformat zwischen den ASICs wieder eine vorzeichenlose Formatierung erreicht werden.

Die Ausgabeschnittstelle ist wegen ihrer geringen Wortbreite besonders kritisch. Es sollte auf jeden Fall ihr voller Wertebereich genutzt werden. Dazu dient vor dem Clippen auf die Ausgabewortbreite ein zweiter Offset, ein Shifter und ein Multiplizierer. Das Clippen kann wahlweise auf vorzeichenbehaftetes oder vorzeichenloses Format erfolgen. Im Notfall kann durch Nachbau des Kaskadierungsempfängers eine Ausgabe mit doppelter Wortbreite erfolgen, die folgende ASICs aber nicht mehr weiterverarbeiten könnten. Da dies nicht besonders sinnvoll erscheint, wurde auf eine parallele Ausgabemöglichkeit über beide Ausgabeports verzichtet.

Als Ergebnis dieser Überlegungen arbeitet das ASIC in der aktuellen Parametrisierung mit 8–Bit quantisierten Pixeldaten und 10–Bit breiten Koeffizienten in Zweierkomplementdarstellung. Das Ergebnis eines mit einem Koeffizienten gewichteten Pixels hat 10–Bit Breite. Die Summation der einzelnen Ergebnisse erfolgt mit voller Genauigkeit. Zur Kaskadierung werden Teilsummen mit 16–Bit Genauigkeit im Zweierkomplement zwischen den ASICs übergeben, die Ausgabewortbreite des Endergebnis beträgt wieder 8–Bit.

6.1.3 Multiplizierer

Die Implementation des im vorherigen Abschnitts angesprochenen Multiplizierers mit reduzierter Ergebniswortbreite erfolgte als rekursive Komponente. Dabei sind einige Annahmen über die Tiefe der Rekursion nötig. Da mit der Parametrisierung auch die konkreten Instantiierungen der rekursiven Komponente variieren, wird es notwendig, die zur korrekten Rekursionsauflösung benötigten Aussagen für alle Parametrisierungen zu beweisen.

Die Komponente *product* realisiert eine Multiplikation zwischen zwei beliebig breiten Bitvektoren. Zur Veranschaulichung sei an die Methode des schriftlichen Multiplizierens erinnert. Dabei müssen allerdings mindestens soviele Zwischenwerte aufaddiert werden, wie der weniger breite Faktor Stellen hat. Die Anzahl der Addierer ist konstant, da jeder die Anzahl der noch aufzusummierenden Summanden genau um einen reduziert. Statt seriell aufzuaddieren, können die Addierer auch in Form eines Binärbaums angeordnet werden. Dies reduziert den längsten Datenweg eines Summanden und damit den Additionszeitbedarf von linearer auf logarithmische Ordnung.

Addiererarchitekturen

Additionen mit n–Bit *ripple–carry*–Addierern benötigen im *worst–case*, in dem jeder Volladdierer einen *carry* generiert, die n–fache *carry*–Laufzeit. Damit können sie zum kritischen Pfad im ASIC werden, der die längste Gatterlaufzeit zwischen Registern aufweist und die

Geschwindigkeit des ASIC begrenzt. Einzelne *ripple–carry* Addierer mit 1.2 ns *carry–* und 4 ns sonstiger (*worst–case*) Signallaufzeit jedes Volladdierers und maximal 18 Bit Wortbreite *on–chip* bzw. 12 Bit in der gewählten Multipliziererinstantiierung sind bei 67 Nanosekunden Zykluszeit der Pixelclock (14.75 MHz) absolut unkritisch. Zwar ist eine Verschaltung aller Addierer in Form eines Binärbaums mit einer asynchronen Ergebnisverzögerung kleiner einem Taktzyklus nicht zu realisieren, eine Pipelinestruktur mit je einem Register nach jedem Addiererausgang ist jedoch problemlos möglich.

Wäre dem nicht so, ließen sich *ripple–carry*–Addierer durch Einfügen von Registern selbst als Pipeline von Volladdierern aufbauen. Das n+1.te Bit der Summe wird erst einen Takt später als das n.te berechnet, dazu sind die n+1.ten Bits der Summanden und das *carry* der Addition der n.ten Bits einen Takt verzögert anzuliefern. Ein Aufeinanderfolgen von Summierern dieser Art ist besonders günstig, da die Ergebnisse der Vorstufe in der gleichen zeitlichen Reihenfolge anfallen, wie sie die Folgestufe benötigt. Der Mehraufwand fällt an den Grenzen solcher Summierbäume an: Das erste bis zum $n-1$.ten Bit des Ergebnisses muß um $n{-}1$ bis zu einem Takt verzögert werden, was $sum(n = 1 \dots n)(n{-}1) = n{*}(n{-}1)/2$ Register erfordert. Leider ist der Aufwand beim Anliefern der Summanden ebensogroß, nur daß hier das MSB maximal verzögert wird und das LSB unverzögert anliegen kann (*most/ least significant bit*).

Ein weiterer Weg zur effektiven Laufzeitreduktion wäre es, *ripple–carry–* Addierer mit $2 * n$ Volladdierern aus n zwei Bit breiten *carry–look–ahead–* Addieren aufzubauen. Damit ließe sich die Laufzeit halbieren, bei drei Bit breiten *carry–look–ahead*–Addierern dritteln usw.. Dies läßt sich durch Austauschen der Standard–Addiererkomponenten auch ohne Änderung im VHDL–Code selbst realisieren. Im ASIC–Entwurf werden nur *ripple–carry–* Addierer eingesetzt.

Rekursive Multiplikationszerlegung

Eine Multiplikation läßt sich folgendermaßen in zwei Multiplikationen mit reduzierter Wortbreite des Multiplikators zerlegen (man beachte dabei, daß die Ganzzahldivision in VHDL den ganzzahligen Anteil der Division zurückgibt, es gilt $n/2 + (n + 1)/2 = n$). Mit $p = (m + 1)/2$ gilt

$$product(n + m - 1 \dots 0) = multiplikant(n - 1 \dots 0) * multiplikator(m - 1 \dots 0)$$

$$\begin{aligned} product(n + m - 1 \dots 0) = {} & \\ & multiplikant(n - 1 \dots 0) \; * \; multiplikator(p - 1 \dots 0) \; + \\ & multiplikant(n - 1 \dots 0) \; * \; multiplikator(m - 1 \dots p) \; * \; 2^{(n-p)} \end{aligned}$$

Der Multiplikator des letzten Produktausdrucks hat dann die Breite $m - p = m/2$, der des ersten $p = (m + 1)/2$. Ist m eine Zweierpotenz, halbiert sich mit jeder Zerlegung des Multiplikators dessen Wortbreite. Nach $ld(m)$ Schritten ist dann die ursprüngliche Multiplikation auf m triviale Multiplikationen mit je einem 1–Bit Multiplikator zurückgeführt. Diese Zerlegung läßt sich auf einen Binärbaum mit beschrifteten Knoten abbilden. Die

Menge der Beschriftungen auf dem n.ten Niveau sei b(n). Der Wortbreite des unzerlegten Multiplikators bildet die Beschriftung der Wurzel. Jeder Knoten steht für eine Zerlegung des Multiplikators, die Wortbreiten der beiden Hälften des zerlegten Multiplikators beschriften jeweils seine Söhne im Binärbaum (s. Abb. C.1).

Die Zerlegung ist beliebig fortsetzbar. Wenn jedoch nur noch mit 0 und 1 bezeichnete Knoten vorhanden sind, sind alle folgenden Niveaus, wenn alle Knoten mit der Bezeichnung 0 im Graph entfernt werden, identisch. Alle Knoten mit den Bezeichner 1 stehen für 1 Bit breite Multiplikatoren, mit denen die Zerlegung der Multiplikation endet, denn 0 Bit breite Multiplikatoren steuern zum Ergebnis nichts bei. Für den Beweis folgender Aussage war es nötig, den beschrifteten Graph einzuführen (Sie ist wichtig, um die korrekte Rekursionsauflösung der Multiplikationsimplementation unabhängig von der Parametrisierung zu beweisen): Wenn der Multiplikator n Bit breit ist, enthält das $ceiling(ld(n))$.te Niveau nur noch Knoten mit den Bezeichnern 0 und 1. Alle weiteren Knoten mit dem Bezeichner 1 liegen ein Niveau höher bzw. es sind keine weiteren vorhanden. Zu beweisen ist dazu $b(n) \in \{\{0,1\},\{1\}\} : n >= ceiling(ld(m))$ wobei m für die Wortbreite des Multiplikators steht. Der Beweis ist im Anhang C aufgeführt. Die Bedeutung dieser Aussage liegt darin, daß alle Niveaus des Baumes, die Einsen enthalten, die Eingabe des Multiplikanten und Multiplikators benötigen. Bei einer Pipelinerealisierung, in der die Berechnung jedes Niveaus einer Pipelinestufe entspricht, muß die Eingabe auf näher zur Wurzel gelegenen Niveaus jedoch verzögert werden, damit das Zwischenergebnis zum richtigen und nicht einem vorherigen Ergebnis beiträgt. Daß es dazu nur einer Verzögerung um einen Takt bedarf ("Einsen" können nur in den beiden untersten Stufen des Baumes auftreten), galt es zu beweisen.

Die Zerlegung kann in einen rekursiven Komponentenaufruf umgesetzt werden. Jede Instantiierung testet, ob sie die triviale Multiplikation auszuführen hat, die durch Zurückgeben des Multiplikanten oder von 0, je nachdem ob das Multiplikatorbit gesetzt ist oder nicht, realisiert ist. Ansonsten instantiiert die Komponente zwei Subkomponenten mit den beiden Multiplikatorhälften und übernimmt nach jeder positiven *clock*–Flanke deren Ergebnisse in ihre Pipelineregister. Deren stellenrichtig addierte Summe reicht sie dann als ihr Ergebnis nach oben weiter. Dies erfordert eine Sonderbehandlung der auf dem Niveau $n-1$ mit 1 bezeichneten Knoten ($n = ceiling(ld(g))$), da sie aufgrund des um einen Knoten kürzeren Weges zur Wurzel einen Takt vorauseilen und ohne weitere Berücksichtigung ihr Summand fälschlicherweise zur vorher durch die Pipeline berechneten Multiplikation addiert würde. Eine Verzögerung der Weitergabe der trivialen Multiplikationsergebnisse solcher Knoten um einen Takt ist elegant zu formulieren, erfordert jedoch u. U. ein vielfaches Zwischenspeichern des mit 0 oder 1 multiplizierten Multiplikanten. Praktischer ist es, sowohl den Multiplikanten, als auch den Multiplikator zusätzlich um einen Takt verzögert zur Verfügung zu stellen und ggf. diese Werte zu benutzen. Der doppelte Aufwand in den Portlisten verschwindet zur Compilezeit, wenn durch die Parameter der VHDL–Beschreibung (generics) die Wortbreiten statisch festgelegt wurden. Unbenutzte Zweige werden nicht realisiert. Bei Wortbreiten des Multiplikators, die Zweierpotenzen entsprechen, entfallen sogar die Verzögerungsregister, da ihre Ergebnisse nirgends benötigt werden. Der Beweis im Anhang dient dazu, nachzuweisen, daß in jedem Fall Multiplikant und Multiplikator nur um einen einzigen Takt verzögert benötigt werden.

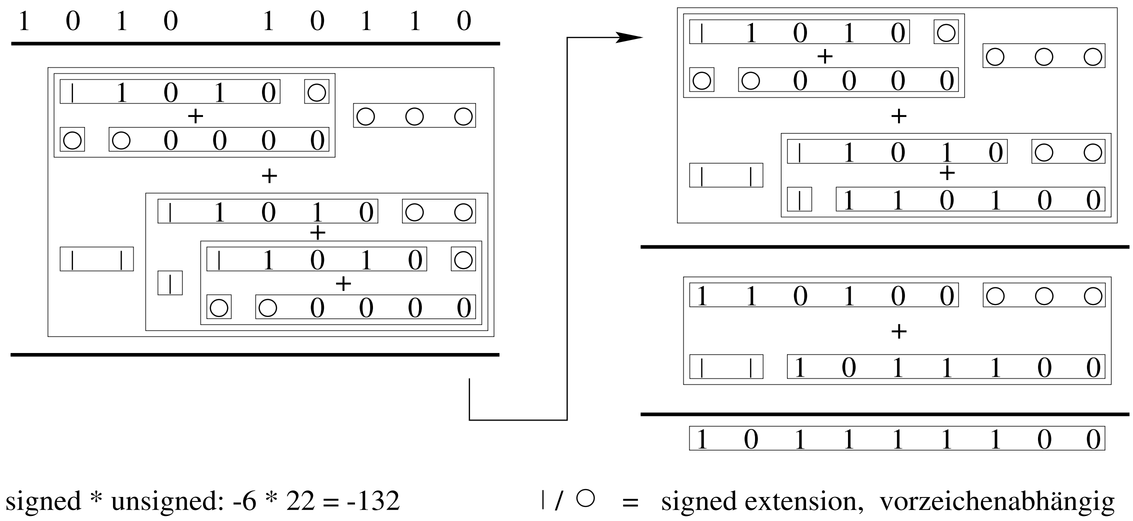


Abbildung 6.1: Beispiel einer Multiplikation

Um die Zerlegung zu verdeutlichen, ist in Abb. 6.1 eine Multiplikation in ihren zeitlichen Stadien gezeigt. Mit jedem Takt wird die Addition der innersten Kästen durchgeführt, deren Wert im nächsten Schritt, statt des die Addition umrahmenden Kastens, erscheint. Die sign–extension Bits sind hier zur Unterscheidung stilisiert dargestellt (s. Legende). Sie beziehen sich immer auf den gesamten Kasten rechts von ihnen, bei dessen Auswertung sie auch erst generiert werden, nicht auf Werte innerhalb der Kästen. (Bei Parametrisierung für *unsigned*–Multiplikanten werden alle *signed*–extension Bits nullgesetzt.) Die Berechnungen starten so, daß sie zeitgleich zum Ergebnis führen. Gemäß dem Beweis muß die Auswertung dazu höchstens in zwei aufeinanderfolgenden Schritten gestartet werden.

In der aktuellen Parametrisierung werden die letzten 6 Bit nicht berechnet. Dies kommt einer Summation nach Abdecken der 6 niederwertigsten Stellen der Zwischenergebniszeilen gleich (nur von der untersten werden 6 Bit an Eingabedaten gekappt, von der zweituntersten noch 5 Bit usw.). Im Vergleich zum nachträglichen Abschneiden des Ergebnisses fällt auf, daß dabei auch die Überträge der niederwertigsten Stelle verlorengehen. Im ungünstigsten Fall wurden dabei die Werte $1/2, 3/4, 7/8, \ldots, 63/64$, also, bis auf die geometrische Reihe $1/2^n$, deren Summe gegen eins geht, jeweils ein niederwertigstes Bit des Restergebnis unterschlagen. Der Fehler beträgt also maximal -5 Digits, weswegen die beiden niederwertigsten Bits, die durch diesen Fehler absolut unsicher sind, nicht zur weiteren Rechnung benutzt werden. Der Fehler aufgrund des Unterdrückens der niederwertigsten Stellen der zur weiteren Berechnung benutzten Zahl beträgt damit 1 Bit, was tragbar erscheint.

6.2 Verzögerungsarchitekturen

Bei dem realisierten Entwurf war es möglich, alle Berechnungseinheiten parallel auf einem ASIC zu implementieren. Dazu müssen alle für die Rechnung benötigten Pixelwerte gleichzeitig zur Verfügung stehen. Dies entspricht im Wesentlichen der Aufgabe sie mehrfach verschieden lange zu verzögern. Eine Verzögerung um wenige Takte ist dabei problemlos mit Schieberegisterketten möglich, längeren Verzögerungszeiten können vorteilhafter auf andere Weise realisiert werden.

Verzögerungsstrategie

Zur Faltung werden Pixel einer lokalen Umgebung mit verschiedenen Koeffizienten multipliziert und aufsummiert. Das Ergebnis steht erst fest, wenn der zuletzt vom Pixelstrom gelieferte Pixel vorliegt. Alle anderen Pixel kann man zeitgleich zur Verfügung stellen, indem sie in Schieberegistern individuell verzögert werden, so daß sie zeitgleich mit dem zuletzt gelieferten Pixel am Ende der Schieberegister anliegen. Nun kann man alle Pixel mit ihren Koeffizienten multiplizieren und aufsummieren.

Alternativ kann man jedes aktuelle Pixel zunächst mit allen in der Matrix vorkommenden Koeffizienten multiplizieren. Die Ergebnisse sind Summanden verschiedener Faltungen. Alle Summanden außer dem Produkt des letzten Koeffizienten der Matrix sind allerdings zu früh berechnet worden. Die Ergebnisse müssen so verzögert werden, daß alle zu einer Faltung gehörenden Summanden zeitgleich eintreffen. Strukturell ähneln sich beide Vorgehensweisen sehr; einmal liegen die Verzögerungselemente jedoch vor, das andere mal hinter den Koeffizienten–Multiplizierern. Betrachtet man jedoch die Möglichkeiten die Architektur zu vereinfachen, so führen beide Vorgehensweisen zu unterschiedlich komplizierten Realisierungen.

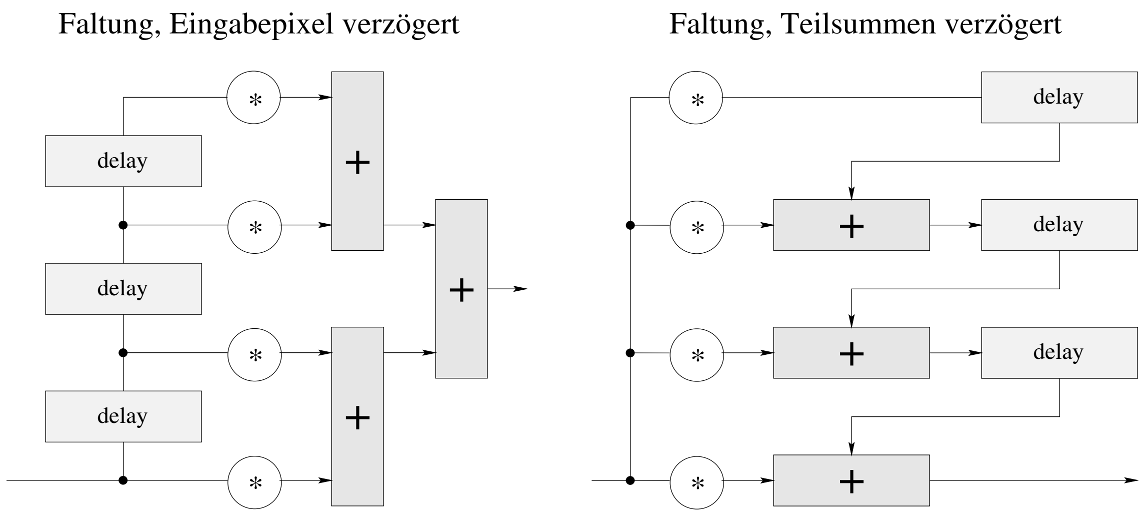


Abbildung 6.2: Verzögerungsstrategien

Die Verzögerung des aktuellen Pixels kann stark vereinfacht werden, da das längere Schieberegister jeweils die Verzögerung des kürzeren mitbenutzen kann. Dazu wird dessen Ausgang nur noch zusätzlich um die Differenz der Verzögerungszeiten verzögert. Rekursiv angewandt kommt man zu einer Schieberegisterkette der maximal geforderten Länge, die dann zusätzlich zwischen den einzelnen Registern Abzweigungen besitzt, wenn die gerade erreichte Verzögerung benötigt wird. Die Verzögerung der bereits gewichteten Summanden der Faltungssumme läßt sich ebenso stark vereinfachen, wenn man die Summation zweier Summanden jeweils frühestmöglich ausführt: Wenn zwei Summanden unterschiedlich lange zu verzögern sind, verzögert man den länger zu verzögernden zuerst um die Differenz ihrer Verzögerungszeiten, summiert sie dann auf und verzögert danach ihre Summe um die restliche Zeit. Rekursiv angewandt kommt man zu einer derart modifizierten Schieberegisterkette, daß jeweils zwischen den Registern der Kette Addierer an den Stellen eingefügt sind, wo weitere Summanden zur Faltungssumme hinzuaddiert werden müssen (s. Abb. 6.2).

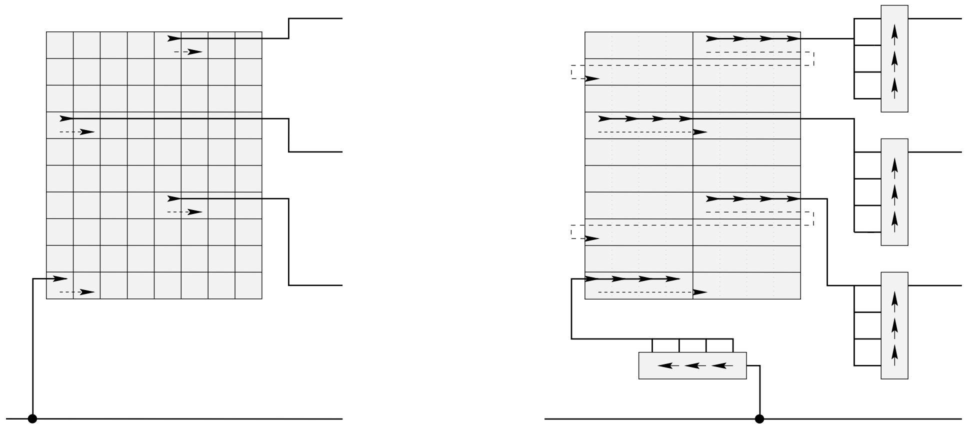
Hier wurde die erste Verzögerungsstrategie, die die Eingabepixel verzögert, für die zeilen– und spaltenweise Verzögerung innerhalb des ASICs gewählt, da die Wortbreite der Pixel kürzer ist als die der gewichteten Summanden. Das führt zu geringeren Kosten für die Pixel–Schieberegister. Die Zeilenspeicher wurden jedoch nicht als lange Schieberegister *on–chip* realisiert, sondern als externes statisches RAM. Da sie nur einen Datenbus besitzen, sind Speicher zwar komplizierter anzusteuern, aber auch deutlich kompakter. Das Produkt aus $Bildbreite * Matrixh\ddot{o}he$ bestimmt die Größe des benötigten Zeilenspeichers, der bei Nutzung von externen SRAMs unkritisch ist. Bei maximal 4096 Pixel Bildbreite und 4 * 16 KByte SRAMs können mindestens 16 Bildzeilen gespeichert werden. Als Adreßwortbreite wurden hier 14 Bit gewählt.

Bei größerem Verzögerungsbedarf können ASICs schon verzögerte Pixel, die sich über den Zentralpixelausgang ausgeben lassen (auch wenn es keine Zentralpixel sind) als Pixelstrom einlesen und zusätzlich verzögern.

Zur Zeilenverzögerung mit einem Speicher, wird bei der Methode der Pixelverzögerung der aktuelle Pixel einmal mit positiv fortlaufender Adresse geschrieben und mit festen negativen Adressoffsets für jeden Koeffizienten, der eine neue Matrixzeile beginnt, ausgelesen. Da selbst SRAMs nicht schnell genug sind, um alle Speicherzugriffe in einem Pixelclockzyklus zu tätigen, muß der Speicherdatenbus um ein vielfaches breiter ausgelegt werden. Jetzt können zeitgleich mehrere Pixel gelesen und geschrieben werden, so daß man Zeit hat, die Speicherzugriffe zum Schreiben und Lesen nacheinander zyklisch durchzuführen. Schieberegister an dem Eingang und den Ausgängen der Verzögerungseinheit zerlegen dazu den stetigen Eingabepixelstrom in Pixelpakete, speichern diese zwischen, lesen mehrere ältere Pakete wieder ein und generieren aus ihnen mehrere, verschieden lang verzögerte, stetige Ausgangspixelströme (s. Abb. 6.3).

Sollen stattdessen die Summanden verzögert werden, müßte das Ergebnis der Faltung von jeder Matrixzeile in einem read–modify–write Zyklus zu Adressen mit festen positiven Adressoffsets zu einer fortlaufenden positiven Referenzadresse hinzuverknüpft werden (nur

3-Zeilen-Bildspeicher



1 Schreib- & 3 Lesezyklen pro Pixel
4 Speicherzugriffe pro Pixelzyklus

1 Schreib- & 3 Lesezyklen alle 4 Pixel
Verzögerungsgranularität 4 Pixel

Abbildung 6.3: Zeilenspeicher im SRAM

der älteste Teilsummand überschreibt den Speicherinhalt jeweils). (Während man im ersten Fall den aktuellen Pixel direkt benutzt und gar nicht über Verzögerungselemente laufen läßt, ist es im zweiten Fall die letzte Summe, die man direkt verwendet, anstatt sie wieder zu verzögern.)

Bei der ASIC–Implementation wurde die einfachere Architektur der verzögerten Eingabepixel bzw. der Zeilenverzögerung für alle Verzögerungen innerhalb des ASICs gewählt. Nur bei der ASIC–Kaskadierung tritt die Verzögerung von Teilsummen zwischen den ASICs sowieso als parasitärer Effekt auf und wurde daher zur alternativen Verzögerungsart erweitert. Dies wird jedoch noch in Abschnitt 6.7 besprochen.

6.3 Faltungsrealisierung

Wird das Bild zeilenweise abgetastet und ohne Zeilenspeicherung verarbeitet, können von den $x * y$ Quellbilddaten bei der nächsten Faltung die jeweils $(x-1) * y$ weiter benötigten wiederverwendet werden. Pro Zielpixel werden lediglich y neue Quellbildpixel gebraucht, die restlichen rücken nur einen Platz weiter, wobei y Pixel in dieser Zeile nicht mehr benötigt werden. Dieser Effekt ist unabhängig von der Höhe y der Matrix. Für $y = 1$, eine eindimensionale Matrix, verdeutlicht dies Abb. 6.4. Lediglich der Wert des Pixel P_{i+6} wird für jede weitere Faltung neu benötigt, die anderen sind schon aus der vorherigen Faltung bekannt.

Wechselt man den Standpunkt von einer über feststehende Bilddaten verschobenen Matrix zu einer feststehenden Matrix mit Koeffizienten, an denen nun die Quellbilddaten

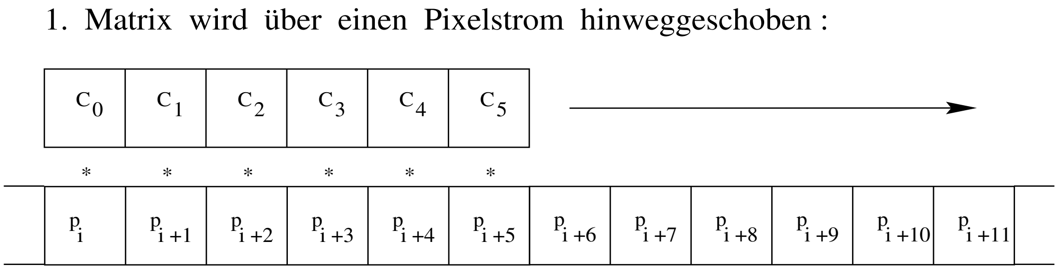


Abbildung 6.4: Koeffizienten über Bilddaten verschieben

vorbeigeschoben werden, scheint es, als wenn an den festen Koeffizienten die Pixel nun in umgekehrter Richtung vorbeigeschoben werden. Ein Vergleich der Faltungsprodukte (6.4 und 6.5) beweist, daß die Ergebnisse weiterhin identisch sind.

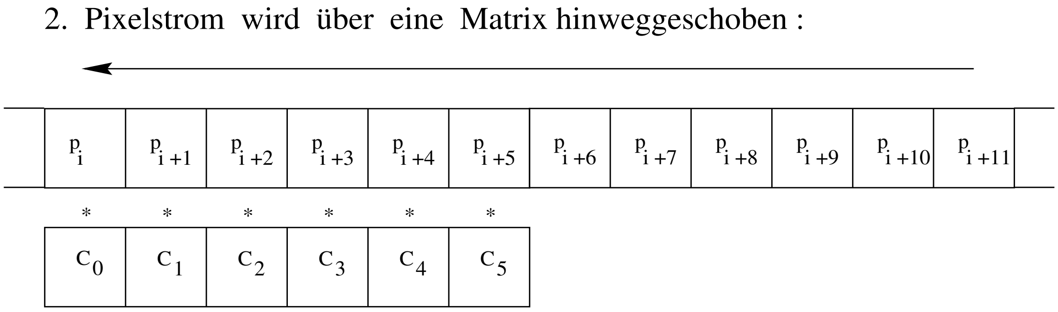


Abbildung 6.5: Verschiebung von Bilddaten über Koeffizienten

Diese Darstellung deutet auf die Realisierungsmöglichkeit der Faltung durch ein systolisches Array hin (s. Abb. 6.6). Eine entsprechende Hardwarerealisierung sieht y Schieberegister der Länge $x-1$ vor, die die zeilenweise verzögerten Bilddaten von einer Seite über ein Feld von Koeffizientenmultiplizierern schieben. Zwischen den Schiebeoperationen wird jeweils das Faltungsergebnis berechnet und ausgegeben. Strukturell findet sich hier natürlich das im vorigen Abschnitt erwähnte Schieberegister der maximalen Verzögerungslänge der Matrixzeilenlänge mit seinen Abzweigungen für jeden Koeffizienten wieder, nur daß hier die Verzögerung mit dem Multiplizierer als Komponente eines systolischen Arrays aufgefaßt wurde.

6.3.1 Koeffizientenverkettung

Hängen die finanziellen Kosten für die Hardwarerealisierung von Echtzeit–Bildverarbeitungsoperatoren nicht von der Matrixgröße ab, sondern — was für eine Architektur anzustreben ist — von der Zahl besetzter Koeffizienten, sind die Matrizen am effizientesten, bei denen alle Koeffizienten null–gesetzt wurden, bei denen dies nur zu Fehlern des Ergebnisses unterhalb einer gewissen Grenze führt. Für radialsymmetrische Operatoren liegen Koeffi-

3. Koeffizientenmultiplizierer hängen an einer Schieberegisterkette :

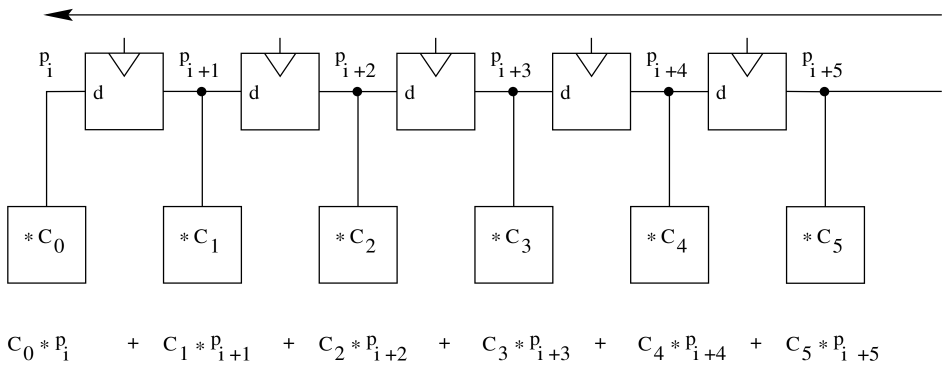


Abbildung 6.6: Hardwarerealisierung

zienten derselben Werte auf konzentrischen Kreisen, so daß eine kreisförmige Matrizenbesetzung die vorhandenen Koeffizienten optimal ausnutzt. Speziell richtungsunabhängige Filter, die für jede Orientierung eines Objektes — oder aus Sicht des Filters der Umgebung — dasselbe Ergebnis liefern, besitzt radialsymmetrisch angeordnete Koeffizienten. Um beliebige Operatorumgebungen unterstützen zu können, wurden hier eine Architektur gewählt, die Koeffizientenzellen nicht festverschaltet angeordnet, sondern sie — als Schieberegisterketten beliebiger Länge — den einzelnen Pixelströmen zuordnet. Zusammen mit der Möglichkeit das Timing der Pixelströme vorzugeben, ermöglicht dies auch ausgefallene Operatorumgebungen effizient zu realisieren.

Koeffizientenanordnung

Gewöhnlich nimmt die absolute Gewichtung von Pixeln der Umgebung mit zunehmender Distanz vom Zentralpixel ab. Sind die Werte hinreichend klein geworden, lassen sie sich schließlich ohne nennenswerten Fehler nullsetzen. Die Anzahl verbleibender von Null verschiedener Koeffizienten bestimmt den Rechenaufwand für die Faltung. Obwohl die benötigten Koeffizienten, die betragsmäßig größer als ein Restwert sind, fast nie rechteckig angeordnet sind (vgl. [*Klette, Zamperoni 1992*, S. 12]), werden meistens rechteckige Faltungsmatrizen benutzt. Bei richtungsunabhängigen radialsymmetrischen Operatoren auf einer $n * n$ Matrix können jedoch z. B. durch Nullsetzen aller Koeffizienten, die kleiner als der größte Randwert der Matrix sind, etwa $(1 - \pi/4) * n^2$ also 21% der Koeffizienten ohne größeren Fehler eingespart werden. Spezielle Operatoren, wie etwa Quadraturfilter (s. [*Jähne 1991*, S. 130]), weisen nur in kleinen Bereichen ihres umschreibenden Umgebungsrechtecks signifikante Koeffizienten auf.

Die sich aus der technischen Realisierung ergebenden Restriktionen lassen sich durch folgendes Modell schnell erläutern: Relativ zu dem um einige Zyklen verzögerten Eingangspixelstrom, lassen sich 6 weitere durch ihre Verzögerung definierte Pixelströme konfigurieren. An jeden der 7 Ströme lassen sich Ketten von Koeffizientenregistern beliebiger Länge

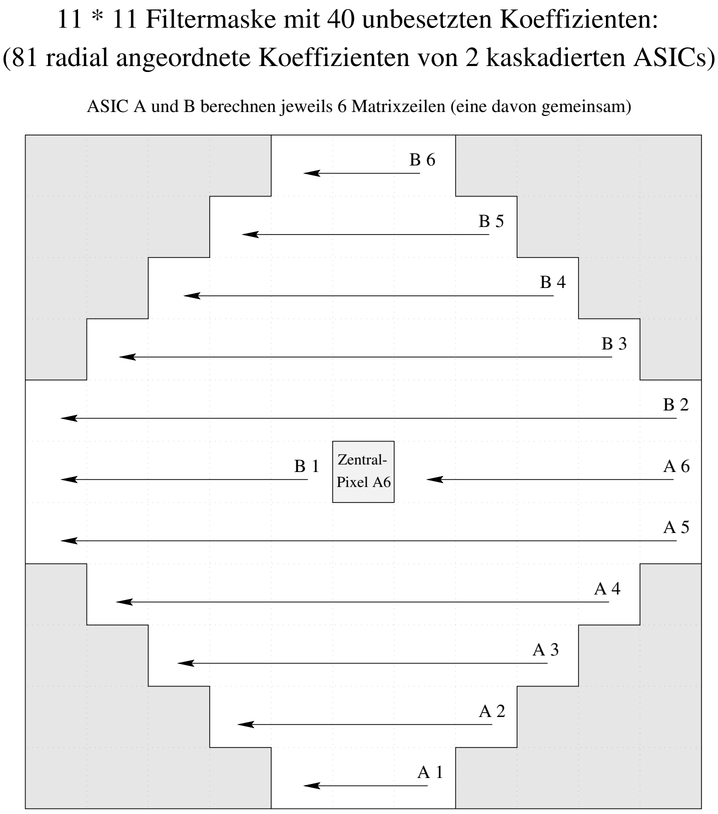


Abbildung 6.7: Realisierung einer radialen Faltungsmatrix

hängen. Die Ketten bilden mit dem Pixeltakt betriebene Schieberegister (vgl. Abb. D.6). Hier findet sich Abbildung der genutzten Datenpfade eines ASICs für zwei konkrete Konfigurationen. Die einzelnen Kettenglieder, die hier nur stilisiert abgebildet sind bilden mit ihrem 10–Bit Integer–Koeffizienten gewichtete Ergebniswerte ihrer Eingabepixel (s. Abb. 7.3, eine detailliert Behandlung folgt noch).

Diese Struktur kommt z. B. radialsymmetrischen isotropen Filtern besser entgegen als eine starre 8 mal 8 Matrix. Mit zwei ASICs läßt sich so z. B. eine näherungsweise radiale Filtermatrix realisieren, die eine maximale Ausdehnung von 11 Pixeln in jeder Richtung hat. Dazu berechnet je eine ASIC die fünf oberen und unteren Koeffizientenzeilen, indem es an fünf seiner entsprechend verzögerten Pixelströme Schieberegisterketten anhängt (s. Pfeile, Abb. 6.7). Die Berechnung der mittleren Matrixzeile, der Matrix teilen sich beide ASICs. Das ASIC, das das Endergebnis ausgibt (A), kann zudem noch den gewichteten Zentralpixel hinzuaddieren. Damit sind alle eingezeichneten 81 Koeffizienten besetzt.

6.3.2 Externer Zeilenspeicher

Zum Auslagern der Zeilenspeicher empfiehlt sich eine Lösung, die möglichst wenig Pins belegt und die äußere Beschaltung auf ein Minimum reduziert. Typisch 20 ns Zugriffszeit ermöglichen einen Zugriff mit doppeltem Pixeltakt (34 ns) auf externe handelsübliche statische RAMs. Da mit jedem Pixeltakt der Eingabewert und die 6 Werte der letzten 6 Zeilen aus dem Speicher benötigt werden, sowie der älteste mit dem Eingabewert zu überschreiben ist, fallen 7 Byte Speicherbandbreite pro Pixeltakt an. Eine Parallelisierung mittels 4 statischer Speicherbausteine mit je einem Byte Zugriffsbreite bei doppeltem Pixeltakt ist also notwendig. Um nur einen Satz Adreßpins herausführen zu müssen, wird der aus vier SRAMs bestehende Speicher hier 32–Bit breit angesprochen. Die eingehenden Daten werden in einem 4 Byte Schieberegister gesammelt, und, sobald es voll ist, in 2 Takten — also einem Pixeltakt — parallel ins RAM geschrieben. In den restlichen 6 Takten (3 Pixeltakten) werden je 4 Bytes der einzelnen Zeilen als Wort gelesen. Nach 8 Takten (4 Pixeltakten) wiederholt sich der Vorgang zyklisch.

Zur Adreßgenerierung für die Lesezyklen verzögerter Pixelströme wird als Offset der negative $Verzögerungswert/4$ addiert. Für die exakte, pixelgenaue Einstellung der Verzögerung wird durch programmierbare Schieberegister mit 0 bis 3 Takten Verzögerung jeder Pixelstrom noch um den Restwert, Verzögerungswert modulo 4, verzögert. Die Adresse von Pixelpositionen der Bilder im Zeilenspeicher ist dabei nicht fest. Vielmehr wird das SRAM durch Ignorieren eines Adreßberechnungsüberlaufs als zyklischer Puffer betrieben. Ist der Puffer bzw. der Speicher zu klein, enthalten die letzten Pixelströme falsche Werte, nämlich die der überschreibenden ersten Pixelströme.

6.4 Faltungskonfiguration

Faltungsmatrizen zur Binomialfilterung erfordern im allgemeinen alle 40 Koeffizienten, die z.B. in einer 7 * 7 Matrix radial angeordnet sind. Einige Operationen erfordern dagegen nur sehr schwach, mitunter nur mit zwei Koeffizienten, besetzte Matrizen — etwa zum Differenzieren. Weiterhin gibt es häufig Operationen, neben deren Ergebnis noch die Summe oder Differenz mit dem Zentralpixel benötigt wird. Statt die Berechnung mit einem zweiten ASIC durchführen zu müssen, bietet der hier realisierte ASIC eine Teilung der Koeffizienten in zwei gleichstarke Mengen, die je ein Ergebnis liefern, sowie einen Extrazweig, der den Zentralpixel synchron und beliebig gewichtet als drittes Ergebnis heranführt. Bei zwei Ausgängen bietet sich nun die Möglichkeit, zwei verschiedene Faltungsmatrizen mit je bis zu 20 besetzten Elementen und getrennt oder eine mit 40 Koeffizienten besetzte Faltungsmatrix zweimal, direkt und mit dem gewichteten Zentralpixel verknüpft auszugeben. Werden zwei Teilmatrizen berechnet, ist man für beide auf denselben Zentralpixel festgelegt.

Dieses kann z. B. für eine simultane Berechnung der X– und Y–Ableitung eines Bildes oder zur Berechnung der Gauß– und Laplacepyramide aus der Gaußpyramide der vorherigen Stufe benutzt werden. Die Verknüpfungen (Summe, Minimum– oder Maximumbildung) der beiden Teilmatrizen, ihrer Vereinigungs–, der Kaskadierungs– und der Zentralpixelverknüpfung sind jeweils getrennt wählbar. Dies ermöglicht z. B. eine Dilatation (Maximumsuche) mit anschließender Differenzbildung zu dem Zentralpixel.

6.5 Randwertproblematik

Da die Pixel keine endlosen Pixelströme, sondern Pixel sind, die horizontale und vertikale Bildränder kennen, über die hinauszufalten nicht sinnvoll ist, gibt es ein Bit breite Zähler der horizontalen und vertikalen Bildränder für den Zentralpixel, den aktuellen Pixelstrom und alle 6 verzögerten Pixelströme. Die Zählbits sind für jeden Pixel zeitlich invariant und können, wie auch sein Wert, mit Schieberegisterketten verzögert werden. Damit werden nur für die durch das SRAM verzögert zur Verfügung gestellten Pixel Zähler benötigt. Zur Berechnung herangezogen werden nur Pixel, die zum Zentralpixel eine bestimmte Zählerdifferenz aufweisen, die sich aus der Faltungsmatrix und der horizontalen Zählerdifferenz pro Bildzeile ergibt. Der Rest wird für die Berechnung durch einen Ersatzwert ausgeblendet. Ein Beispiel für die Maskierung zeigt Abb. 6.8. In ihr sind gesetzte Zählbits der horizontalen Bildbereiche durch einen waagerechten und solche vertikaler Bildbereiche durch einen senkrechten Strich gekennzeichnet. Der Bildbereich wurde hier bewußt über die normalen Bildränder geschoben, da nur hier Randwerte maskiert wurden.

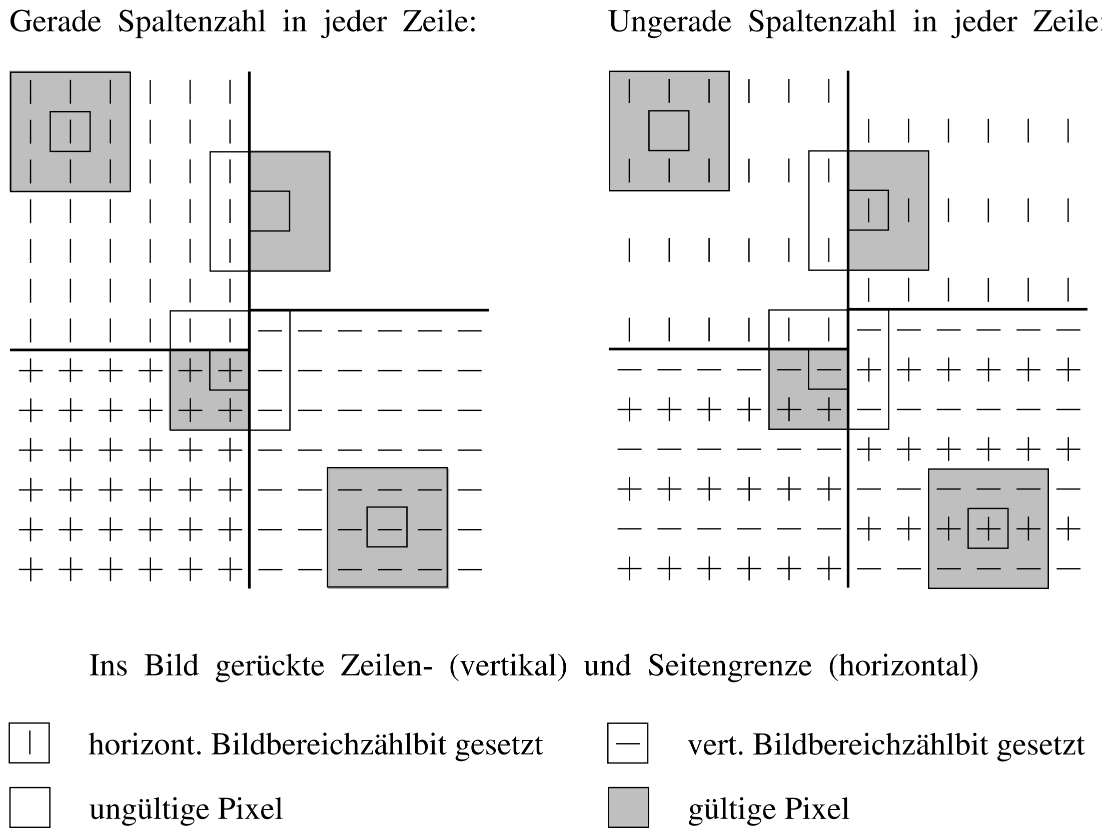


Abbildung 6.8: Blick auf vier aneinandergrenzende Bilder

Ein Gesamtbild, in dem alle Bildbereiche eines Pyramidenbildes durchgezählt und die Bedingungen für gültige Pixel der Faltungsmatrix angegeben sind, findet sich im Anhang, Abb. D.5. Dieses Verfahren funktioniert genau, solange alle Faltungsmatrixkoeffizienten maximal zu dem aktuellen Fenster des Zentralpixels benachbarte Fenster überdecken. Eine hinreichende Bedingung ist, daß die Faltungsmatrix nicht größer als der kleinste definierte Bildbereich ist. Bei im Matrixzentrum gelegenen Zentralpixeln darf der kleinste definierte Bildbereich sogar halb so groß wie die Faltungsmatrix sein. $9 * 9$ Matrizen mit in der Mitte gelegenen Zentralpixeln können über bis zu $4 * 4$ Pixeln großen Bilder korrekt mit Ersatzwerten falten. Die restlichen Pyramidenstufen $2 * 2$ und das Pixel der obersten Pyramidenebene können nicht mehr korrekt berechnet werden.

6.6 Initialisierung

Kaskadierte ASICs hängen alle am selben Eingangspixelstrom und die horizontal kaskadierten eventuell auch noch an demselben SRAM–Zeilenspeicher (s. Abb. 6.10). Ein *power–on*–Reset ist zwingend nötig, damit keine zwei ASICs vor der Initialisierung den RAM–Datenbus treiben. Dieser schaltet sie darüber hinaus in einen 64 * 1 Matrixmode: Eine einzige, lange Koeffizientenregisterkette ist bei allen ASICs direkt an den Pixeleingang des ASICs geschaltet. Wie in einer *scan–pass*–Queue werden nun die Koeffizienten und Konfigurationsdaten in die ASICs zur Initialisierung hineingetaktet. Ein Signal fordert dann das ASIC auf, die momentan geladenen Daten zu übernehmen, deren IDs dem geladenen ID–Bitmuster entsprechen. Sind alle ASICs initialisiert, werden die Bildbereichszählbits aller ASICs zeitgleich auf einen vorbestimmten Verarbeitungszustand gesetzt. Dieser Zählerreset ist auch jederzeit über die Leitung *xy_reset* aufrufbar. Im Anschluß werden dann die regulären Pixeldaten verarbeitet. Da das ASIC synchron zum Pixeltakt läuft, ist eine Synchronisation nur nach einem Reset nötig. Nach einer vorausberechenbaren Zeit hat sich das ASIC aus einem ROM, dessen Adreßbus parallel zum Zeilenspeicher liegt und das über ROM_ena statt der Pixeldaten auf den Eingang geschaltet wird, selbständig konfiguriert.

Die Dauer für die Initialisierungsphase entspricht der Zeit, die benötigt wird, um das Initialisierungsrom auszulesen. Dessen Länge kann aus der Symboltabelle des ROM–Generators (Treibersoftware) abgelesen werden. Es muß nur noch die gewählte Teilung der ROM–Clock berücksichtigt werden. Nach ROM–Inhaltslänge * ROM–Teilungsfaktor positiven Pixelclockflanken beginnt mit der nächsten Flanke das Einlesen der Pixel. Dies entspricht der ersten Pixelclockflanke nach dem Zurücknehmen des ROM–Enable Signals. Da die Zähler der Pixelpositionen für die Pixel einer internen Pipelinestufe (die zur Randwertmaskierung) relevant sind, muß bei der Initialisierung statt der Position des beim ersten Takt anliegenden Pixels die 6 Takte vorher anliegende Pixelposition angegeben werden. Die Ausgabeverzögerung teilt sich in eine systeminhärente und eine durch Pipelinestufen verursachte Verzögerungszeit. Erstere ergibt sich aus der Distanz des Zentralpixels zum zuletzt übertragenen Randpixel der Operatorumgebung. Nach Vorliegen des Zentralpixels muß die Zeit bis zum Anliegen des zuletzt benötigten Randpixels abgewartet werden, bevor die Berechnung abgeschlossen werden kann. Der zusätzliche Zeitverlust beträgt 22 Takte; zusammen mit der Eingabeverzögerung ergeben sich demnach 28 Takte Pipeliningverzögerungen. Sowohl die Ein– als auch die Ausgabeverzögerung sind dabei nur von der ASIC–

Parametrisierung abhängig, nicht von der gewählten Betriebsart. Die Positionen des ersten Eingabe– und Ausgabepixels nach dem Reset werden von dem Konfigurationsprogramm berechnet und zur Information angezeigt. Die Anzeigekomponente ist auf den Startpixel (0, 0) ebenso eingestellt, wie die Beipielkonfigurationsdateien. Den Startzeitpunkt ermittelt sie durch Auswerten des ROM–Enable Signals. Als Matrixgröße wurde von einer 5 * 5 Pixel Matrix ausgegangen, und die sich daraus ergebende Verzögerung des Ausgabepixels durch einen Positionsoffset für das Ausgabebild kompensiert.

Für den Startzeitpunkt müssen u. a. die Startzähler für alle kaskadierten ASICs konsistent berechnet werden. Dabei sind die Verzögerungen der Teilsummenberechnung durch die Kaskadierung zu berücksichtigen. ASICs deren Ergebnis verzögert ausgewertet wird müssen den jeweiligen Zentralpixelzählerzustand des Ausgabe–ASIC am Ende der Kaskade entsprechend früher einnehmen. Jede Kaskadierungsstufe schlägt dabei mit drei Takten Verzögerung zu Buche. Um nicht für jede Operator–Konfiguration wieder eine gültige Zählerbitkonfiguration durch Zurückzählen der Zähler, entsprechend der Verzögerung, berechnen zu müssen, erledigt dies ein Konfigurationsprogramm für einzelne und kaskadierte ASICs. Es geht davon aus, daß kaskadierte ASICs ebenso wie in der VHDL–Testumgebung "verschaltet" sind.

6.7 Kaskadierung

Die Kaskadierungsmöglichkeit dieses ASICs ermöglicht es Operatormatrizen unbeschränkter Größe zu realisieren. Jedes ASIC berechnet dabei einen Ausschnitt aus dem Faltungsfenster des Operators. Zur Berechnung der Faltungssummen aus den Teilsummen, die jedes einzelne ASIC berechnet, werden diese zur Ergebnisweitergabe linear verkettet oder "kaskadiert".

Zur Bewältigung der großen Datenrate der 7 Pixelströme ist bereits ein 32 Bit breiter Zeilenspeicher aus 4 parallel betriebenen 8–Bit SRAMs erforderlich. Bei meßtechnischer Bildverarbeitung werden häufig größere Matrizen gefordert. Dem trägt dieser ASIC durch weitrechende Kaskadierungsmöglichkeiten Rechnung. Ein besonders einfaches Konzept besteht darin, mehrere ASICs parallel laufen zu lassen, wobei alle ASICs bis auf einen nur ihre 6 im Timing variierbaren Pixelströme nutzen, um jeweils andere Koeffizienten der Matrix abdecken zu können; das letzte kann auch den siebten aktuellen Pixelstrom nutzen. Alle kaskadierten ASICs, berechnen nun Teilsummen des Faltungsergebnis.

Ohne eine Instanz zum Bilden der Gesamtsumme kommt man aus, wenn jedes ASIC in der Lage ist, zu seinem Ergebnis noch das eines zweiten ASICs hinzuzuverknüpfen (Summation, Extrema). Die ASICs werden dann zur Ergebnisberechnung in Reihe geschaltet. Das Verknüpfen und Weiterreichen der Teilsumme der Faltung über Pipelinestufen benötigt zwar einige Extratakte, ist aber trotzdem unproblematisch, da das Timing der Pixelströme so gewählt werden kann, daß Ergebnisse, die mehrere ASICs und somit Pipelinestufen durchlaufen, entprechend viele Takte vorher berechnet werden.

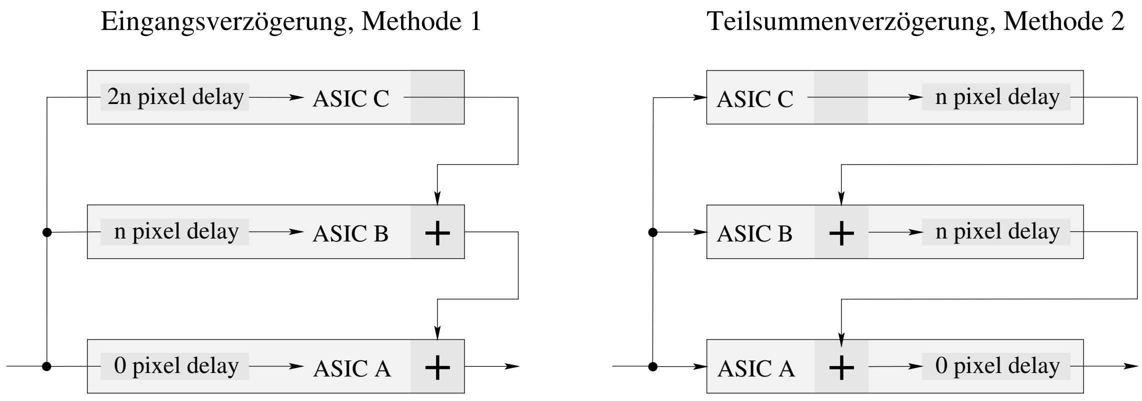


Abbildung 6.9: 3 horizontal kaskadierte ASICs

Speziell bei der horizontalen Kaskadierung von ASICs, die hier so genannt wird, weil die ASICs auf denselben Bildzeilen, allerdings auf horizontal benachbarten Bildbereichen, arbeiten, sind komplette Zeilenspeicher nicht unbedingt nötig, da die Pixelströme jeweils nur um wenige Takte individuell in jedem ASIC verzögert werden müßten. Hier empfiehlt es sich, programmierbare *on–chip delay–lines* einzusetzen (s. Abb. 6.9, links; die Faltung der Teilmatrix findet an der Stelle statt, an der die Bezeichnung der ASICs stehen). Die horizontalen ASICs hängen alle am selben Zeilenspeicher. Alle ASICs wählen die SRAM–Offsets des ASICs, der die Daten zuerst benötigt. Zusätzliche Verzögerungen werden über interne *delay–lines* erzeugt. Da die ASICs dann absolut synchron arbeiten, können ihre Busse zum RAM parallelgeschaltet werden. Damit ein Hardwarefehler nicht Bausteine durch unterschiedlich getrieben Busse zerstört, wird nur ein ASIC als Master konfiguriert, alle anderen unterdrücken ihre write–Bustreiber. Die ASICs werden nun so konfiguriert, daß sie ihre einzelnen Pixelströme zusätzlich durch die *delay–lines* wie benötigt verzögern. Soll weiter horizontal kaskadiert werden, als die *on–chip delay–lines* erlauben, muß wie bei vertikaler Kaskadierung wieder neuer Zeilenspeicher vorgesehen werden oder folgende horizontale Verzögerungsart benutzt werden.

Alternativ zum individuellen Verzögern gibt es, wie schon besprochen, auch die Möglichkeit, die Ergebnisse vor der Verknüpfung zu verzögern. Eine fixe Verzögerung ist ohnehin schon durch die Pipelinestufen bei der Ergebnisberechnung über die kaskadierten ASICs vorgegeben. Zusätzlich kann nun mit einem variabel langen Schieberegister eine zusätzliche Verzögerung gewählt werden. Diese Lösung ist insofern unflexibler, als die bearbeiteten Matrixzeilenbereiche nicht individuell zu verschieben sind, sondern nur als Ganzes. Mit der individuellen Verzögerung lassen sich bei radialsymmetrischen Matrizen auch bei horizontaler Kaskadierung einige Koeffizienten einsparen, was mit der globalen Verschiebung nicht möglich ist. Sie benötigt jedoch weniger Schieberegister und kann somit vorteilhaft zur größeren gemeinsamen horizontalen Verschiebung der Umgebung dienen, die das ASIC bearbeitet. Außerdem lassen sich so beliebig viele ASICs horizontal kaskadieren, so daß sie mit nur einem gemeinsamen Zeilenspeicher auskommen. Die Länge bzw. die Implementation dieser Verzögerungseinheiten läßt sich durch die Parametrisierung kontrollieren.

Diese horizontale Kaskadierungsmethode ist rechts in Abb. 6.9 dargestellt (man beachte die strukturelle Ähnlichkeit zu Abb. 6.2).

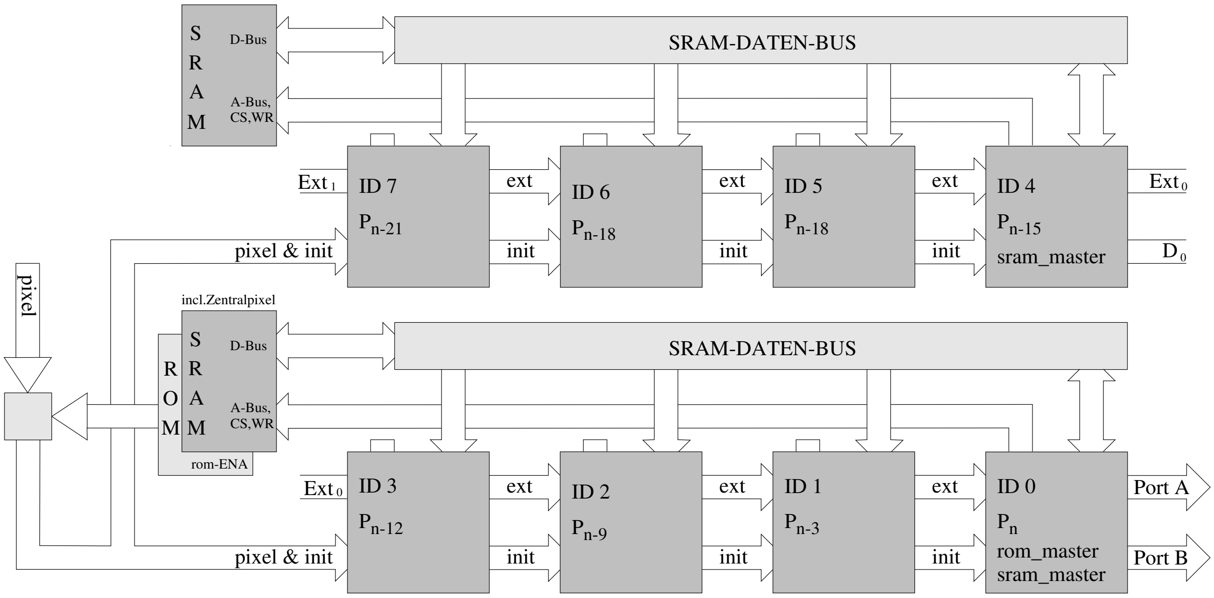


Abbildung 6.10: Kaskadierung: Zweifach vertikal, vierfach horizontal

In der aktuelle Parametrisierung des ASICs — und auch von der Treibersoftware her — werden beide Methoden zur horizontalen Kaskadierung unterstützt. Um einen gewissen Spielraum zur individuellen Verschiebung der Koeffizientenzeilen — trotz nach der zweiten Methode horizontal kaskadierten ASICs — zu erhalten, lassen sich alle Pixelströme intern individuell um bis zu drei Pixeltakte verzögern. Die Verzögerung nach der zweiten Methode erlaubt es zudem, den Bildbereich jedes ASICs um jeweils drei bis zu zehn Pixel horizontal gegenüber seinem Vorgänger zu verschieben. Die minimale Verschiebung ergibt sich daraus, daß pro Kaskadierungsstufe drei Pipelinestufen in den einzelnen ASICs zur Ergebnisverknüpfung durchlaufen werden müssen. Diese Verzögerung tritt auch bei der vertikalen Kaskadierung auf, da jedoch in diesem Fall jedes ASICs seinen eigenen frei konfigurierbaren Verzögerungsspeicher besitzt, läßt sich dieser Effekt vollständig kompensieren. Er wird in diesem Fall durch die Treibersoftware vor dem Anwender verborgen.

Beide Kaskadierungsarten, die horizontale, bei der die ASICs einen Zeilenspeicher gemeinsam nutzen, was dem Anwender jedoch Timingbeschränkungen auferlegt, und die vertikale, die auf Kosten individueller Zeilenspeicher unbeschränkte Konfiguration der in jedem ASIC berechneten Koeffizientenzeilen der Matrix erlaubt, lassen sich kombiniert einsetzen. In Abb. 6.10 ist als Beispiel eine ASIC–Kaskade dargestellt, die Operatoren bis zu 13 Zeilen Höhe (der aktuelle Pixelstrom ist nur von der unteren ASIC–Reihe im Bild nutzbar), bis zu 40 Pixeln Breite und mit insgesamt bis zu 321 Koeffizienten ($8*40$ Koeffizienten/ASIC und ein Zentralpixel) erlaubt. (Die geringe Höhe ergibt sich daraus, daß sich weitere vertikal kaskadierte Einheiten strukturgleich ergänzen lassen, hier nicht eingezeichnet wurden.)

Kapitel 7

Ergebnisse und ASIC–Realisierung

Im folgenden Kapitel wird die Implementation des ASICs besprochen. Dies beinhaltet auch eine Beschreibung der Komponenten auf der Hauptblockebene und deren Zusammenfassung zu hardwarebeschriebenen Probeaufbauten.

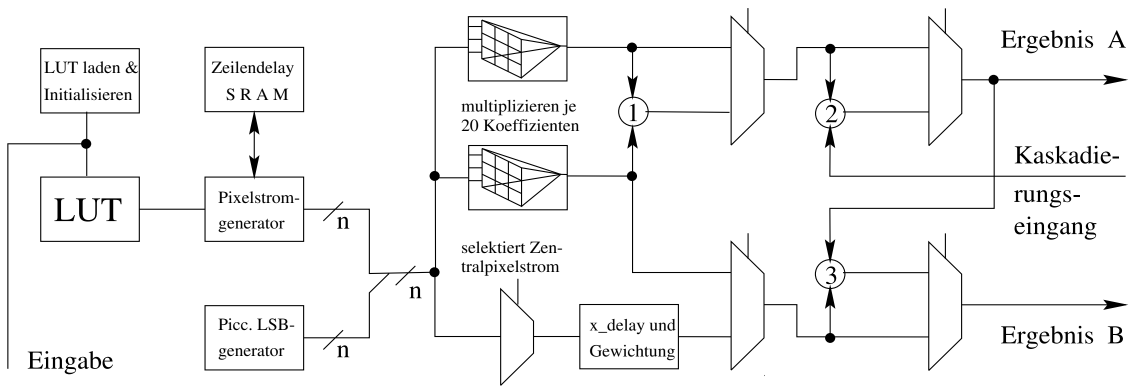
Das hier entworfene ASIC ist *stand–alone* ohne jede intelligente Peripherie arbeitsfähig ist und liest seine Konfigurationsparameter nach jedem Reset aus einem angeschlossenen ROM aus. Da die Registerinhalte im ROM nicht *byte–aligned* stehen und sich ihre Länge und Anzahl mit der gewählten Parametrisierung des ASICs zudem ändern, wurde das Programm *gen_rom* entwickelt, das aus einer Liste von Werten und Bitbreiten von Registern einen Bitstring generiert und diesen in Bytes formatiert ausgibt.

Trotz dieser relativ komfortablen Eingabemöglichkeiten, erwies sich die korrekte Initialisierung der Bildbereichszähler und die Berücksichtigung von Verzögerungen zwischen kaskadierten Chips als so komplex, daß eine korrekte Initialisierung meistens erst nach mehreren Versuchen, zuweilen erst durch Nachvollzug der internen Verarbeitungsschritte im ASIC mit dem VHDL–Debugger, gelang. Da dieses Vorgehen zeitraubend bzw. bei späterer Anwendung nicht mehr möglich ist, wurde noch die Treibersoftware *get_para* entworfen, die Parameterdateien automatisch erzeugt. Dazu werden die freien Parameter vom Benutzer interaktiv abgefragt, wobei immer der mögliche Eingabebereich mitangegeben wird. Neben der Parameterdateiausgabe wird auch der Eingabedialog als Datei gespeichert. Kleinere Detailänderungen können nun in dieser Datei nachgetragen und ohne großen Aufwand zu einer neuen Parameterdatei übersetzt werden.

Anschließend wurde der ASIC–Entwurf mit verschiedenen Parametrisierungen übersetzt und an einem kleinen Bild einige Faltungs–, Dilatations– und Erosionsoperatoren getestet. Der ROM–Inhalt zur Parameterversorgung wurde automatisch generiert. Zum Schluß wurde auf Simulationsebene ein kleiner Testaufbau, bestehend aus dem generierten ROM, einem Testbild– oder Realbildgeberprozeß, einem Monitorprozeß, der einen Pixelstrom in einem Fenster als Bild darstellt und bereits als fertiger VHDL–Prozeß vorlag, dem Zeilenspeicher und dem ASIC selbst, beschrieben, übersetzt und simuliert. Zwei weitere Testaufbauten, in denen einmal zwei ASICs mit getrennten Zeilenspeichern und einmal mit einem gemeinsamen Zeilenspeicher, realisiert wurden, dienten der Erprobung der Kaskadierungsschnittstellen.

7.1 VHDL–Implementation

Abb. 7.1 zeigt ein Strukturbild der Hauptblockebene. Im folgenden werden die einzelnen Komponenten der Abbildung besprochen. Verhaltensorientiert formulierte Komponenten, wie z. B. *lut*, werden dabei informativ beschrieben, wobei in der Beschreibung naturgemäß keine Details der Schaltungsstruktur referenziert werden können. Bei verhaltensorientiert beschriebenen Prozessen, deren strukturelle Umsetzung offensichtlich ist, wie etwa bei Schieberegistern, wird trotzdem auf solche, in der konkreten VHDL–Beschreibung nicht konkret auftauchenden, Strukturelemente Bezug genommen. Die Komponente *multi_queue* wurde im Gegensatz zu *lut* strukturbeschrieben, so daß sich bei ihrer Beschreibung zahlreiche Referenzen auf ihren VHDL–Sourcecode finden. Eine Ebene tiefer in der Hierarchieabbildung der ASIC–Komponenten (s. Abb. D.1) finden sich schon Komponenten der Register–Transferebene. *ser_2_par* ist ein Schiebergister mit serieller Eingabe und paralleler Ausgabe der Daten aller Schiebestufen, *par_2_ser* dagegen ein Schieberegister, das seine Daten parallel lädt und dann seriell heraustaktet. Die Komponente *delay* schließlich ist ein Schieberegister mit variabler, über ein Bitvektor selektierbaren, Länge. Die Komponente *lut_sram* nimmt eine Sonderstellung ein. Sie liefert nur zu Simulationszwecken eine Verhaltensbeschreibung eines Makrozellen–SRAMs. Kommentare, beginnend mit speziellen Schlüsselworten, verhindern daher eine Synthese der Verhaltensbeschreibung. Nach der Synthese des übrigen ASIC–Entwurfs wird die aus der Megacell–Library generierte SRAM–Makrozelle dazugelinkt. (Eine exakte Referenz auf das benötigte SRAM findet sich im Kommentar der VHDL–Verhaltensbeschreibung).



1. Verknüpfung der beiden Halbmatrizen
2. Verknüpfung des Teilergebnis eines weiteren ASICs (Kaskadierung)
3. Verknüpfung des Zentralpixels mit Ergebnis A zum Ergebnis B

Abbildung 7.1: ASIC–Entwurf auf Hauptblockebene

Die Abbildungen 7.2, 7.4, 7.5 und 7.6 stellen den Datenfluß im ASIC innerhalb der verschiedenen Verarbeitungsstufen dar. Sie zeigen je einen Quadranten eines einzigen Bildes. Um den Zusammenhang zu verdeutlichen, wurde dieses Bild zudem in den Anhang als Abb. D.3 aufgenommen. Ebenfalls dort findet sich Abb. D.4 in ihr ist dagegen der strukturelle Aufbau dargestellt und auch die Bildbereichszähler *sub_pics_counter* eingezeichnet, die im vorigen Bild der Übersichtlichkeit halber nicht dargestellt waren; aus demselben Grund wurden hier nun allerdings die letzten Komponenten der Ausgabeformatierung als eine *black–box*–Komponente dargestellt.

7.1.1 Initialisierung der Look–Up–Table: *lut*

Die Komponente *lut* erfüllt hauptsächlich Konfigurationsaufgaben. Sie initialisiert nach jedem Reset das SRAM, das als LUT dient, die Konfigurationsqueues innerhalb der Koeffizientenzellen, die das Pixelstromtiming bestimmenden Offsetregister und alle sonstigen Register des ASICs. Nach vollendeter Initialisierung steuert sie nur noch das *on–chip* SRAM als LUT an.

Zur Ablaufsteuerung der Initialisierung wurde ein endlicher Automat realisiert, der auf zwei Prozesse verteilt ist, die mit der positiven und negativen Flanke der Pixelclock getaktet werden. Der Arbeitstakt dieses Automaten ist mit dem Signal *work_ena* maskiert. Dieses wird durch ein UND–Gatter aus den untersten Bits eine Zählers gewonnen, der mit jedem Pixeltakt inkrementiert wird. Werden z. B. die untersten drei Bits verwendet, läßt *work_ena* nur jeden achten Takt einen Zustandswechsel des Automaten zu. Die Verzögerung dient dazu, auf die Daten aus dem gegenüber dem Zeilenspeicher–SRAM deutlich langsameren Konfigurations–ROM zu warten, das während der Initialisierungsphase die Daten liefert. Die sich dadurch ergebende Taktteilung ist in Zweierpotenzen mit dem Parameter *rom_dl* einstellbar. Beliebig hohe Teilungsfaktoren (auch der Faktor eins, entsprechend $rom\_dl = 0$) sind dadurch möglich. In der aktuellen Parametrisierung wurde zur Initialisierung eine Verzögerung um den Faktor acht gewählt.

Synchron zum Arbeitstakt wird nun permanent die ROM–Adresse hochgezählt. Der eingehende Datenstrom wird in ein zweistufiges Schieberegister getaktet, so daß immer drei nacheinanderfolgende Datenbytes simultan gelesen werden können. Der endliche Automat interpretiert diese drei Bytes immer dann als neuen Befehl, wenn der vorherige abgearbeitet wurde. Eines der drei Befehlsbytes enthält die ID des ASIC, für den die Daten bestimmt sind. Diese Befehlszuordnung ist nötig, um kaskadierte ASICs mit denselben Daten individuell konfigurieren zu können. Das nächste enthält den Befehl selbst und das dritte schließlich die Länge der Daten, die zu dem Befehl gehören und hinter ihm folgen. Alle ASICs müssen nun die bis zum nächsten Befehl noch ausstehenden Datenbytes herunterzählen und so den Anfang des nächsten Befehls abwarten. Das ASIC, das die Daten verarbeiten soll, erzeugt währenddessen außerdem noch Steuersignale, die ein Eintakten der Daten in die, in Form von Schieberegister realisierten, Konfigurationsregister des ASIC bewirken. Zum Laden des LUT–SRAM müssen zusätzlich Adressen generiert und die Steuerleitungen des SRAMs angesteuert werden.

Es gibt zwei Befehle, die auch ohne daß die ID des Befehls und des ASICs übereinstimmen ausgeführt werden: Der Befehl *init_ok* versetzt alle ASICs in den Ausführungsmodus. Dazu legt er den Initialisierungsteil lahm und nimmt das globale *init_phase* Signal zurück. Statt der ROM–Adressen werden nun die Zeilenspeicher–SRAM–Adressen an den entsprechenden ASIC–Bus angelegt. Außerdem werden erst jetzt die Schreib– und Datenleitungen durchgeschaltet, da ansonsten die nicht vollständig konfigurierten horizontal kaskadierten ASICs die Treiber zu ihrem gemeinsamen Zeilenspeicherbus durch gleichzeitige unkoordinierte Schreibzugriffe zerstören könnten. Der Automat zur Zeilenspeicheransteuerung in der Komponente *multi_queue* läuft sofort nach dem Reset los. Während des ASIC–spezifischen Eintaktens der Timing–Konfiguration finden dabei unkoordinierte Speicherzugriffe auf das Zeilenspeicher SRAM statt. Der Adreßbus ist jedoch nicht gefährdet. Als unidirektionaler Bus muß er nur von einem ASIC getrieben werden, die Adreßbusse der anderen ASICs bleiben unverbunden. Ist die Konfiguration vollendet, ist die Gefahr gebannt. Zum einen würden alle ASICs nun dieselben Werte auf den Datenbus legen, zum anderen wird nur ein ASIC als *Busmaster* konfiguriert. Dies verhindert Hardwaredefekte, falls ein ASIC defekt sein sollte oder fehlerhafte Konfigurationsdaten verwendet werden.

Der zweite ID–unmaskierte Befehl, *load_val*, dient zum Eintakten der Koeffizientenzellenkonfiguration. Jede Koeffizientenzelle beinhaltet eine kleine Queue aus zwei Bytes. Statt nun 40 dieser Queues in Reihe zu schalten, werden sie an die lokal vorhandene Pixelstromleitung angeschlossen. Durch den Reset liegen die Koeffizientenzellen in Form einer 40 Koeffizienten breiten (1 Pixel hohen) Matrix vor, also als Queue verschaltet. Diese ist an den Pixeleingang angeschlossen, der in der Initialisierungsphase stattdessen als Dateneingang für das Konfigurations–ROM dient. Nach jeweils 41 Takten enthalten alle Koeffizientenzellen (incl. der des Zentralpixels) neue Daten, die alten wurde herausgetaktet. Statt die Daten als Pixelwerte zu interpretieren, werden sie nun als Konfigurationsdaten in alle 41 zwei Byte langen lokalen Konfigurationsqueues der Koeffizientenzellen getaktet. Dazu wird nach dem Eintakten zuerst das Register–Enable für die 41–elementige Queue zurückgenommen. In allen ASICs liegen nun an allen 40 Koeffizientenzellen und an derjenigen des Zentralpixels, die nur während der Konfiguration das letzte Glied der Queue bildet, die neuen Daten permanent an. Für das richtige ASIC muß nun noch für die lokalen zwei Byte langen Queues innerhalb der Koeffizientenzellen *einen* Takt lang das Enable–Signal zum Schieben gegeben werden.

Dazu dient einer der vier Befehle, die nur ausgeführt werden, wenn die ASIC–ID mit der Befehls–ID übereinstimmt. Er heißt *coeff* und benötigt an sich keine Daten. Da jedoch der nächste Befehl *LUT* 256 Daten besitzt, aber maximal 255 im Datenlängefeld angeben werden können, wird der dort angegebenen Wert bei der Übernahme stets inkrementiert. Aus diesem Grund folgt dem *load_val* Befehl noch ein *dummy*–Datenbyte beliebigen Inhalts. (Genaugenommen werden drei "Datenbytes" mehr gelesen als im Feld angegeben, da die ersten beiden der drei Befehlsbytes für sich genommen auch nur uninterpretierbare Daten darstellen. Das nächste Byte komplettiert dann erst den nächsten Befehl.) Es wäre zu

Überlegen, ob ein am Ende der Datenzyklen von *load_val* generierter Enablepuls vielleicht günstiger wäre, als den Extra–Befehl *coeff* dekodieren zu müssen. Das Enable–Signal der Register in den Koeffizientenzellen läßt sich jedoch keinesfalls einsparen, da es auch zur Schiebetaktteilung bei geteiltem Initialisierungstakt benötigt wird.

Der zweite ID–maskierte Befehle heißt LUT; er lädt den Inhalt der LUT, bestehend aus 256 Bytes. Dazu werden fortlaufend alle LUT–Adressen erzeugt und die Steuerleitungen so angesteuert, daß die eingetakteten Daten direkt ins LUT–SRAM geschrieben werden. Die restlichen beiden Befehle *Timing* und *Status* sind sehr ähnlich. Der erste initialisiert die lokalen Daten in der Komponente *multi_queue*, also alle Adressoffsets für den Zeilenspeicher und die *delay*–Werte für die Pixelströme. Der letzte versorgt schließlich die Initialisierungswerte für die Randwertzähler und alle sonstigen Register mit ihren Konfigurationsdaten. Während die Daten einlaufen wird dazu einfach Enable für das Timing– bzw. Statusregister gegeben, die als Schieberegister beide am Pixel–/ Dateneingang angeschlossen sind.

Da die Reihenfolge der Befehle zur Konfiguration des ASICs gleichgültig ist, gibt es mehrere gleichwertige Möglichkeiten, zu derselben Konfiguration zu kommen. Zudem führen viele verschiedene Konfigurationen zu demselben äußeren Verhalten des ASICs (wie auch z.B. verschiedene Sortieralgorithmen dasselbe leisten). Während des Normalbetriebs beschränkt sich die Aufgabe der Komponente LUT lediglich darin, dem LUT–SRAM die Eingabepixel gelatched als Adresse zu übergeben.

7.1.2 Zustandsregister: *state_reg*

Die beiden Konfigurationsschieberegister im ASIC–Design, die — wie bereits besprochen — die Verzögerungskonfiguration des Pixelstromgenerators *multi_queue* bzw. alle sonstigen Konfigurationsdaten (incl. der Bildbereichstabelle für die Komponente *sub_pics_counter*) enthalten, wurden in der Komponente *state_reg* isoliert. Das diente dazu, ihre reguläre Struktur zur Synthese durch entsprechende Attribute gesteuert auch ohne zeitraubende und nutzlose Optimierbemühungen auf Gatter abbilden zu können. Man könnte diese Register jetzt auch problemlos zu einem zusammenfassen. Einsparungen würden dabei jedoch in der Komponente *state_reg* selbst nicht auftreten, da in den letzten Bytes der Schieberegister nur die tatsächlich benötigten Bits in Hardware realisiert werden. Da es zudem einige Änderungen und ein erneutes Austesten der Konfigurationssoftware erfordern würde, wurde darauf verzichtet.

7.1.3 Pixelstromverzögerung: *multi_queue*

Die einzige Aufgabe der Komponente *multi_queue* (in Abb. 7.2 als *black–box*–Komponente “δ”, im Anhang als Teil der Abb. D.4 dargestellt) besteht darin, einen einlaufenden Pixelstrom mittels eines Zeilenspeicher–SRAMs mehrfach um beliebige Zeitspannen verzögert auszugeben. Dazu werden, ebenso wie rechts in Abb. 6.3 gezeigt, im ersten Zyklus vier Pixel aus einem seriell eintaktenden Schieberegister parallel ins SRAM geschrieben. Um

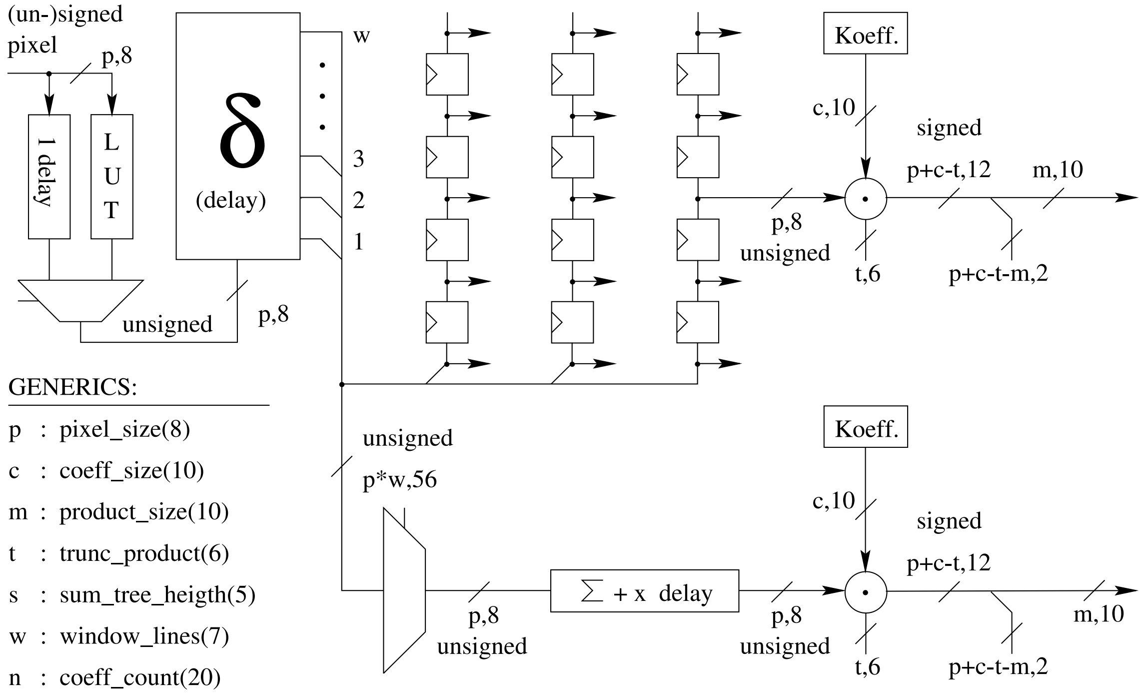


Abbildung 7.2: Pixelgewichtung (Matrix konfiguriert dargestellt)

die Speicherbandbreite voll auszunutzen, wird für lesende Speicherzugriffe mit dem doppelten Pixeltakt gearbeitet. Während alle anderen ASIC–Komponenten mit dem Pixeltakt (*clk1*) arbeiten, der aus dem doppelt so schnellen Eingabetakt *clk2* durch Taktteilung hervorgeht, werden zum Auslesen des Zeilenspeichers die folgenden drei Pixeltakt–Zyklen in sechs *clk2–Zyklen* unterteilt. In jedem werden vier Pixel (Signalbus *data*) mit einem individuellen Adressoffset zur fortlaufenden Basisadresse *base* ausgelesen. Diejenigen, die in der ersten Zyklushälfte des Pixeltakts gelesen wurden, werden in einem D–Flipflop gespeichert (*data_b*), die anderen gelatched (*data_a*). Dies ist nötig, damit die Komponente unabhängig davon funktioniert, ob die Pixelclock clk1 oder ob clk2 an gemeinsamen Flanken früher toggelt.

Anschließend folgen wieder mit dem einfachen Pixelclock getaktete Schieberegister, die die Daten zu der ihnen zugewiesenen Phase übernehmen und seriell austakten. Danach liegen die Pixelströme zum ersten mal nach dem Zeilenspeicher wieder seriell vor — allerdings sind sie zeitlich noch gegeneinander verschoben (*multi_tdma_bus*). Erst durch ein Schieberegister von null bis drei Pixeln Länge kann eine pixelgenaue Verzögerung erfolgen. Es kompensiert sowohl, daß die Queues der ersten Phasen ihre Werte zwei Pixeltakte früher erhalten als die der letzten Phase als auch daß der Adressoffset eine Granularität von vier Pixeln pro Adreßwert aufweist. Das Ergebnis wird auf einem Teil der Signale des Busses *multi_line_out* ausgegeben. Die fehlenden zwei Signale pro Pixelstrom steuern die Bildbereichszähler in Form ihrer 1–Bit Zähler für die vertikalen und horizontalen Bildbereiche bei.

Zum Abzählen der Phasen des SRAM–Zugriffs wird ein Prozeß instantiiert, der ein 3–Bit

Wort zyklisch von 0 bis 7 inkrementiert und damit alle acht Phasen des Zeilenspeicherzugriffs abzählt. Nach je einem solchen Phasenzyklus wird zudem die Basisadresse *base*, unter der die Pixel direkt abgespeichert werden, inkrementiert (im Prozeß *FSM_state_increment*). Der folgende Prozeß (*FSM_function*) steuert nun in Abhängigkeit der Phasen Signale für die Schreib– und Lesezyklen des Zeilenspeicher–SRAMs und die Adreßgenerierung. Letztere beinhaltet eine Pipelinestufe für die Offsetaddition, damit die Adreßwerte sofort gültig ausgegeben werden können. Dazu wird mit jedem Phasenwechsel die vorher berechnete Offset–Adresse (*next_addr*) vom erstgenannten Prozeß (*FSM_state_increment*) ausgegeben.

Ein weiterer Prozeß generiert die *load_enable* Signale der Schieberegister (Signal *load*), die die Signale *data_a* und *data_b* jeweils parallel laden und permanent seriell austakten. Die Erzeugung des write–Signals für den doppelt so langen Schreibzyklus wurde durch die zwei Prozesse *delay_sram_write_signal* und *extended_init_phase* realisiert. Es ist nicht auszuschließen, daß mitten innerhalb des Schreibzyklus die Initialisierungsphase des ASIC von der Komponente LUT abgeschlossen wird. Dies führt zu einem mit der negativen Pixelclockflanke zeitgleichen Umschalten auf das Zeilenspeicher–SRAM — mitten in den laufenden Schreibzyklus hinein. Da dieser halbe Schreibzyklus zu zahlreichen *timing–violations* führt hat der zweite Prozeß die Aufgabe, in einem solchen Fall das Durchschalten des *write*–Signals bis zum Ende des laufenden Schreibzyklus zu verzögern.

7.1.4 Bildbereichszähler: *sub_pics_counter*

Der Bildbereichszähler zählt die horizontalen und vertikalen Bildgrenzen, die überschritten werden, mit Hilfe von 1–Bit Zählern. Dies kommt der Aufgabe gleich, beim Passieren einer Bildgrenze jeweils das entsprechende Zählbit zu toggeln. Neben den äußeren Bildgrenzen, dem rechten und unteren Bildrand, ist eine weitere Unterteilung in logische Bildbereiche innerhalb des Gesamtbildes möglich. Dazu kann das Bild in horizontale Bildstreifen zerlegt werden. Jeder dieser Streifen kann wiederum durch vertikale Bildgrenzen weiter unterteilt werden. Jedes so entstandene Teilbild, wird nun unabhängig vom benachbarten verarbeitet. An den Bildgrenzen wird dazu ein Randersatzwert für die Pixel der Operatorumgebung eingeblendet, die in benachbarte Teilbilder ragen.

Prinzipiell bedarf es eigentlich nur eines Bildbereichszählers, da die Zählbits zu jedem Pixel ebenso wie seine Position im Bild ein unveränderbares Attribut darstellen. Dieses ließe sich, nachdem es für den Eingangsstrom einmal berechnet wurde, gemeinsam mit dem Wert des Pixels verzögern. Bei kaskadierten ASICs kann es jedoch vorkommen, daß der verzögerte Pixelstrom, der den Zentralpixel zum Verarbeitungszeitpunkt enthält, gar nicht auf dem ASIC gebildet wird. Anstatt nun dafür einen Pixelstrom zu opfern, nur um an die mit ihm übertragenen Zählbits zu gelangen, ist es günstiger, für den Zentralpixel einen zweiten Bildbereichszähler zu instantiieren. Durch seine Zählbits gibt er für alle Koeffizientenzellen den Bildbereich vor, für den das Rechenergebnis gültig sein wird. Will man nun den Aufwand für den externen Zeilenspeicher möglichst gering halten, kann man die Zählbits auch erst nach der Generierung der verzögerten Pixelströme für jeden einzelnen berechnen und erst von nun an mit dem Pixelstrom weiter übertragen. Statt zehn Bit sind jetzt lediglich noch acht Bit Daten zu verzögern; dafür werden jetzt sechs zusätzliche Bildbereichszähler benötigt. Sie bestehen im wesentlichen jedoch nur aus zwei Pixelzählern und –komparatoren sowie zwei Multiplexern, um auf die zwei globalen Bildbereichstabellen indiziert zugreifen zu können (die eingebettet im ASIC–Statusregister vorliegen).

Die Teiltabelle der horizontalen Bildbereiche existiert dabei pro Bildstreifen genau einmal. Die aktuelle Tabelle wird durch das vertikale Indexregister (s. u.) selektiert. Sie enthält der Reihe nach die horizontalen Bildgrenzen und den Index des Eintrages, der die Bildgrenze des Gesamtbildes referenziert. Die vertikale Bildbereichstabelle existiert dagegen nur einfach. Neben den aufsteigend geordneten vertikalen Bildgrenzen wird auch hier der Index aufgeführt, mit dessen Bildgrenze gleichzeitig die Gesamtbildgrenze erreicht wurde. Neben der ASIC–globalen Tabelle gibt es noch zwei individuelle Indexregister, die aus den Tabellen die Pixelpositionen der nächsten zu erwartenden horizontalen und vertikalen Bildränder selektieren, und zwei Zähler für die aktuelle vertikale und horizontale Pixelposition.

Der horizontale Pixelzähler wird nun mit jedem Pixeltakt inkrementiert. Stimmt sein Inhalt mit der nächsten horizontalen Bildgrenze überein, wird das Zählbit der horizontalen Bildbereiche getoggelt und das entsprechende Indexregister inkrementiert. Lag jedoch schon der letztbenutzte Index vor, wurde also der rechte Gesamtbildrand erreicht, wird der horizontale Pixelzähler und das Indexregister auf null zurückgesetzt und der vertikale Pixelzähler inkrementiert. Für diesen gilt nun genau dasselbe. Wurde der vertikale Pixelzähler auf den Wert der nächsten vertikalen Bildgrenze inkrementiert, der durch das vertikalen Indexregister aus der Bildbereichstabelle selektiert wurde, wird das vertikale Zählbit getoggelt und das vertikale Indexregister inkrementiert. Falls jedoch die vertikale Gesamtbildgrenze erreicht wurde, wird das vertikale Indexregister zusammen mit dem vertikalen Pixelzähler auf null zurückgesetzt.

Die Tabelle enthält für m mögliche Bildstreifen mit maximal n horizontalen Bildbereiche $m * n$ horizontale Bildgrenzeneinträge und m Einträge der vertikalen Bildgrenzen. Man sollte also bei der ASIC–Parametrisierung vor allem n, die Zahl horizontaler Bildbereiche nicht größer als unbedingt nötig wählen. Das ASIC wurde mit $n = 2$ und $m = 13$ instantiiert. So lassen sich in der Horizontalen, individuell für jeden Bildstreifen, z. B. ein — links– oder rechtsbündig zum Gesamtbild liegender — benutzter von einem unbenutzten Bildbereich abgrenzen. In der Vertikalen lassen sich 12 Bildstreifen und ein unbenutzter unterer Randstreifen definieren. Dies ermöglicht u. a. die Verarbeitung von in einem Gesamtbild vertikal eingebetteten Bildpyramiden. Neben den Zählbits wird auch das niederwertigste Bit des letzten horizontalen Index des aktuellen Bildstreifens ausgegeben. Es gibt an, ob das horizontale Zählbit für gleiche horizontale Bildbereiche mit jeder Zeile toggelt, weil eine ungerade Zahl horizontaler Bildgrenzen pro Bildstreifen gewählt wurde, oder ob es konstant bleibt. Mit dieser Information, dem Zustand der Zählbits des Zentralpixelzählers und dem Wissen welcher Matrixzeile ein Koeffizient angehört kann nun aufgrund der mitgeführten Zählbits jedes Pixels durch eine boolsche Funktion individuell festgestellt werden, ob der Pixel im selben Bildbereich wie der Zentralpixel liegt, und damit gültig ist, oder ob er durch den Randwert ausgeblendet werden muß.

7.1.5 Matrixkoeffizienten: *product_cell*

Die Multiplikation mit den Koeffizienten geschieht in 40 einzelnen Multipliziereinheiten, die als systolisches Array feldmäßig verteilt ihre Bilddaten von einem Nachbarn bekommen und danach an ihr Gegenüber weitergeben (s. Abb. 7.2). Eine weitere, leicht modifizierte Multipliziereinheit (*product_central*) gewichtet außerdem den Zentralpixel. Die Koeffizientenwerte selbst sind vorzeichenbehaftete Integerwerte, ebenso wie das Ergebnis der Pixelgewichtung und das nachfolgender Verarbeitungseinheiten. Die Pixeleingabewerte, werden jedoch nach dem Durchlaufen der LUT als Werte in *unsigned*–Darstellung interpretiert. Nur bei der Ergebnisausgabe treten, bei entsprechender Konfiguration, wieder *unsigned*–Darstellungen auf. Das Ergebnis der Gewichtung eines Pixels hat also immer dasselbe Vorzeichen wie der Koeffizient (oder es ist null). Da die LUT beliebige Operationen erlaubt, stellt sich die Frage der Pixelwert–Darstellung für den Eingabestrom nicht (die der Pixelwert–Interpretation dagegen schon, von ihr wird weiter unten noch die Rede sein). Die Beschränkung auf eine Verarbeitung von *unsigned*–Pixelwerten stellt jedoch keine Einschränkung dar. Vorzeichenbehaftete Bilddaten lassen sich mit der Eingangs–LUT durch eine Addition des maximalen negativen Betrages zu allen Pixelwerten immer in eine *displaced*–Integerdarstellung bringen. Da die Faltung eine lineare Operation darstellt — und die Faltungsmatrix bekannt ist —, können durch Subtraktion eines Offsets nach der Faltung die korrekten Ergebnisse erhalten werden. Zum Eingabebild wurde bei der *displaced*–Darstellung formal ein konstantes Grauwertbild des *displaced*–Wertes addiert. Das konstante Faltungsergebnis dieses Grauwertbildes mit der Matrix entspricht dem gesuchten Korrekturoffset. Bei Dilatations– und Erosionsberechnungen ist es noch einfacher, es muß nur wieder der *displaced*–Wert abgezogen werden. Es ist allerdings nicht möglich, die auslaufende Umgebungsgewichtung von Faltungen auf Dilatations– und Erosionsberechnungen *vorzeichenbehafteter* Bilder zu übertragen, da Rangordnungsoperatoren nicht linear sind.

Für Dilatations– und Erosionsberechnungen *vorzeichenloser* Bilder sind gewichtete Umgebungen jedoch möglich, man muß nur das Vorzeichen der Koeffizienten geschickt wählen und eventuell eine Ausgabe des negativen gesuchten Ergebnis in einer *displaced*–Darstellung akzeptieren. Dieser Schönheitsfehler kann dann von der LUT des nächsten ASICs oder einer externen LUT (schnelles 256–Byte ROM) korrigiert werden. Die Ursache liegt darin, daß Koeffizientengewichtungen mit dem Wert 0 zur Ausblendung bei der Erosion, die das Umgebungsminimum sucht, denkbar schlecht geeignet sind. Das Ergebnis null stände von vornherein fest. Auch beliebig hohe Koeffizienten erfüllen den Zweck schon deswegen nicht, weil Pixel mit dem Wert 0 unabhängig von ihrer Gewichtung konstant null bleiben. Daher wird bei Dilatations– und Erosionsberechnungen der Koeffizient 0 anders interpretiert. Es wird anstatt des Ergebnisses 0 immer der Extremwert der Zahldarstellung geliefert, der die Maximum– oder Minimumsuche nicht beeinflußt. Der gegenteilige Erosions–Koeffizient wäre z. B. 1.00. Beschränkt man sich auf diese beiden Koeffizienten, lassen sich auch vorzeichenbehaftete Bilder in displaced–Darstellung mit anschließendem Korrekturoffset berechnen (s. o.). Liegen vorzeichenlose Bilder vor, ist die Sonderbehandlung des Koeffizienten 0 nicht unbedingt nötig (wenn auch nicht störend). Bei der Dilatation (Maximumsuche) müssen lediglich positive, bei der Erosion (Minimumsuche) dagegen negative Koeffizientenwerte eingesetzt werden, damit auch der Wert 0 die Extremasuche nicht beeinflußt. Koeffizienten zwischen 1 und 0 bzw. -1 und 0 erfüllen dann ihren Zweck, Pixel der Umgebung unterschiedlich stark gewichtet ins Ergebnis miteinzubeziehen. Bei Erosionsoperatoren liegt das Ergebnis dann allerding negativ vor. Mit einem Offset kann es immerhin so verschoben werden, daß es den unsigned– oder auch signed–Bereich bei der Ausgabe und damit die 8–Bit Quantisierung voll ausnutzt.

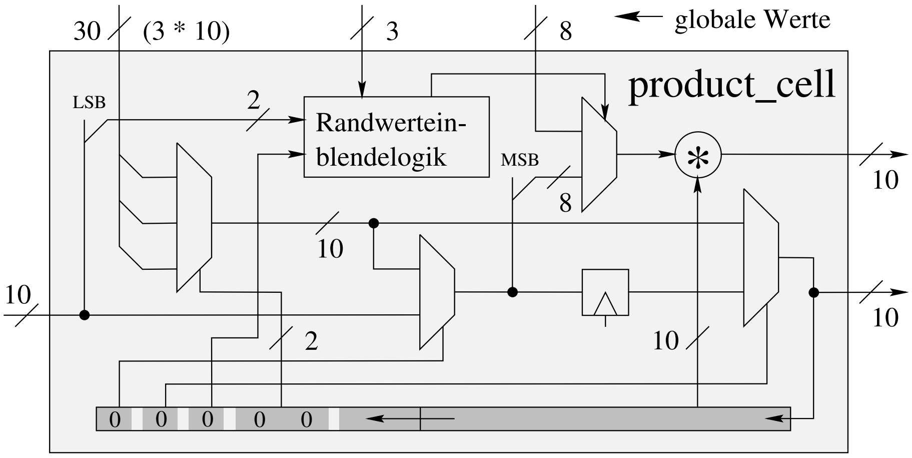


Abbildung 7.3: Einzelne Koeffizientenzelle

Der Aufbau einer Koeffizientenzelle ist in Abb. 7.3 dargestellt (der Übersichtlichkeit halber ohne den Mechanismus zur Randwerteinblendung). Jede Koeffizientenzelle hat zwei Dateneingänge und einen Ausgang. In der Resetkonfiguration bildet sie mit allen anderen Zellen eine 41–elementige Queue. Mit ihrem einen Eingang sind sie jeweils mit dem Ausgang ihrer Vorgängerzelle verbunden. An die erste Zelle der Queue ist stattdessen der aktuelle also unverzögerte Datenstrom angeschlossen. Jede Koeffizientenzelle kann außer ihrem Vorgänger auch Pixel jedes zweiten verzögerten Pixelstroms direkt übernehmen. Die Auswahl der drei Pixelströme erfolgt über einen Multiplexer, wobei die zugänglichen Pixelströme mit jeder Zelle abwechseln. Benötigt die Zelle keine Daten der ihr zugänglichen Pixelströme, weil sie die ihres Vorgängers verarbeitet, kann sie zu ihrem Nachfolger anstatt des ein Takt verzögerten eigenen Pixelstroms auch einen der ihr zugänglichen Datenströme über ihren Ausgang durchschalten, den dieser wiederum nicht erreichen kann. Damit können in der überwiegenden Zahl aller Fälle alle Koeffizientenzellen genutzt werden, obwohl die Hälfte der Verdrahtung entfällt und die Multiplexer nur halb so groß sind.

Im Anhang, Abb. D.6 finden sich zwei Konfigurationsbeispiele. Um einen Überblick zu ermöglichen, sind die Datenpfade der Eingabepixel durch die Koeffizientenzellen eines auf fünf Matrixzeilen und 18 Koeffizienten parametrisierten ASICs stark verkleinert und nur grob stilisiert dargestellt (anstelle von Multiplexern sind nur die benutzten Verbindungen als breite Linien eingezeichnet). Für beide Teilmatrizen sind alle Matrixformen möglich, die die Pixelströme der Reihe nach anfordern. Sind die Pixelströme so gewählt, daß jede zweite Matrixzeile von je einer Zelle erreichbar ist und die Matrixzeilen fortlaufend in die Koeffizientenqueue eingebettet sind, führt die Annahme eines Ressourcen–Konfliktfalls zum Widerspruch. Kann eine Zelle ihren Quellpixelstrom nicht selektieren, wählt sie ihn über ihren Vorgänger an. (Benötigte die erste Zelle die nicht vorhandene nullte zur Quellstrom–Selektion, wird die Konfiguration der Quellströme vorher so vertauscht, daß alle Zellen jeweils die anderen Quellströme lesen können). Annahme: Der einzige Quellenstrom–Multiplexer der Vorgängerzelle wird schon von dieser belegt. Widerspruch: Die Vorgängerzelle benötigt also dieselbe Matrixzeile oder eine mit geradzahligem Zeilenabstand, denn die mit ungeradzahligem Zeilenabstand sind ihr nicht zugänglich. Die einzigen Quellpixelströme, die sie bei fortlaufender Einbettung der Matrixzeilen benötigen kann, sind jedoch die des Vorgängers oder seines Quellenstrom–Multiplexer, was im Widerspruch zum angenommenen Ressourcen–Konflikt steht.

Durch die Aufteilung in zwei identische Teilmatrizen tritt jedoch für Parametrisierungen mit einer ungerader Anzahl von Koeffizienten pro Teilmatrix eine Irregularität auf. Selbst wenn die alternierende Reihenfolge beibehalten würde, könnte die erste Zelle nicht auf alle Ströme zugreifen, da die Zuordnung der Matrixzeilen zu den Pixelströmen schon durch die erste Teilmatrix reglementiert ist (falls diese nicht den aktuellen Pixelstrom für die erste Matrixzeile nutzt). Unter ungünstigen Umständen kann so eine Koeffizientenzelle unbenutzbar werden, da sie auch über ihren Vorgänger nicht ihren gewünschten Quellstrom lesen kann. Es ist jedoch möglich, daß der Konflikt mit einem Abweichen von der Regel der Pixelstromverteilung umgangen werden kann. Mit der freien Wahl der Matrixzeilen für jeden (außer dem aktuellen) Pixelstrom stehen ebenso viele Möglichkeiten offen, die sechs frei definierbaren Pixelströme in zwei Gruppen einzuteilen wie es 6–Bit Zahlen mit drei gesetzten Bits gibt, wobei das n–te Bit die Gruppenzugehörigkeit des n–ten Stroms selektiert. Auch ohne Rechnung sind dies deutlich weniger als 64 Wahlmöglichkeiten, die von der Treibersoftware ohne merkliche Verzögerung getestet werden.

Außer den drei Multiplexern zur Pixelstromverteilung und dem Register zur verzögerten Weitergabe des eigenen Pixelstroms zur nächsten Zelle werden in der Komponente noch der Multiplizierer für das Produkt aus einem *signed* und einem *unsigned*–Integerfaktor, der zur Pixelgewichtung dient, instantiiert. Außerdem finden sich in der Komponente noch ein Multiplexer zur Randwerteinblendung nebst Ansteuerlogik, die die beiden zusammen mit den Pixeldaten mitgeführten Zählbits, drei vom Zentralpixelzähler stammende Bits und ein Konfigurationsbit zu seiner Ansteuerung logisch verknüpft (vgl. Anhang, Abb. D.5).

Die Reset–Konfiguration als durchgehende Queue wird, wie bereits beschrieben, von der Komponente *lut* genutzt, um die interne Konfigurationsqueue (unten in Abb. 7.3) der Koeffizientenzellen zu initialisieren. Die beiden Bits im Konfigurationsregister, die die Verkettung der Koeffizientenzellen steuern, müssen dazu bei jeder Parametrisierung aus dem obersten Byte ihrer lokalen zweistufigen Konfigurationsqueue stammen. Damit ist sicher-gestellt, daß die Koeffizientenzellen bis zu ihrer vollständigen Konfiguration als Queue verschaltet vorliegen. Dazu wird bei ungünstigen ASIC–Parametrisierungen (in einem von acht Fällen) die Queue um ein Dummy–Bit erweitert. Der Einfachheit halber geschieht dies dadurch, daß die Busbreite des Queueselektors (ASIC–Parameter: *lin_sz*), der ebenfalls in der Konfigurationsqueue steht, ein Bit breiter als nötig gewählt wird.

7.1.6 Zentralpixelkoeffizient: *product_central*

Die Komponente *product_central* dient der Zentralpixelgewichtung. Sie besteht aus einer reduzierten Fassung der Komponente *product_cell*. Für den Zentralpixel erübrigt sich naturgemäß eine Randwerteinblendung, da er ja den Bildbereich des Ergebnisses festlegt. Desweiteren ist er nur während der Initialisierungsphase letztes Glied der Queue, während er im Betrieb für sich alleine steht. Deswegen benötigt er keinen Ausgang, aber dafür doppelt soviele Eingänge, da er jeden Pixelstrom selbst erreichen können muß. Da sein Ergebnis keine Verknüpfungspipeline durchläuft, muß es gegenüber den anderen verzögert werden, um Verknüpfungen zwischen dem Zentralpixel und dem Faltungsergebnis zu erlauben. Dazu bietet sich ein Schieberegister am Eingang an, denn dort sind nur acht statt zehn Bit breite Daten zu verzögern. Zudem muß die horizontale Verzögerung, die bei der Faltung durch die Queue, die die Koeffizientenzellen innerhalb einer Matrixzeile bilden, erreicht wird, für den Zentralpixel durch eine explizit programmierbare Verzögerungsleitung nachgebildet werden. Damit kann dann ein beliebiger Koeffizient als Zentralpixel gewählt, extra verarbeitet und ausgegeben werden. Der Zentralpixelkomponente zur Seite gestellt ist ein Extra–Bildbereichszähler. Dieser gibt seine Zählbits und ein weiteres Tabellenbit (s. *sub_pics_counter*) als globale Signale zur Randwerteinblendung an alle anderen Koeffizientenzellen weiter. Auch wenn der Zentralpixelstrom bei kaskadierten ASICs nur in dem das Ergebnis abliefernden ASIC real vorhanden ist, zählen die Zentralbildbereichszähler in allen ASICs seine Position ab. Durch direkten Eingriff in die Konfigurationsdatei läßt sich bei allen ASICs, die ihren zweiten Ergebnisausgang nicht benötigen, ein beliebiger interner Pixelstrom auf diesem Ausgang ausgeben. Dies kann man erreichen, indem man ihn über den unbenutzten Datenweg des Zentralpixels an den zweiten Ausgang legt. Damit können die ASICs, die obere Koeffizienten der Matrix enthalten, als Eingabe bereits verzögerte Pixelströme benutzen, womit der beschränkte Adreßraum und damit die maximale Größe des Zeilenspeicher–SRAMs keine Einschränkung mehr darstellt.

7.1.7 Summierbaum: *sum_tree*

Die Produkte der Koeffizientenmultiplizierer, die gewichtete Eingabepixel darstellen, werden über einen Binärbaum mit Hilfe von speziellen ALUs verknüpft. Jede der ALUs besteht im wesentlichen aus einem Addierer, dessen einer Eingang optional, einschließlich des *carry–defaults* 0, invertiert werden kann, so daß die Differenz beider Argumente anstatt deren Summe berechnet wird. Zusätzlich zum Addierer ist noch ein Dreifach–Multiplexer vorhanden, der es erlaubt, die Summe oder eines der Argumente als Ergebnis auszugeben. Es gibt nun zwei Funktionseingänge, mit denen die Summe beider Argumente oder — indem der Multiplexer zusätzlich durch das Vorzeichenbit der Differenz angesteuert wird — das kleinere oder größere Argument ausgegeben werden kann. Die Komponente sum_tree ist ebenso wie prod_tree rekursiv formuliert. Der Aufruf zur Instantiierung des Verknüpfungs–Binärbaums erfolgt mit vorgegebener Anzahl von Argumenten und einem entsprechend breiten Argumentenvektor. Da aus Sicht des Datenflusses mit jeder Stufe die Daten ebenfalls ein Bit breiter werden, sind Überläufe ausgeschlossen. Von den speziellen Verknüpfungsmodi einmal abgesehen, läuft der Vorgang ähnlich wie bei der Addition der Zwischenergebnisse des Multiplizierers ab, nur daß diesmal die Blätter, die einen Zweig höher liegen, individuell verzögert werden müssen. Dies ist erforderlich, da von allen einzelnen Koeffizientenzellen her verschiedene Daten anliegen. Ebenso wie dort müssen die Eingabedaten um maximal einen Takt verzögert werden, was durch Zwischenschalten eines getakteten Registers erfolgt.

7.1.8 Bitbreitenreduktion: *barrel_shifter*

Diese Komponente führt arithmetische Schiebeoperationen aus. Allerdings wird dazu nur eine Selektion eines Teilbitstrings vorgenommen. Ergänzt man den Eingabebitstring um diegleiche Anzahl ungenutzter höchstwertigster Bits (*sign–extension*), erhält man einen richtigen *barrel_shifter*. Da dieser jedoch bei einigen Instantiierungen gar nicht benötigt wird, wurde diese Sparversion bevorzugt. In den Diagrammen zum Datenfluß wird zu ihrer Darstellung ein Symbol mit der Beschriftung "*asr*" gewählt, dessen Eingang links breiter als der Ausgang ist. Die Linienbündel sollen die verschiedenen möglichen Selektionpositionen des Teilbitstrings aus dem Eingabebitstring andeuten. Alle Daten werden zudem als vorzeichenbehaftete Zahlen interpretiert und der Selektionsprozeß als arithmetische Rechtsschiebeoperation. Sind demzufolge Bits oberhalb der selektierten Bitstring verschieden vom Vorzeichenbit, liegt ein Überlauf vor (bei der anfangs erwähnten *sign–extension* in voller Breite kann dies nicht passieren). Das Ergebnis wird dann vorzeichenrichtig auf den Maximalwert gesetzt.

7.1.9 Kaskadierung und Ausgabe: *front_end*

Die Komponente *front_end* dient der Aufbereitung der Daten der beiden Teilmatrizen und des Zentralpixels. Dazu werden durchgängig zwei ähnliche Datenpfade mit je einem Ausgang angeboten. Es gibt zwei Konfigurationsvarianten, entweder werden die Ergebnisse beider Teilmatrizen getrennt ausgegeben, oder es wird das Ergebnis der Gesamtmatrix und einer beliebigen Linearkombination des ersten Ergebnisses und des Zentralpixels berechnet. Die oberste Verknüpfungsstufe mußte dazu getrennt instantiiert (s. Abb. 7.4) werden, um auch auf die Einzelergebnisse der beiden Teilmatrizen getrennt zugreifen zu können.

Die Daten zur Berechnung des ersten Ergebnis durchlaufen die oberste Stufe zwar auf jeden

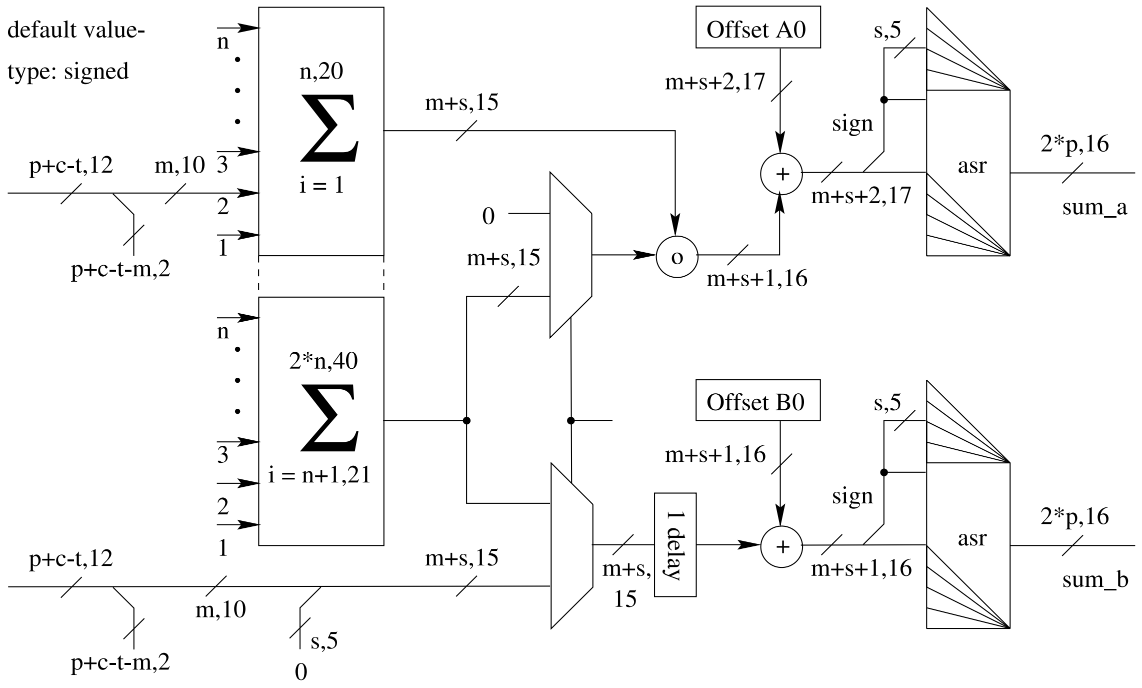


Abbildung 7.4: Summierstufen

Fall, um die Laufzeitverzögerung unabhängig von der Konfiguration konstant zu halten. Statt der zweiten Teilmatrix läßt sich aber auch der Wert Null verknüpfen und der Verknüpfungsmodus der obersten Stufe getrennt einstellen. Eine Summation von Null kann dann auch keine Dilatations– oder Erosionsberechnungen auf der ersten Teilmatrix mehr beeinflussen. Das Ergebnis der zweiten Teilmatrix kann statt der Verknüpfung mit demjenigen der ersten nun auch getrennt ausgegeben werden. Ein Verzögerungsregister kompensiert dazu die Laufzeit der obersten Verknüpfungsstufe der ersten Teilmatrix. Ansonsten wird auf dem unbenutzten zweiten Verarbeitungsstrang der Zentralpixel mitgeführt. Da er zur späteren Zentralpixelverknüpfung dasselbe Gewicht wie die Faltungssumme erreichen können muß, wird er auf die höchstwertigsten Bits des breiteren Teilsummenbus geschaltet. Ein Zurückschieben des Zentralpixelwertes in die Ausgangsgewichtung — etwa für Verknüpfungen mit Dilatations- oder Erosionsergebnissen — ist mit den nachfolgenden Shiftern möglich.

Bei der Kaskadierung von ASICs müssen interne Überläufe von Zwischenergebnissen vermieden werden. Deshalb ist es nötig, den Wertebereich des 16–Bit Kaskadierungsbus bei der Bildung des Teilergebnisses des ASICs, nicht voll auszuschöpfen. Um nicht schon bei den Koeffizienten auf die Nutzung eines Großteils des erlaubten Wertebereiches verzichten zu müssen, was zu erhöhten systematischen Quantisierungsfehlern der Matrix führen würde, ist es möglich, berechnete Teilergebnis so weit arithmetisch nach rechts zu schieben, daß einige höchstwertige Bits ungenutzt bleiben. In der technischen Realisierung werden dazu zuerst einige Vorzeichenbits als höchstwertigste Bits hinzugefügt (*sign–extension*) und dann das Ergebnis arithmetisch nach rechts geschoben. Wird um null Bit geschoben, liegt der ursprüngliche Wert an, ansonsten werden einige *sign–extension* Bits übernommen. Der Datenfluß der gewichteten Pixel bis zu dem aritmetischen *barrel_shifter* ist in Abb. 7.4 dargestellt.

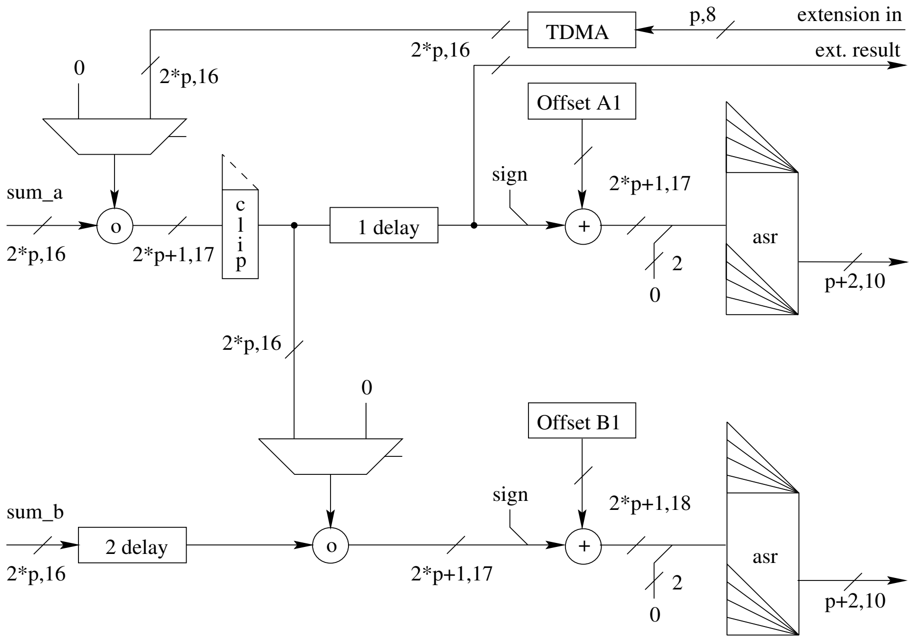


Abbildung 7.5: interne Extension–loop

Anschließend wird zum Ergebnis des ersten Verarbeitungsstranges bei Bedarf das Zwischenergebnis des Kaskadierungseingangs *extension_in* hinzuverknüpft (s. Abb. 7.5) und das oberste Bit durch einen fest parametrisierten *barrel_shifter* weggeclipt. Es liegt, wie oben schon angesprochen, in der Verantwortung des Benutzers, daß hier keine Überläufe stattfinden. Das Vorgehen dazu ist identisch zu dem der vorher erfolgten obersten Matrixverknüpfung, nur daß die Verzögerungskompensation des zweiten Verarbeitungsstranges wegen des Clippens nun zwei Takte beträgt. Anschließend wird dieselbe Prozedur für den zweiten Verarbeitungsstrang durchgeführt. Bei der Eine–Matrix Konfiguration wird zum Zentralpixel nun das Berechnungsergebnis der Matrix hinzuverknüpft. Durch Gewichtungsmöglichkeiten der Matrix und des Zentralpixels lassen sich beliebige Linearkombinationen bilden. Dadurch, daß noch mit doppelter Genauigkeit gearbeitet wird, kann auch ein gering gewichtetes Matrixergebnis später ohne Genauigkeitsverlust ausgegeben werden. Nun findet eine abschließende Offsetkorrektur statt, und die Ergebnisse werden mittels zweier *barrel_shifter* auf ein 10–Bit Zwischenformat gebracht.

Nach einer Multiplikation mit einem zwischen 1 und 2 wählbaren Faktor (das höchstwertigste Einsbit ist festverdrahtet, was die Treibersoftware berücksichtigt) wird zur Sicherheit noch auf 9–Bit Genauigkeit geclipt (s. Abb. 7.6). Die Multiplikation selbst dient dazu, den 9–Bit Wertebereich voll nutzen zu können. Die Multipliziereinheiten sparen wieder

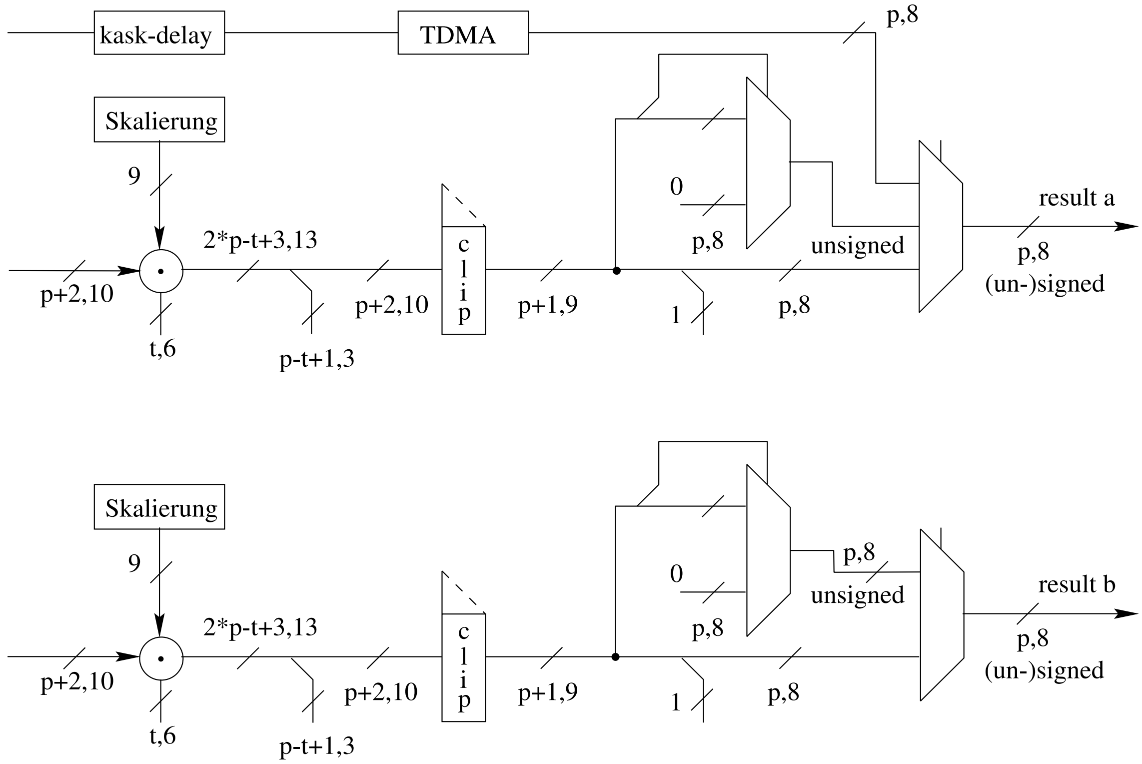


Abbildung 7.6: Ausgabeverknüpfungen des front–ends

die Berechnung der niederwerigsten sechs Bits ein und runden auf die nächsten drei, so daß die Ausgabe im selben Format wie die Eingabe vorliegt. Bei signed–Integerausgabe werden nun die acht höchstwertigen Bits ausgegeben, bei unsigned Ausgabe negative Werte anhand des Vorzeichens mit dem Multiplexer auf den Extremwert 0 geclipt und ansonsten die positiven Werte im unsigned–Format ausgegeben. Der Ausgabe–Multiplexer des ersten Verarbeitungsstranges erlaubt auch noch die alternative Ausgabe des Kaskadierungszwischenergebnis über den ersten Ausgang. Die doppelte Datenrate wird durch abwechselndes Senden des *high*– und *low*–Bytes des Zwischenergebniswortes synchron zu den Pixelclocktaktflanken bewältigt. Ein entsprechendes Empfangselement war in Abb. 7.5 ebenfalls dargestellt. Durch den Verlust des niederwertigsten Bits bei der signed–Ausgabe und durch den Skalierungsfaktor, der dafür sorgt, daß das oberste Bit nicht mehr benutzt wird, können bis zu zwei niederwertigste Bits verlorengehen. Um dies zu kompensieren, wurde nach der letzten Offsetkorrektur das Zwischenergebnis durch Hinzufügen zweier niederwertigster Nullbits (s. Abb. 7.5) entsprechend nach links geshiftet. Zur unsigned–Ausgabe wäre nur eines nötig, so daß der Wert zum arithmetischen Rechtsschieben um eins höher zu wählen ist. Die Treibersoftware berücksichtigt diesen Effekt bereits.

7.2 Konfiguration

Zur Konfiguration des ASICs wird ein Initialisierungs–ROM benötigt. Die entworfene Treibersoftware erstellt dieses in einem zweistufigen Prozeß mit Hilfe einer Software zur interaktiven Konfigurationseingabe, die eine Beschreibung des Inhaltes der ASIC–Konfigurationsregister erzeugt (*get_para*) und des Programmes *gen_rom* das diese in eine VHDL ROM–Beschreibung umsetzt. Die Datei *get_para.tpl* bildet mit dem gleichnamigen Programm zusammen eine Einheit, und darf nicht ohne weiteres verändert werden. Das Programm *gen_rom* benötigt zur weiteren Übersetzung einen VHDL ROM–Entwurf, in dessen Kopie die ROM–Daten eingefügt werden, er liegt als Datei *init_rom.tpl* vor.

Beide Programme lassen sich unter UNIX mit einem GNU C–Comiler *gcc* oder auf einem PC unter MS-DOS mit *Turbo–C++ Version 1.0* übersetzen und danach ausführen. (In letzterem Fall wird noch ein Batchfile *mv.bat* mit dem Inhalt "@rename %1 %2" im aktuellen Pfad benötigt.)

7.2.1 Bitstringprozessor: *gen_rom*

Die Aufgabe, ein Konfigurations–Rom zu erstellen, wird durch ein im Rahmen dieser Arbeit erstelltes Programm zur Verarbeitung von Bitstrings wesentlich erleichtert. Es müssen nur noch die richtigen Werte und Bitbreiten angegeben werden. Die Aufgabe, aus dieser Datei eine VHDL ROM–Beschreibung zu erzeugen, wird von dem Programm geleistet. Als zweite Eingabedatei benötigt sie dazu noch eine leere VHDL ROM–Beschreibung. Dank implementiertem Parsen arithmetischer Ausdrücke und von Konstanten kann die Eingabedatei nach Angabe der Parameter, mit denen das ASIC übersetzt wurde, veränderte Bitbreiten korrekt anpassen. Nur wenn sich die Anzahl der zu verarbeitenden Werte durch die Parametrisierung ebenfalls ändert, muß die Eingabedatei strukturell geändert werden. Da noch weitere Aufgaben zur Berechnung einer korrekten Konfiguration anfallen, wurde die Berechnung einer Eingabedatei auf eine Treibersoftware verlagert, die ganz im Gegensatz zu diesem Programm nur auf die Konfiguration dieses ASICs spezialisiert ist. So steht das folgende Programm in der Mitte einer Verarbeitungsqueue, seine Eingabe liefert die Treibersoftware, die Ausgabe übersetzt der VHDL–Compiler. Daher soll hier nur von seiner Funktionalität in Abhängigkeit von den Eingabedaten die Rede sein. Die Eingabedatei ist mittels der Schlüsselwörter *CONST*, *FORMAT* und *DATA* in die drei Bereiche aufgeteilt.

CONST–Umgebung

Um Bitstrings abhängig von Parametern berechnen zu können, erlaubt dies Programm die Nutzung von Konstanten, denen nur einmal ein Wert zugewiesen werde darf. Damit können innerhalb der *CONST*–Umgebung Textblöcke beliebig verschoben werden, ohne daß dies eine Auswirkung auf das Ergebnis hat — vorausgesetzt daß nicht gerade vor der Definition einer Konstanten auf sie zugegriffen wird, was zu einer Fehlermeldung führt. Um

32–Bit Strings als Eingabedaten zu ermöglichen, wurde für alle Konstanten die unsigned–Darstellung gewählt, was zur Folge hat, daß nur positive Konstanten erlaubt sind. Als mathematische Ausdrücke sind Summe, Differenz, Multiplikation, Division, Modulo, Potenzierung und der Logarithmus erlaubt, letzterer als zweistelliger infix–Operator der Basis gefolgt von dem Argument. Es werden die üblichen Rechenregeln befolgt, wobei Klammern unterstützt werden. Da auch bei Zwischenergebnissen nur positive Werte erlaubt sind, ist es wichtig zu wissen, in welcher Reihenfolge z. B. $9 - 5 + 3$ ausgewertet wird. Ausdrücke werden hier in derselben Reihenfolge ausgewertet, in der sie auch geschrieben werden, so daß der oben aufgeführte berechnet werden kann. Während sich Konstanten nur in der *CONST*–Umgebung definieren lassen, sind mathematische Ausdrücke überall erlaubt, wo Zahleneingaben erwartet werden.

DATA–Umgebung

In der durch das Schlüsselwort *DATA* eingeleiteten Umgebung werden nun Bitstrings durch ihren Zahlwert und ihre Bitbreite, getrennt durch einen Doppelpunkt, angegeben, die das Programm dreistufig verarbeitet. Zuerst werden die Eingabebitstrings zu einem einzigen langen Bitstring konkateniert. Anschließend wird dieser in Ausgabebitstrings einer festgelegten Bitlänge zerbrochen, die schließlich als Zahlen einer beliebigen Basis mit entsprechend vielen Zahlsymbolen pro Ausgabewort in eine Datei geschrieben werden. Für jede der ersten beiden Verarbeitungsstufen gibt es zwei Betriebsmodi, die insgesamt vier verschiedene Betriebsarten ermöglichen. Die dritte Stufe, die Ausgabeformatierung, kann durch Parameter gesteuert werden. Der erste Modus erlaubt es, die Reihenfolge der Eingabedaten zu vertauschen, der zweite hat Einfluß auf die Interpretation der Ein– und Ausgabedaten.

Der erste Modus ist default–mäßig auf *forward* eingestellt, die Eingabedaten werden in der Reihenfolge verarbeitet, wie sie gelesen werden. Mit dem Schlüsselwort *reverse* kann auf die umgekehrte Eingabereihenfolge der Eingabebitstrings umgeschaltet werden; *forward* stellt die normale Reihenfolge wieder her. Der Modus kann beliebig häufig gewechselt werden.

Mit den Schlüsselworten *little_endian* und *big_endian* wird der zweite Modus festgelegt. Er steuert sowohl für die Eingabe– als auch die Ausgabebitstrings, ob sie im Little– oder Big–Endian Format zu interpretieren sind. Für die Eingabebitstrings steuert er dabei, in welcher Reihenfolge sie konkateniert werden sollen. Wenn man die Eingabedaten als Bitstrings betrachtet, entspricht der default–Modus *little_endian* lediglich einem Aneinanderrücken der Bitstrings in derselben Reihenfolge, wie sie in der Eingabe auftauchen. Dieser Gesamtstring ergibt sich dann auch wieder aus den Ausgabebitstrings, wenn sie aneinandergereiht werden. Nach dem Befehl *big_endian* wird die Reihenfolge genau vertauscht, der zuletzt eingegebene Bitstring taucht zu Beginn des Gesamtstrings auf. Dieser wird dann von seinem Ende her beginnend herausgeschrieben, sein höchstwertigstes Wort folgt also wiederum zuletzt. Haben alle Eingabebitstrings dieselbe Länge wie die Ausgabebitstrings, bleibt eine Änderung dieses Modus ohne Auswirkung. Ein Wechsel dieses Modus ist, da er auch die Ausgabereihenfolge wechselt, nur nach Bitstrings, mit denen Ausgabeworte enden, möglich (während big–endian Ausgaben werden die Ausgabeworte auf einem Stack zwischengespeichert).

Die Formatiermodi des Bitstringverarbeitungsprogramms *gen_rom* lassen sich am einfachsten anhand einiger Beispiele verdeutlichen. Dazu werden die Zahlen 511, 63 und 7, die als 12–, 8– bzw. 4–Bitzahl vorliegen konkateniert. Die folgenden zwei Tabellen stellen die Eingaben und die jeweiligen Ergebnisse für alle Kombinationen der beiden Modi gegenüber. Als Ausgabeformat wurden hier wieder Bitstrings gewählt, um alle Werte problemlos vergleichen zu können. In der ersten Spalte der zweiten Tabelle wurde die Ausgabewortbreite so groß gewählt, daß keine Zerlegung des Gesamtbitstrings nötig war, und dieser somit direkt als Ergebnis ausgegeben wurde. Da in diesem Fall gar keine Ausgabeinterpretation als little– oder big–endian formatierte Daten stattfindet — es wird ja nur ein Wort ausgegeben —, bewirken beide Modi jeweils nur ein Vertauschen der Eingabedaten. Da doppeltes Vertauschen wieder den Ursprungszustand herstellt, ist die 24–Bit formatierte Ausgabe, wenn beide Modi gewechselt werden, wieder identisch (vgl. linke Spalte: Zeile 1 und 4 bzw. 2 und 3). Für den rechten Tabellenteil wurde eine byteformatierte Ausgabe gewählt. Hier zeigt sich z. B. auch, daß ein gleichzeitiges Wechseln beider Modi eine Vertauschung der Ausgabereihenfolge der Ausgabebytes bewirkt:

Eingabemodi für: "000111111111"; "00111111"; "0111";

1.		little_endian;	511:12; 63:8; 7:4;	
2.		big_endian;	511:12; 63:8; 7:4;	
3.	reverse;	little_endian;	511:12; 63:8; 7:4;	forward;
4.	reverse;	big_endian;	511:12; 63:8; 7:4;	forward;

	Ausgabe als 24–Bitstring	Byteformatierte Ausgabe
1.	"000111111111001111110111"	"00011111", "11110011", "11110111",
2.	"011100111111000111111111"	"11111111", "11110001", "01110011",
3.	"011100111111000111111111"	"01110011", "11110001", "11111111",
4.	"000111111111001111110111"	"11110111", "11110011", "00011111",

Der Befehl *align* erlaubt es, das letzte angefangene Ausgabewort auf eine anzugebende Anzahl von Bits aufzufüllen. Typischerweise entspricht sie der Anzahl von Bits pro Ausgabewort. Wählt man ein Vielfaches, kann jedoch auch z. B. bei einer *byte*–orientierten Ausgabe auf 16–Bit Worte aufgefüllt werden (*word–aligned*). (Nach einem *align*–Befehl ist ein Wechsel des Endianmodus mittels *little_endian* oder *big_endian* möglich.) An Stelle des Befehls werden die Nullbits so eingefügt, als stünden sie an seiner Stelle. Das vorige Beispiel soll auch hier — entsprechend modifiziert — zur Veranschaulichung dienen:

Eingabemodi für:		“0111111111”; “0111111”; “0111”; align;	
1.		little_endian; 511:10; 63:7; 7:4; align;	
2.		big_endian; 511:10; 63:7; 7:4; align;	
3.	reverse;	little_endian; 511:10; 63:7; 7:4; align;	forward;
4.	reverse;	big_endian; 511:10; 63:7; 7:4; align;	forward;

	Ausgabe als 24–Bitstring	Byteformatierte Ausgabe
1.	“011111111101111110111000”	“01111111”, “11011111”, “10111000”,
2.	“000011101111110111111111”	“11111111”, “11111101”, “00001110”,
3.	“000011101111110111111111”	“00001110”, “11111101”, “11111111”,
4.	“011111111101111110111000”	“10111000”, “11011111”, “01111111”,

Der Befehl *address*, gefolgt von einem Konstantenbezeichner, ist trotz Leerzeichens als *ein* Konstantenname aufzufassen. Er darf nach allen Eingabebitstrings stehen, mit denen ein Ausgabebitstring beendet wurde. Der Konstante wird dann die Adresse des nächsten Ausgabewortes zugewiesen. Um diese Konstanten auch vor ihrer Definition z. B. auch im *CONST*–Definitionsbereich nutzen zu können, ist das Schlüsselwort *2pass* vor *CONST* einzufügen. Es führt zu einem zweistufigen Übersetzungslauf. Allerdings sollten sich die berechneten Adressen im zweiten Lauf nicht von denen des ersten unterscheiden, da dann keine statische Konsistenz der Daten gewährleistet werden kann — sie könnten sich auch bei jedem weiteren Übersetzungslauf ändern. Das Programm bricht in diesem Fall mit einer Fehlermeldung ab.

Der Befehl *paragraph* dient der Formatierung der Ausgabe. An der Stelle seines Auftretens wird in der Ausgabe die aktuelle Zeile beendet und eine Leerzeile eingefügt. Da dies nicht innerhalb eine Ausgabezahl möglich ist, darf der Befehl nur nach vervollständigten Ausgabebitstrings auftreten. Dazu kann vorher ein *align*–Befehl eingefügt werden.

FORMAT–**Umgebung**

Alle Parameter, die der Festlegung des Ausgabeformates entsprechen, werden in einer eigenen Umgebung definiert, die mit dem Schlüsselwort FORMAT eingeleitet wird. Sie liegt zwischen der *CONST*– und der *DATA*–Umgebung. Neben dem verwendeten Zahlsystem lassen sich alle die Zahlen trennenden Ausgabestrings frei definieren und zudem die Ausgabe in eine andere Datei einbetten. Damit gelingt es z. B., die Daten eines ROMs VHDL–konform aufzuzählen und direkt in eine VHDL–ROM–Beschreibung hineinzukopieren, so daß das ROM unmittelbar danach als VHDL–Komponente übersetzt werden kann.

Zur Festlegung des Ausgabezahlformates kann über Parameter die Anzahl von Bits pro Ausgabesymbol eingestellt werden. Als Ausgabesymbole werden die hexadezimalen Zahlzeichen verwendet, bei mehr als 4–Bit pro Symbol wird im Alphabet weitergezählt. Damit sind binäre Ausgaben (1), oktale (3) und hexadezimale (4) problemlos möglich. Ein zweiter Parameter steuert die Anzahl von Bits, die eine Zahl ergeben soll. Wird hier nicht ein Vielfaches der Symbol–Bitbreite gewählt, so wird die oberste Stelle nur zum Teil genutzt

(z. B. Bytes in Oktaldarstellung: “000” bis “377”). Ein letzter Parameter steuert dann noch die Ausgabeformatierung durch die Angabe, wieviele Zahlen pro Zeile ausgegeben werden sollen.

Um die Ausgabe direkt weiterverarbeiten zu können, läßt sich eine Zeilenpräambel, die als Textstring vor die erste Zahl kopiert wird, ein Zahlentrennstring und ein Zeilenepilog definieren, der hinter die letzte Ausgabezahl der Zeile kopiert wird. Der Zeilenabschlußstring ist getrennt zu definieren, da er auch bei Einsatz des Formatierbefehls *paragraph* ohne den Zeilenepilog zum Einsatz kommt. Neben der rechnertypischen newline ($\backslash n$) und return ($\backslash r$) Kombinationen, können hier auch Kommentarzeichen eingefügt werden, falls der Ergebniscode nachher noch kommentiert werden soll. Der entstandene Ausgabetext kann zudem optional an einer beliebigen Stelle einer anderen Datei eingefügt werden. Neben dem Namen einer solchen template–Datei und der Ausgabedatei wird mit einem Parameter dazu die Anzahl der Bytes angegeben, nach der die Ausgabe eingefügt werden soll.

Kommandozeilenparameter

Alle Optionen dienen nur zum Debuggen des Eingabecodes oder um den Grund für unerwartete Ergebnisse klären zu können. Die Option *–h* liefert folgende Optionsauflistung:

gen_rom [filename] [–h] [–2pass] [–info] [–symb] [–syn] [–bin] [–expr]

Die Option *–2pass* dient demselben Zweck wie das ensprechende Schlüsselwort am Anfang der Eingabedatei. Die *–info* Option gibt die Werte aller Adreßkonstanten und alle Konstanten, die ihren Wert im zweiten Übersetzungslauf geändert haben aus. Der Parameter *–symb* gibt nach jedem Lauf eine Symboltabelle aller Konstanten aus. Der *–syn* Parameter legt die interne Verarbeitung der Bitstrings offen. Die Option *–bin* legt eine Datei mit gleichnamiger Endung an, die den gleichen Inhalt wie die *DATA*–Umgebung besitzt. An Stelle von Zahlwerten oder Konstanten werden hier jedoch die entsprechenden Bitstrings aufgeführt. Das erste Wort des alten Zeileninhaltes folgt danach als Kommentar, um die Orientierung zu erleichtern. Die Option *–expr* führt schließlich dazu, daß das Parsen aller Ausdrücke kommentiert am Bildschirm verfolgt werden kann.

7.2.2 Treibersoftware: *get_para*

Dieses Programm dient der Eingabe des gewünschten Timings der Pixelströme und der Definition des Operators selbst. Soweit mehrere ASICs kaskadiert vorliegen, um einen Operator mit größerer Umgebung zu realisieren, wird ihre gemeinsame Konfiguration unterstützt. Das Programm entbindet den Benutzer nicht von allen Überlegungen, die bei der manuellen Erstellung einer ROM–Konfigurationsdatei für das Programm *rom_gen* anzustellen sind, sondern eher von lästigen Standardaufgaben.

7.2.3 Leistung der Software

Zur Leistung der Software zählt erstens das optimale Mapping der Pixelströme auf die Pixelstromgeneratoren, von denen wegen der Optimierung nur die Hälfte von jeder zweiten Produktzelle abwechselnd zu erreichen ist. Damit wird erreicht, daß fast immer alle 40 Koeffizienten des ASICs zur Verfügung stehen. Speziell bei zwei Teilmatrizen, die verschiedene Pixelströme benötigen, zahlt sich hier die vollständige Suche einer Lösung aus.

Zweitens kontrolliert die Software, ob das Umgebungsfenster für alle gewählten Unterbildbereiche noch klein genug ist, damit die Randwerteinblendung funktioniert. Dies ist immer dann der Fall, wenn die Umgebung höchstens zum Zentralpixelbildbereich benachbarte Bildfelder überdeckt. Dabei ist u. a. zu beachten, daß eine Matrix, die über den rechten Gesamtbildrand ragt, eine Zeile mehr überdeckt, als sie hoch ist.

Als drittes wird eine gültige Konfiguration der Unterbildbereichszähler berechnet. Eine konsistente Initialisierung liegt genau dann vor, wenn alle Zähler einem bestimmten Pixel, der ja in allen Pixelströmen zu irgendeiner Zeit vorkommt, jeweils dieselben Zählbits für den vertikalen und horizontalen Bildbereich zuordnen. Um dies zu erreichen, wird hier von dem Zählbitzustand aus, den der Bildbereichszähler des aktuellen Pixelstroms direkt nach dem Reset besitzt, zu allen anderen Pixeln zurückgezählt, die zu diesem Zeitpunkt von den Pixelstromgeneratoren geliefert werden oder den Zentralpixel verkörpern. Bei jeder Bildgrenze, die dabei überschritten wird, werden die Zählbits für die vertikalen und horizontalen Bildgrenzen getoggelt. Mit den Ergebniszählbits für ihre reset–Pixelposition werden dann die entsprechenden Bildbereichszähler initialisiert. Die Idee dabei ist, daß die Zähler im Betrieb bis zum Abliefern des aktuellen Pixels, der ja verzögert wird, wieder genausooft toggeln und danach alle die Zählbits für den ehemals aktuellen Pixel liefern, die sein Zähler nach dem reset geliefert hat. Dies Verfahren würde übrigens auch bei Zählern mit mehr als einem Bit oder — in der Modellvorstellung — mit Integern funktionieren, die dann mit einem negativen Offset zum reset–Zählerstand des aktuellen Pixelstroms zu initialisieren wären. Weiterhin muß die Verzögerung für die Ergebnisse kaskadierter ASICs berücksichtigt werden. Sie müssen der Zeit des Ausgabe–ASIC in jeder Beziehung um entsprechend viele Takte voraus sein, also auch mit unterschiedlichen Zentralpixelwerten initialisiert werden.

Die vierte Aufgabe besteht darin, Besonderheiten einzelner Konfigurationen zu verbergen. So wird bei der signed–Ausgabe das niederwertigste Bit nicht ausgegeben, vor den zweiten Shiftern werden in der Ausgabeformatierung zwei Nullbits als niederwertigste Bits hinzukonkateniert und der Zentralpixel wird beim Gesamtmatrix–Modus in die höchstwertigsten Bits des zweiten Verarbeitungszweiges eingespeist. Dazu wird ein Verarbeitungsweg ohne solche Besonderheiten angenommen, und deren Auswirkungen durch eine Transformation der Eingabewerte der Shifter etc. berücksichtigt. Wenn der erste Shifter im zweiten Verarbeitungszweig im Gesamtmatrixmode um -5 bis 0 Stellen nach rechts schieben kann, verbirgt sich dahinter, daß der Zentralpixel in dieser Konfiguration an die Bits fünf bis fünfzehn des Shifters angeschlossen ist, also schon vorweg um fünf Bits nach links, oder formal um -5 nach rechts, geschoben wurde.

7.2.4 Eingabemodi und Parametrisierung

Das Programm liest entweder eine selbsterstellte eventuell modifizierte Log–Datei wieder ein oder erfragt in einem Eingabedialog alle benötigten Daten. Ersteres Vorgehen ist dabei komfortabler. Sind aufgrund der modifizierten Eingaben alle sonstigen noch gültig, wird die Eingabedatei in eine ROM–Konfigurationsquelldatei für *gen_rom* übersetzt. Überall, wo sich der Eingabedialog geändert hat — z. B. aufgrund geänderter Wertvorgaben —, wird der alte und der neue Dialogtext als Warning einander gegenübergestellt. Da sich hierbei auch die Position der Eingabedaten geringfügig ändern kann, werden sie immer in einer gewissen textuellen Umgebung der Position gelesen, an der sie in der neuen Log–Datei auftauchen. Hat man sich vergewissert, daß dabei keine Mißverständnisse aufgetreten sind, kann man durch eine Option die alte Log–Datei von der neuen überschreiben lassen.

Die Restriktionen, die die Hardware vorgibt, äußern sich darin, daß zu jeder Eingabe ein Wertebereich angegeben wird, in dem Eingaben möglich sind. Daraus ergibt sich auch die Eingabereihenfolge von Koordinaten, die zuerst eine Angabe des Y– und dann des X–Wertes vorsieht. Abhängig von der Zeilenangabe ist dann die Eingabe des X–Wertes beschränkt. Da Matrizen über ein Bild hinausragen können, wurde zudem die Koordinateneingabe noch um einen Z–Wert erweitert; dieser steht für die Bildnummer, in der die Koordinateneingabe danach erfolgt.

Die Treibersoftware ist selbstverständlich ebenso wie das ASIC voll parametrisierbar. Änderungen in der ASIC–Parametrisierung müssen nur im Header einer zum Programm zugehörigen Datei *get_para.tpl* nachgetragen werden. Diese Datei erfüllt zwei Aufgaben, sie dient einerseits dem Einlesen der ASIC–Parametrisierung und beinhaltet andererseits Textschablonen zum Generieren der Datei, die als Eingabe für *gen_rom* dient. Zur iterierten Ausgabe solcher Schablonen und an den Stellen, an denen die Werte einzutragen sind, enthält die Datei einige Steuerzeichen. Diese Zeichen dürfen keinesfalls vertauscht werden, da das Programm beim Einlesen eines unerwarteten Steuerzeichens abbricht. Außerdem sind hier einige Formeln zur Berechnung von Konstanten angegeben, die ebenfalls im Programm auftauchen. Werden hier einseitige Änderungen vorgenommen, können inkonsistente Eingabedateien für das Programm *gen_rom* generiert werden, das dann mit einer Fehlermeldung abbricht.

Sollte eine Parameteränderung eine Änderung der Arraygrenzen innerhalb des C–Programms notwendig machen, gibt das Programm eine entsprechende Meldung aus. Nach einer Änderung im Header und einer erneuten Compilation ist die Treibersoftware wieder einsatzbereit. Die aktuellen Parametereinstellungen werden als Konstanten in den Konfigurationsquellcode für *gen_rom* aufgenommen und lassen sich somit bei aktivierter Symboltabellenoption mit denen des ASIC–Entwurfs vergleichen. Abb. 7.7 zeigt die Parameterdefinition in der Datei *get_para.tpl*, in der nur die unabhängigen Parameter auftauchen (abhängige Parameter lassen sich zwar prinzipiell aus den unabhängigen berechnen, jedoch nicht in VHDL, das keine Logarithmusfunktion kennt). Im VHDL–Code selbst werden alle Parameter, die unabhängigen und die abhängigen, aufgezählt. Sie müssen unbedingt mit denen übereinstimmen, von denen die Treibersoftware ausgeht und die von *gen_rom* optional als Symboltabelle ausgegeben werden (Pfeil “1” in der Abb. 7.7). Dies sollte genau dann der Fall sein, wenn die unabhängigen Parameter gleich gewählt wurden (gestrichelte Linie). Inkonsistente Parametrisierungen äußern sich in ebensolchen Konfigurationen und damit falschen Filterergebnissen, unendlich langen Initialisierungsvorgängen und Wertbereichsverletzungen. Letztere werden bei der Simulation durch assert–Statements gemeldet oder führen dazu, daß der VHDL–Simulator die Simulation wegen inkompatibler Bitvektorzuweisungen abbricht, bzw. assertation–Verletzungen beim Vorgang elaborate (der Instantiierung generischer Komponenten) auftreten.

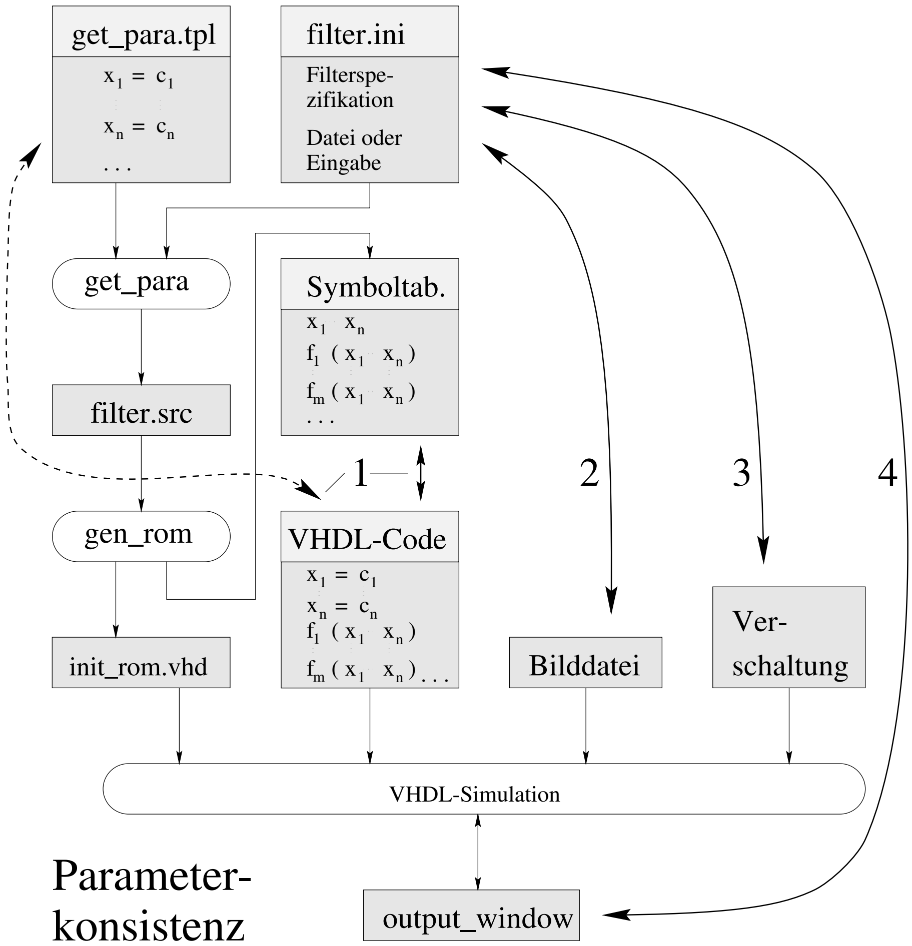


Abbildung 7.7: Abhängigkeiten der Parameter

Abb. 7.7 gibt neben dieser Konsistenz der Parametrisierung von ASIC und Treibersoftware auch noch die Parameter an, die konsistent zu den Angaben der Konfigurationsdatei sein müssen, um eine korrekte Filterung erwarten zu dürfen. Pfeil “2” deutet auf den Umstand hin, daß die Bildabmessungen einer eingelesenen Realbilddatei, mit den in der Konfigurationsdatei angegebenen Werten übereinstimmen müssen. Die Verschaltung von kaskadierten ASICs und die Vergebung der individuellen ASIC–ID Werte muß konventionsgemäß erfolgen. Das Programm geht von einer Verschaltung der ASICs nach Abb. 6.10 aus (Pfeil “3”), wie man sie auch in den vier Testumgebungen kaskadierter ASICs vorfindet (s. Abschnitt 7.4). Die synchrone Bilddarstellung der Eingabe– und der Ausgabebilder ist nur gewährleistet, wenn nach der Konfiguration der Pixel mit der Koordinate (0,0) am ASIC–Eingang erwartet wird. Die Koordinate des Ausgabepixels ist in den beiden *top–level* Simulations–VHDL–Dateien *test_image* und *test_picture* im Header als Ergebnisoffsetwert einzutragen (Pfeil “4”).

7.2.5 Eingabewerte

Bild– und Konfigurationsdefinition Zu Beginn müssen im direkten Eingabedialog die Höhe und Breite des Gesamtbildes eingegeben werden; anschließend werden Unterbildbereiche abgefragt, bis das Gesamtbild vollständig aufgeteilt ist. Wird keine Untereinteilung gewünscht, ist einfach die rechte untere Bildkoordinate einzugeben. Direkte Verzögerungsangaben tauchen im Eingabedialog nur an einer Stelle auf. Ansonsten werden nur Bildkoordinaten eingegeben, die für die reset–Konfiguration und nach jedem *xy_reset* gelten. Um daraus intern Verzögerungen zu berechnen, muß die aktuelle Position angegeben werden, die nach einem Reset für das erste einlaufende Pixel eingenommen werden soll. Diese Koordinate hat insofern eine besondere Bedeutung, als alle anderen Koordinaten vor dieser liegen müssen (alle anderen Pixel wurden verzögert, d. h. sie waren vor dem reset–Zeitpunkt aktuelle Pixel; somit liegen ihre Koordinaten vor der hier eingegebenen). Anschließend erfolgt die Eingabe der Anzahl von ASICs, die zur Berechnung eines Operators kaskadiert werden sollen. Alle folgenden Eingaben sind für jedes der kaskadierten ASICs einzugeben.

Man muß nun jedes erste ASIC pro horizontal kaskadierter ASIC–Gruppe zum SRAM–Master erklären. Der SRAM–Master schreibt als einziger auf den gemeinsamen SRAM–Bus horizontal kaskadierter ASICs. Da dies nur eine Vorsichtsmaßnahme darstellt — alle würden ohnehin dasselbe schreiben —, spielt es keine Rolle, welchem ASIC man den Schreibzugriff gestattet. Besitzt jedes ASIC seinen eigenen Zeilenspeicher, sind alle ASICs SRAM–Master. (Die Antwort steuert auch, ob von horizontaler Kaskadierung zu den vorherigen, oder von einer vertikalen Kaskadierung ausgegangen wird.)

Die Chip–ID ist bei einer Hardwarerealisierung durch entsprechende pullup– bzw. pulldown–Widerstände am Ausgang result_b jedes ASICs einzustellen, die zum Ende des Resets detektiert werden. Sie steuert außerdem noch, an welches ASIC parallel zum Zeilenspeicher das Konfigurations–ROM angeschlossen ist.

Pixelstromdefinition Anschließend muß die Anzahl der benötigten Pixelströme angegeben werden. Wählt man die maximale Anzahl, so wird auch der aktuelle Pixelstrom benutzt. Dies führt bei der folgenden Koodinateneingabe der Pixelströme zu einer festgelegten Eingabe der Koordinate des letzten Pixelstroms, wenn nicht vorher schon ein solcher eingegeben wurde, der keine Zeilenspeicherverzögerung benötigt. Wählt man weniger Pixelströme als möglich, kann man dagegen nun für jeden beliebige Resetkoordinaten eingeben.

Genau genommen werden zuerst die Koordinaten eingegeben, die der Zeilenspeicher generiert, danach diejenigen, die in die Koeffizientenmatrix eingespeist werden. Bei horizontal kaskadierten ASICs entfällt diese erste Eingabe, da das Timing des Zeilenspeichers mit dem ersten horizontal kaskadierten ASIC bereits festgelegt wurde. Stattdessen muß die globale Verzögerung des Zwischenergebnisses bei der Kaskadierung angegeben werden. Das Minimum beträgt drei Takte, das Maximum ist abhängig von der Parametrisierung des Kaskadierungsschieberegisters und beträgt hier zehn Takte.

In jedem Fall müssen jetzt noch die Eingabepixel der Matrix festgelegt werden. Die Differenz der im letzten Schritt eingegebenen Zeilenspeicherkoordinaten zur maximal möglichen Koordinateneingabe beruht auf der (erwünschten) Kaskadierungsverzögerung. In der Ausdehnung des nun möglichen Koordinateneingabebereichs äußert sich die Existenz erweiterter Ausgangsschieberegister der Zeilenspeichereinheit. Diese können länger als die zur Beseitigung der Zeilenspeichergranularität von vier Pixeln benötigten Länge von drei Stufen parametrisiert werden. Hier wurden sie um drei Pixel erweitert. Dies erlaubt bei einer horizontalen Kaskadierung, daß auch bei gemeinsam genutztem Zeilenspeicher, also den gleichen zuerst eingegebenen Master–Koordinaten, einzelne Pixelströme ASIC–individuell um bis zu drei Pixel verzögert werden können. Damit lassen sich trotz globalem Versatz durch die Verzögerung des kaskadierten Zwischenergebnisses an Matrixrändern noch bis zu drei Koeffizienten pro Matrixzeile einsparen. Werden nicht kaskadierte ASICs, horizontal kaskadierte ASICs auf rechteckigen Operatorumgebungen oder kaskadierte ASICs mit individuellen Zeilenspeichern eingesetzt, bietet diese Eingabe nur einen überflüssigen Freiheitsgrad.

Als nächstes ist der Zentralpixel zu definieren. Dazu werden der Pixelquellstrom der Matrixzeile, in der er enthalten ist, und seine Koordinaten eingegeben. Bei kaskadierten ASICs muß der Zentralpixel vom Ausgabe–ASIC erreicht werden, jedenfalls wenn sein Wert bei der Ausgabe benutzt werden soll. Alle anderen ASICs benötigen seinen Wert nicht, können meist sogar nicht einmal auf ihn zugreifen. Daher kann mit der Eingabe null auf einen virtuellen Zentralpixel umgeschaltet werden. Die nachfolgende Koordinateneingabe wird dann nur zur Initialisierung des Zentralpixelzählers benötigt. Dessen Koordinate sollte für alle kaskadierten ASICs identisch sein.

Matrizendefinition Es besteht die Wahl einer Gesamtmatrix oder zweier Teilmatrizen. Für jede Matrix wird die Anzahl der besetzten Koeffizientenzeilen abgefragt und dann der Eingabestrom für jede Zeile festgelegt. Über die Größe der Matrix selbst ist damit noch nichts ausgesagt. Sie könnte beliebig groß und dünn besetzt sein oder auch nur eine Zeile mit vielen horizontal getrennten Koeffizientenbereichen besitzen. Daher steht bei der folgenden Angabe, welcher Matrixzeile die Koeffizientenzeile entspricht, ein Matrixzeilenbereich der Größe des Zeilenbereiches des Gesamtbildes zur Verfügung. (In den meisten Fällen besteht eine 1:1–Zuordnung zwischen Matrix– und Koeffizientenzeilen.)

Auf eine automatische Zeilenzuordnung anhand der Pixelstromkoordinaten wurde verzichtet. Bei einer nichtzusammenhängenden Matrix, die in einer Zeile getrennte Koeffizienten mit dem Abstand einer halben Gesamtbildbreite besitzt, wäre sie mehrdeutig! Man könnte nicht entscheiden, ob die Matrix direkt oder über den rechten Gesamtbildrand gesehen zwischen ihnen liegt. Wäre dann einer der beiden Koffizienten auch der Zentralpixel, hätte man dafür noch nicht einmal einen Anhaltspunkt. Eine unsinnige Zeileneingabe führt formal zu einer über mehrere Bildbreiten ausgedehnten Matrix (Zeilenwechsel innerhalb einer Matrixzeile sind grundsätzlich erlaubt, um reset–Konfigurationen, in denen Matrizen über Bildrändern liegen, zu ermöglichen). Damit ist jedoch die notwendige Bedingung für den Randwerteinblendungsalgorithmus verletzt. Dies wird durch spätere Vorgabe des Wertes null für entsprechend weit vom Zentralpixel entfernte Koeffizienten quittiert. So können sie dann das Ergebnis der restlichen Koeffizienten nicht verfälschen, womit die Konfiguration wenigstens aus formaler Sicht *gerettet* und gültig ist. Wird man von auf null festgelegten Koeffizienten überrascht, sollte man also auch an dieser Stelle die korrekte Zeilenzuweisung kontrollieren.

Koeffizientendefinition Die Positionen aller ersten nach einem Reset mit den Koeffizienten verknüpften Pixel steht nun fest. Zur Sicherheit werden sie, ebenso wie die Koeffizientenzeilenzuordnung und die Matrix, zu der der Koeffizient gehört, als Eingabeaufforderung des Koeffizientenwertes ausgegeben. Der Wertebereich erlaubter Eingaben ergibt sich entweder durch die Koeffizientenwortbreite oder ist konstant null — letzteres, wenn ansonsten eine korrekt verlaufende Randeinblendung nicht gewährleistet wäre. Dies wird für jeden Koeffizienten einzeln geprüft. Kritisch sind kleine Unterbildbereiche. Werden Koeffizienten nur vor oder nach der Zentralpixelspalte nullgesetzt, liegen die Probleme darin, daß eine Matrix, die über dem horizontalen Gesamtbildrand liegt, eine Zeile mehr überdeckt als sie hoch ist. Zumindest tun dies die Koeffizienten links oder rechts vom Zentralpixel, weswegen sie in diesem Falle nullgesetzt wurden. Falls der Zentralpixel nicht virtuell (s. o.) ist, wird sein Koeffizient ebenfalls im Anschluß erfragt.

Ausgabeaufbereitung Die restlichen Eingaben beziehen sich auf die Ausgabeaufbereitung. Nachdem die Ergebnisse im ASIC mit voller Wortbreite korrekt berechnet vorliegen, sind die folgenden Parameter dafür verantwortlich, daß sie auch korrekt zu den Ausgängen durchgeschaltet werden und nicht auf dem Weg dorthin im Zuge der Bitbreitenreduktion wegen einer Übersteuerung zu den Extrema null oder 255 bzw. -128 oder 127 entarten. Welche Parameter hier erfragt werden, hängt von der restlichen Konfiguration ab; nicht alle hier vorgestellten Optionen werden daher bei einem realen Dialog erscheinen.

Für jede Matrix kann zuerst der Verknüpfungsmodus eingestellt werden. Mit *sum* wird gefaltet, mit *max* wird dilatiert und mit *min* schließlich erodiert. Für die Rangordnungsvarianten sind die vorher bereits genannten Regeln zur Koeffizientenwahl zu befolgen (null spielt eine Sonderrolle zur Konfiguration einer unbenutzten Koeffizientenzelle). Im folgenden wird der direkte Randersatzwert eingegeben. Er durchläuft als Konfigurationswert natürlich nicht vorher noch die LUT. Ob die Eingabedaten die LUT durchlaufen, kann im folgenden entschieden werden. Sie zu umgehen kann bei defektem *on–chip* SRAM zur

Erzielung einer kürzeren Initialisierungsdauer sinnvoll sein.

Als nächstes müssen die Offsets der Addierer vor den ersten Shiftern und anschließend deren Shiftwerte eingegeben werden. Eine Eingabe von -4 bewirkt dabei z. B. ein arithmetisches Rechtsschieben um minus vier, also ein Linksschieben um vier Dezimalstellen. Dies entspricht einer Multiplikation mit 16. Negative Werte und verschobene Bereichsgrenzen beruhen, wie oben besprochen, auf Implementationsdetails des ASICs.

Nun wird eingegeben, ob die Werte des Extensionseingangs zum ersten Verarbeitungsstrang verknüpft werden sollen, ob der resultierende Wert zu dem des zweiten Verarbeitungsstranges verknüpft werden soll (Zentralpixelverknüpfung für den zweiten Ausgang) und ob das erste 8–Bit formatierte Ergebnis oder das 16–Bit Kaskadierungszwischenergebnis gemultiplext auf dem ersten 8–Bit Ausgang ausgegeben werden soll. Die Verknüpfungsmodi sind gegebenenfalls danach einzugeben — ebenso wie bei den Matrizen.

Anschließend erfolgt die Eingabe des gewünschten Ausgabeformates, als *signed*– oder *unsigned*–Integer, der letzten Offsetkorrektur vor der Ausgabe, des Schiebewertes des nachfolgenden Shifters und des die Ausgabeformatierung abschließenden Multiplizierers. Er dient nur der vollen Ausnutzung der ausgegebenen obersten 8–Bits. Sein Wertebereich erstreckt sich folglich nur über eine Oktave.

Alle Pixelpositionen, die eingegeben wurden, beziehen sich auf die Pipelinestufe, in der bei Bedarf der Randwert eingeblendet wird. Um die Konfiguration zu erleichtern, wurde das ASIC so entworfen, daß die vom ASIC zu verantwortende parasitäre Pipelineverzögerung unabhängig von der Konfiguration konstant ist. Die Distanz vom aktuellen Eingabepixel zum Zentralpixel, die bis zur Ergebnisausgabe prinzipiell abgewartet werden muß (da vorher noch nicht alle benötigten Daten für das Ergebnis vorliegen), muß zu der Pipeliningverzögerung der Ausgabe addiert werden. Außerdem existiert vor besagter Verarbeitungsstufe eine parameterabhängige, ansonsten jedoch konstante Pipeliningverzögerung der Eingabe. Um mehrere Verarbeitungsoperationen ohne Puffer hintereinander ausführen zu können, ist es jedoch wesentlich, welche Pixel nach dem Resetzeitpunkt an die Pins des ASICs gelegt werden müssen, und welche er selbst zu diesem Zeitpunkt gültig als Ausgabe anlegt. Daher werden für das Ausgabe–ASIC diese Werte jetzt berechnet und ausgegeben. Die Distanz zu den Koordinaten des aktuellen Pixels beträgt bei der jetzigen Parametrisierung sechs Takte, was schon bei der Eingabe berücksichtigt werden sollte. Das Ergebnis, das zu diesem Zeitpunkt geliefert wird, ist dasjenige der Daten, die 22 Takte plus der oben genannten prinzipiellen Verzögerung früher anlagen; entsprechend liegt die Ausgabekoordinate vor der Eingabekoordinate.

Kommandozeilenparameter

Das Programm bietet ebenso wie *gen_rom* einige Optionen zum Debuggen der internen Verarbeitung an. Die Option *–h* liefert für *get_para* folgende Optionsauflistung:

get_para [–replace] [–sep] [–tmpl] [–info] [–cntr]

Die *–replace* Option ersetzt eine eventuell modifizierte Log–Datei, die wegen abweichender Wertebereiche der Eingaben zu Warnings führte, durch eine neue Log–Datei, die die aktuellen Eingabewertebereiche enthält. Mit *–sep* kann trotz kaskadierter ASICs jedes ASIC seine eigenen Pixelausgänge nutzen, was nur zu Testzwecken sinnvoll ist. Die Option *–tmpl* gibt die während der Verarbeitung aus der Datei *get_para.tpl* eingelesen Strings aus, womit Fehler durch falsch positionierte Steuerzeichen innerhalb dieser Datei geortet werden können, die *–cntr* Option dient zur Ausgabe der getesteten Verteilung der Pixelströme auf die Ausgänge der Komponente *multi_queue* und mit *–info* lassen sich schließlich noch diverse internen Variablen anzeigen, die den Eingabewertebereich beeinflussen.

7.3 Komponententests

Im Verlauf der ASIC–Entwicklung wurden die meisten Komponenten der Hauptblockebene bereits ausführlich für sich getestet. Dieses geschah durch Simulation, während der die interessierenden Signale ge–*traced* wurden — ihr Signalverlauf also über die komplette Simulationszeit protokolliert und als sog. *waveforms* dargestellt wurde. Die zu testende Komponente wurde in einer Testumgebung instantiiert, die die benötigten Simulationsstimuli durch Prozesse generierte. Den Eingangsignalen der zu testenden Komponente wurden dazu zur gewünschten Zeit die benötigten Werte zugewiesen. So wurden einige Dutzend Multiplikationsergebnisse der Multiplizierkomponente *product* nachgerechnet, Verzögerungen der Zeilenspeicherkomponente *multi_queue* durch Vergleich von verzögerten Pixelströmen, deren Wert stetig inkrementiert wurde, verifiziert und der Zustand der Zählbits der Bildbereichszähler *sub_pics_counter* nachvollzogen.

Im Gesamtdesign wurde dann noch die Randwerteinblendung innerhalb der Koeffizientenzellen *product_cell* kontrolliert ebenso wie die Werte aller Zwischenergebnisse der Komponente zur Ausgabeformatierung *front_end*. Die Steuersignalverläufe während der Initialisierungsarbeit der Komponente *lut* wurden mit den vorher erstellten Sollwerten verglichen, und einige Ergebnisse der Summierbäume *summ_tree* nachgerechnet. Nachdem alles im Einzelnen verifiziert war, wurden eine Konfiguration per Hand als ROM kodiert und die Kontrolle der Filteroperationen als Ganzes durch Anzeige kleiner, mit dem simulierten ASIC gefilterter, Bilder fortgesetzt. Die dabei vorgefundenen unsinnig gefilterten Bilder waren fast immer das Ergebnis von Kodierungsfehlern des Konfigurations–ROMs, die sich nur mühsam durch Einsatz von *hot–spot*–Matrizen hervorheben ließen, die nur einen Koeffizienten beinhalten und damit dessen Ausblendung an Rändern veranschaulichen. Daraufhin wurde die Treibersoftware entwickelt.

Mit dieser wurden nun verschiedene Binomial–, Sobel–, Dilatations– und Erosionsfilter getestet, wobei die letzten beiden die Notwendigkeit einer Sonderrolle des Koeffizienten 0 bei Filterungen ihrer Art offenbarten. Nach diesen Tests wurden die Testumgebungen jeweils zweier kaskadierter ASICs geschrieben und die Kaskadierungsschnittstelle getestet (s. Abschnitt 7.4). Als Bildeingabe wurden zuerst horizontale und vertikale Graurampenbilder, dann Realbilder und diagonale Graurampenbilder eingesetzt. Zum Schluß wurde das ASIC noch einige Male umparametrisiert und jeweils mit denselben Bildern getestet. Sowohl mit neun, fünf und schließlich sieben Pixelströmen bei 64, 25 bzw. 40 Koeffizienten funktioniert das umparametrisierte ASIC und die Treibersoftware auf Anhieb.

Da hier eine Abhandlung der simulierten *waveforms* einen viel tieferen Einstieg in die VHDL–Beschreibung erfordern würde, wurde darauf verzichtet. Als Funktionsbeleg des ASICs sei hier ein reales Originalbild und die Filterung mit einem $3 * 3$ Sobelfilter in X– und in Y–Richtung im Anhang aufgeführt (s. Abb. D.11 ff.). Die Verarbeitung erfolgte durch den als VHDL–Beschreibung simulierten ASIC. Da dies sehr viel Zeit beansprucht, wurden nur sehr kleine Bilder verwendet, und die einzelnen Pixel vergrößert dargestellt. Die beiden Ergebnisbilder wurden zeitgleich vom ASIC als Ergebnis je einer Teilmatrix berechnet und ausgegeben.

7.4 Simulation

Für das ASIC wurden drei Testumgebungen in VHDL beschrieben. In der ersten wurde ein einzelnes ASIC instantiiert. In der zweiten und dritten Testumgebung sind zwei ASICs mit gemeinsamem Zeilenspeicher (horizontal–) und mit individuellem Zeilenspeicher (vertikal–) kaskadiert aufgebaut. Die Pixelstrom–Eingabe und die beiden Ergebnisausgänge wurden mittels einer vom Fachbereich zur Verfügung gestellten Komponente ([*Hendrich 1994*]), die Integervektoren als Grauwertzeilen in einem X–Window–Fenster ausgibt, graphisch dargestellt. Das erste ASIC liefert bei der kaskadierten Textumgebung seine Ausgabe über die Kaskadierungsschnittstelle an das zweite, dessen Ausgabebilder nun in den Ausgabefenstern erscheinen. Zu debug–Zwecken existiert die zweite und dritte Testumgebung noch in einer Version, in der die ersten Ausgänge beider ASICs angezeigt werden. Dazu muß das erste ASIC allerdings seine Daten über den Ausgang normal und nicht im kaskadierten Datenformat ausgeben; eine Option der Treibersoftware läßt diese Test–Konfiguration ausnahmsweise zu.

Zur Simulation existieren fünf Batchjobs, die als Eingabe noch eine Konfigurationsdatei benötigen. Zusammen mit dieser wird die Treibersoftware aufgerufen und eine Quelldatei für das nachfolgende Bitstringverarbeitungsprogramm erzeugt. Dieses übersetzt die Quelldatei in eine VHDL ROM–Beschreibung. Anschließend werden die ROM–Beschreibung, je nach Batchjob eine der fünf Testumgebungen und noch zwei verschiedene Top–Level–Komponenten übersetzt. Die erste Komponente liefert dem ASIC ein diagonales Graurampentestbild, die zweite eine Bilddatei als Eingabestrom. Nachdem der Batchjob nun abgearbeitet ist, läßt sich ein Top–Level–Design in den VHDL–Simulator laden und simulieren. Das Graurampenbild hat sich dabei extrem bewährt, da Filterfehler, wie falsch eingeblendete Randwerte etc., als Sprung in der Graurampe sofort auffallen.

Für die Einzel–ASIC–Testumgebung wurde eine Konfigurationsdatei für ein Binomialfilter und eine für Sobelfilterungen geschrieben, wobei im letzten Fall eine Teilmatrix den in X–Richtung und die andere den in Y–Richtung differenzierenden Sobelfilter simultan berechnet (im ersten Fall wird der Zentralpixel auf dem zweiten Ausgang wieder ausgegeben). Für die kaskadierten ASICs wurde nur je eine Konfiguration erstellt, die zwei horizontal bzw. vertikal versetzte Binomialmatrizen berechnet.

Zu Testzwecken wurde der Randeinblendewert auf den Maximalwert festgelegt, um erkennen zu können, wann er eingeblendet wird. Bei den kaskadierten ASICs ist der Zentralpixel z. B. asymmetrisch im Zentrum der ersten Binomialmatrix. Entsprechend wird für die Randwerte der zweiten überlagerten Binomialmatrix auf einer Seite für fast alle Koeffizienten der Ersatzwert eingeblendet, was zu einem seitlichen weißen Balken führt. Die beiden letzten zu debugging–Zwecken implementierten Testumgebungen bedürfen einer tiefergehenden Interpretation. Da das erste ASIC seine Daten verfrüht abliefert — sie werden bei der Kaskadierung durch das zweite ASIC noch verzögert, was bei direkter Ausgabe nicht der Fall ist —, sind die Bildgrenzen der Ausgabe um einige Pixel verschoben. Die Bildgrenzen fallen daher nicht mit den dargestellten überein, und die Randwerteinblendung findet im Bildbereich statt.

Batchjob		Grundkonfiguration	Operatoren
mk_single	single	1 ASIC, 1 Operator	Binomialfilter
mk_single	sobel	1 ASIC, 2 Operatoren	X– und Y–Sobelfilter
mk_du_hor	dual_hor	2 kaskadierte ASICs	horizontal überlagerte Binomialfilter
mk_du_ver	dual_ver	2 kaskadierte ASICs	vertikal überlagerte Binomialfilter
mk_db_hor	db_du_hor	2 kaskadierte ASICs	s. o., aber getrennte Ausgabe
mk_db_ver	db_du_ver	2 kaskadierte ASICs	s. o., aber getrennte Ausgabe

7.5 Systemverbund zur Merkmalsberechnung

Ein System auf Basis der bisher mit dem Bildverarbeitungsprogramm auf dem PC vollzogenen Rechenschritte besteht aus 13 gleichen ASICs (vgl. Abb. 8.2). Es werden allerdings noch vier mehrere Zeilen lange Verzögerungsleitungen benötigt, die kompensieren, daß zur Berechnung der klassifizierenden Merkmale nicht die gleiche Anzahl an Operatoren benötigt werden. Zur Not können diese natürlich auch durch ASICs realisiert werden. Das erste ASIC dient der Eingangsbinomialfilterung, das zweite und dritte der iterativen multigrid–Berechnung der Pyramidenbilder auf einem externen Halbbildspeicher; es werden dabei gleichzeitig die Bilder der Laplace– und Gaußpyramide berechnet. Die restlichen zehn dienen nun direkt der Berechnung der einzelnen klassifizierenden Merkmale.

Die weitere Verarbeitung erfolgt in zwei gleichen Verarbeitungseinheiten. Da durch die Bilder der Pyramide die Fläche aller Bilder gegenüber dem Originalbild um 33% vergrößert ist — aufgrund der nicht kompakten Anordnung der Pyramidenbilder die benötigte Bruttobildfläche sogar um 50% (s. Abb. 5.1) — werden einmal die unterste Pyramidenstufe, im Falle der Gaußpyramide also das Originalbild, verarbeitet und einmal die oberen Bilder der Pyramide, die in einem Pixelstrom untereinander linksbündig erzeugt wurden. Auf jede Verarbeitungseinheit entfallen damit noch je 5 ASICs.

Eines davon dient der Dilatationsberechnung incl. der Differenzbildung zum Zentralpixel. Das über einen Schwellenwert daraus eventuell abgeleitete Binärbild kann, falls eine solche Auswertung benötigt wird, dem Rechner direkt oder erodiert und unterabgetastet zur Detektion von Rissen übergeben werden. Zweimal zwei Verzögerungsleitungen oder ASICs müssen die klassifizierenden Merkmale, die ohne Berechnung schon vorliegen, solange verzögern, wie die Dilatationsberechnung dauert.

Die verbleibenden vier ICs schließlich dienen der Orientierungsbestimmung von Texturen nach [*Jähne 1991*]: Einer dient der Ableitungsbildung in X– und Y–Richtung. Die gemischten Ableitungsprodukte der X– und Y–Differentiale können mit drei ROM–Tabellen berechnet werden, einschließlich ihrer Differenz– bzw. Summenbildung. Die letzten drei ASICs werden zur Gaußfilterung der ROM–Ergebnisse benötigt. Der dreidimensionale Merkmalsvektor kann dann ohne Verluste unterabgetastet (hohe Frequenzen wurden durch die Gaußtiefpaßfilterung eliminiert) und ebenso wie die Astlochdetektionsantwort zur statistischen Auswertung einem Histogrammchip bzw. einem linearen Klassifikator zur direkten Merkmalsdetektion übergeben werden.

Zur korrekten Konfiguration des Systems ist die Pixelverzögerung durch die Pipelinestufen der einzelnen ASICs individuell für jede Operatormatrix zu berücksichtigen. Es ist darauf zu achten, daß die Reset–Koordinaten von verbundenen Ein– und Ausgabeports übereinstimmen, was durch modifizierte Timingvorgaben der Pixelströme erreicht werden kann.

Da bereits ein ASIC in der VHDL–Simulation für winzige Bilder Minuten benötigt und einen Großteil des verfügbaren Arbeitsspeichers belegt, verbietet es sich, ein wie eben geschildertes System als Ganzes simulieren zu wollen. Dabei ist es keine Frage der Rechenzeit, da die Nutzung von virtuellem Speicher, der vielfach größer ist als der reale, die Auslastung der CPU auf Werte unter einem Prozent einbrechen läßt — der Rechner ist praktisch nur noch mit *swapping* beschäftigt. Eine ähnlich aufwendige Simulation des ASICs auf Gatterebene (im Gegensatz zur Simulation auf VHDL–Ebene) wurde nach drei Stunden abgebrochen. Zu diesem Zeitpunkt waren erst 9 ns Realzeit simuliert, mithin noch keine einzige Taktflanke erreicht. Es bleibt also nur die Möglichkeit, einzelne Operatoren auf dem ASIC und das Zusammenspiel weniger kaskadierter ASICs zu testen; beides wurde durchgeführt. Ansonsten ist zwischen ASICs nur einzuhalten, daß die Pixeldaten zur positiven Flanke gültig anliegen, was gewährleistet ist.

Kapitel 8

Diskussion

8.1 Zusammenfassung

Zur automatisierten Güteklassensortierung von Hölzern bedarf es der Echtzeitverarbeitung von Bildern. Aufgrund der variablen Größe zu detektierender Astlöcher und der unterschiedlichen Dichte der Holzmaserung wird eine frequenzselektive Verarbeitung z. B. auf der Basis von Bildpyramiden benötigt ([*Fleischer 1993*]). Als geeignete klassifizierende Merkmale zur Astlochdetektion erwiesen sich der Helligkeitswert des Originalbildes, das bandpaßgefilterte Originalbild und der Bildkontrast zum hellsten lokalen Umgebungspixel. Zur Maserungsbewertung ließen sich der Orientierungsvektor der Textur und die Texturintensität als klassifizierendes Merkmal verwenden. Alle Merkmale wurden durch ein hierzu entwickeltes Bildverarbeitungsprogramm berechnet. Einige zufällig gewählte Vertreter von Objekten der verschiedenen zu detektierenden Objektklassen wurden im Rahmen einer Voruntersuchung im Merkmalsraum dargestellt. Es zeigte sich, daß sie durch einen linearen Klassifikator problemlos zu trennen sein würden.

Zielsetzung

Zur Echtzeitbildverarbeitung müssen alle zur Berechnung mit dem Bildverarbeitungsprogramm eingesetzten Operatoren von Hardwarekomponenten abgedeckt werden. Dabei sind hier nur arithmetische Punktoperationen, Faltungsoperatoren, Dilatationsoperatoren und die Berechnung der Bildpyramide angefallen. Zur Pyramidenberechnung wurden zwar auch nur Faltungsoperatoren eingesetzt, jedoch wurden die Ergebnisse nach der Berechnung unterabgetastet und als Ebenen der Bildpyramide in ein Gesamtbild eingebettet. Dies kann durch eine geschickte Adressierung eines Bildspeichers auch in Echtzeit geschehen. Neben einem Pyramiden–ASIC, das diese Aufgabe wahrzunehmen hätte und noch zu entwickeln ist, fehlte nur noch ein ASIC zur Faltung und Dilatation, das in der Lage ist, beide Operationen auf der eingebetteten Bildpyramide ohne Beeinflussung der Ebenen untereinander (die bei der Einbettung unmittelbar aneinanderstoßen und damit die lokale Bildumgebung ihrer Randbereiche stören) durchzuführen.

Eine Architektur für ein solches ASIC zu entwickeln und in VHDL zu implementieren, war das Ziel dieser Arbeit. Im Unterschied zu sonstiger Bildverarbeitungshardware sollte das ASIC ohne Vollbildspeicher auskommen, die einzelnen eingebetteten Pyramidenebenen im Gesamtbild bezüglich der Operatorumgebungen getrennt behandeln und alle zur Berechnung der klassifizierenden Merkmale benötigten Operatoren nach entsprechender Initialisierung unterstützen. Dies würde dazu führen, daß pro Bildverarbeitungsoperator ein ASIC benötigt wird und somit ein Bildverarbeitungssystem aus vielen gleichartigen Operator–ASICs aufgebaut wäre. Dadurch ließen sich niedrigere Fertigungsstückzahlen und die damit verbundenen hohen Kosten pro ASIC vermeiden (s. [*Lagemann 1987*, S. 274]. Um späteren Änderungswünschen der Verarbeitungsleistung nachkommen zu können, die auch durch fortschreitend höherintegrierende Prozeßtechnologie bedingt sein könnten, wurde das ASIC in allen wesentlichen Parametern parametrisierbar beschrieben.

Ergebnis

Das entwickelte ASIC ist *stand–alone* lauffähig, es ist kein Hostrechner zu seiner Ansteuerung vorgesehen. Vielmehr werden die Verarbeitungsparameter nach einem Reset von jedem ASIC aus einem individuellen ROM gelesen, wonach die Bildverarbeitung sofort startet. Neben wenigen SRAM Bausteinen als externe Zeilenspeicher fällt keine weitere Peripherie an. Da keine Bildspeicher eingesetzt werden, beträgt die Verzögerung durch die Verarbeitung nur 28 Takte, also den Bruchteil einer Bildzeile.

Die erreichbare Filtermatrixgröße ist hier durch die Möglichkeit der Kaskadierung einzelner ASICs unbeschränkt. Für kleinere Faltungsmatrizen lassen sich zudem zwei unabhängige Faltungen pro ASIC realisieren. Die Randwertproblematik wurde hier durch eine Ersatzwerteinblendung für betroffene Umgebungspixel gelöst. Dieses ist sowohl an den physikalischen als auch an den logischen Bildrändern möglich, die innerhalb des Bildes definierbar sind — was die Verarbeitung eingebetteter Bildpyramiden ermöglicht. Es wurde weiterhin eine Treibersoftware entwickelt, die den Inhalt der benötigten Konfigurations–ROMs aus den Angaben der gewünschten Konfiguration selbständig berechnet.

Da die Anforderungen an das ASIC, z. B. die benötigte Rechengenauigkeit und damit sowohl Filterfenstergröße als auch Pixelwert– und Koeffizientenbreite, erst nach umfangreichen praktischen Versuchen feststehen würden, wurde das ASIC bezüglich aller wesentlichen technischen Daten parametrisierbar gehalten. Um das Stückzahlproblem zu entschärfen, wurde die interne Konfiguration des ASICs so variabel gestaltet, daß es für alle obigen Aufgaben genutzt werden kann. Neben dem noch zu entwickelnden ASIC zur Ansteuerung des Bildspeichers zur Pyramideneinbettung, das aufgrund seiner geringeren Komplexität eventuell auch als FPGA realisiert werden könnte, statischen RAMs und ROM–Tabellen ist es der einzige für ein Bildverarbeitungssystem zur Merkmalsberechnung benötigte VLSI–Bausteintyp.

8.2 Diskussion der ASIC–Implementation

Eine Faltung mit 40 Koeffizienten erfordert 40 Multiplikationen und 39 Additionen. Das sind beim Pixeltakt eines Standardvideosignals von 14.75 MHz für Echtzeitverarbeitung über 500 Millionen Multiplikationen und Additionen pro Sekunde. Diese Verarbeitungsleistung läßt sich nicht mit Universalprozessoren erreichen, sondern ist nur durch massive Parallelisierung möglich. Das vorliegende ASIC instantiiert alle Rechenwerke, die zur Faltungsberechnung benötigt werden, parallel. Da der Verarbeitungsfluß nicht stoppt, bietet sich eine Pipeline–Realisierung der internen Verarbeitungsstufen an, um hohe Pixeltaktraten erzielen zu können.

Der laufzeitkritischste Pfad führt durch einen *ripple–carry*–Addierer und einen Multiplexer, die mit dem doppelten Pixeltakt betrieben werden und besitzt eine Verzögerungszeit von etwa 18 ns. Das ASIC würde sich von daher etwa mit 50 MHz externem Takt, also 25 MHz Pixeltakt anstatt der angepeilten 14.75 MHz betreiben lassen — entsprechend schnellere Zeilenspeicher–SRAMs (10 ns) vorausgesetzt. Die reine Standardzellfläche ohne Verdrahtung beträgt 39 mm^2. Als nächster Schritt der Realisierung steht "Placing & Routing" in *Verilog* an.

Eine Signallaufkonfiguration zwischen den ASICs, die die einzelnen Bildverarbeitungsoperatoren realisieren, könnte durch entsprechende Verdrahtung oder steckbare Kabel für die Pixelbusse vorgenommen werden. Dem Nachteil der nur manuellen Eingriffsmöglichkeiten stände der Vorteil der unbegrenzten Erweiterungsmöglichkeiten und minimaler Kosten beim Serieneinsatz gegenüber. Eine Hostrechneranbindung ließe sich durch Austausch der Konfigurations–ROMs der ASICs durch RAM–Bausteine realisieren, die dann vor dem Reset vom Host einzeln zu selektieren und zu initialisieren wären.

8.3 Verbesserungsvorschläge

Wie in 3.1.2 bereits angesprochen, lassen sich optimierte Differentialoperatoren berechnen. Da selbst die Teilmatrix eines ASICs mit zwei Koeffizienten pro Faltungsmatrix nicht ausgelastet ist, sollten entsprechend optimierte Differentialoperatoren mit mehreren Koeffizienten zum Einsatz kommen.

Einzelne Komponenten des ASICs ließen sich auch noch optimieren. Ein mehrstufiger und bei Bedarf pipelineartiger Aufbau der Barrelshifter würde den benötigten Hardwareaufwand reduzieren: Eine mögliche Lösung wäre ein 4–fach Multiplexer, der z. B. 16 benachbarte aus 31 Bits selektiert, um 4 Bit gestuft verschiebt — als mathematischer Operator also Identität, /16, /256, /4096 berechnet — und von 31 auf 19 Bit reduziert. Ein zweiter 4–fach Multiplexer verschiebt nun bitgenau — also Identität, /2, /4, /8 — und reduziert somit von 19 auf 16 Bit. Dieses Vorgehen benötigt $16 + 19 = 35$ 4–fach Multiplexer. Die Verzögerungszeit ist so unkritisch, daß beide Multiplexer in einer Pipelinestufe seriell arbeiten könnten. Alternativ ist ein 4 stufiger Aufbau mit $16 + 17 + 19 + 23 = 85$

2–fach Multiplexern möglich, was in einem Zyklus zeitlich knapp werden könnte und daher wahrscheinlich als Pipeline auszuführen wäre. Die Triviallösung mit sechzehn 16–fach Multiplexern wird im aktuellen ASIC–Entwurf verwendet.

Es existieren weiterhin Makrozellen für Multiplizierer. Da diese ebenso wie einzelne Standardzellen in full–custom–Qualität vorliegen, könnten sie auch ohne Einsparungen (durch Wegfall der Berechnung der niederwertigsten Bits) der generisch rekursiven Komponente, die durch den Standardzellentwurf realisiert wird, überlegen sein.

8.4 Weiterführende Arbeiten

8.4.1 Pyramidenberechnung

Hier soll eine mögliche Architektur des noch benötigten Pyramiden–ASICs besprochen werden. Diese beruht auf einem Bildspeicher. Im Gegensatz zur Bildverarbeitung mit dem entwickelten ASIC, wo kein Bildspeicher benötigt wurde, stellt sich jedoch die Frage, ob nicht auch zur Berechnung von Bild–Pyramiden darauf verzichtet werden sollte. Um dies zu klären, seien hier die Verzögerungen aufgezählt, die sich selbst bei unmittelbarerer Verarbeitung aller Eingabedaten zu einer kompletten Bildpyramide ergäben.

Systeminhärente Verzögerung

Bei einer Pyramidenrealisierung, in der ein Prozessor jedes Pyramidenbild aus der tieferen Pyramidenstufe berechnet, würde eine Kette von Prozessoren vom Vorgänger die Gaußpyramide als Eingangsdatenstrom erhalten und jeweils die nächste Pyramidenstufe errechnen. Bei Verwendung einer $(2*n+1)$ * $(2*n+1)$ Binomialmatrix müssen immer n Zeilen über den Zentralpixel hinaus schon eingelesen sein, bevor das Ergebnis erscheint — selbst an den Bildrändern, wo sie zur Berechnung durch den Randwert ja ersetzt werden. Dieses führt in den oberen Pyramidenstufen zu großen Verzögerungen: In der letzten $(2*n+1)$ * $(2*n+1)$ Pixel Pyramidenstufe bedeuten n Zeilen Verzögerung bereits knapp die halbe Bildübertragungsdauer. In der Stufe davor tritt die halbe Verzögerung auf usw. Diese geometrische Reihe addiert sich zu einer ganzen frame–Dauer systeminhärenter Verzögerung. Wenn bei synchroner Verarbeitung schon ein Zeitversatz von bis zu einem Vollbild auftritt, könnte man auch einen Bildpuffer einsetzen, der eine sehr effiziente Bildpyramidenberechnung erlaubt, ohne eine größere Verzögerung zu verursachen. Es bietet keinen Vorteil, auf einen Bildspeicher zu verzichten. (Der tiefere Grund dieser unerwarteten Verzögerung liegt darin, daß bei synchroner Verarbeitung auf alle Pixel der Operator–Umgebung gewartet wird — unabhängig davon, ob diese überhaupt benutzt oder durch Randwerte ausgeblendet werden.)

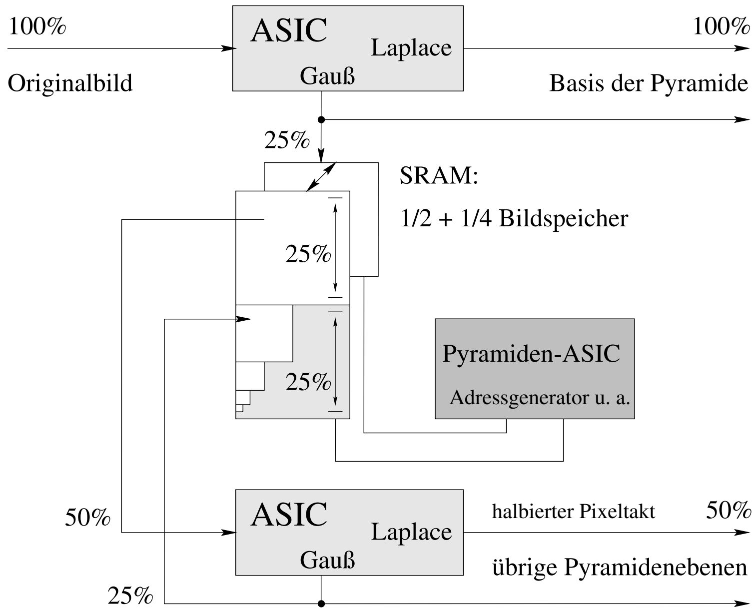


Abbildung 8.1: Datenraten, die bei der Pyramidenberechnung anfallen würden.

Vorschlag einer Architektur

Mit Hilfe eines *double–buffering* Viertelbildspeichers, bestehend aus einem statischen RAM, der simultan vom ersten ASIC gefüllt und vom zweiten ASIC als letzter Bildinhalt gelesen wird (s. Abb. 8.1), kann eine Pyramidenberechnung realisiert werden. Die Ansteuerung der Bildspeicher wird dazu von einem noch zu entwickelnden Pyramiden–ASIC vorgenommen. Der zweite ASIC schreibt sein Ergebnis in einen weiteren einfachen Viertelbildspeicher, der mit seinem Adreßraum direkt an den von ihm sichtbaren Teil des *double–buffering* Halbbildspeichers anschließt und so einen Halbbildspeicher bildet. Mit einer Latenz von einem Vollbild werden die oberen Pyramidenstufen in diesem Halbbildspeicher untereinander akkumuliert und als zweiter Pixelstrom in Echtzeit zur Verfügung gestellt.

Alle oberen Ebenen der Bildpyramiden können so mit Hilfe eines zweiten Satzes an ASICs verarbeitet werden, was gegenüber einzelbildorientierten Systemen eine hohe Effizienz der Hardwareverarbeitung bis in die obersten Bilder von Pyramiden sicherstellt. Die zu einem Bild zusammengefaßten oberen Pyramidenebenen lassen sich anhand von Bildzeilenintervallen problemlos nach der Merkmalsauswertung wieder trennen, da alle Bildzeilen im Gegensatz zum genutzten Bildbereich dieselbe Nettobildbreite besitzen.

Daher wird folgendes Verfahren vorgeschlagen: Der erste ASIC liefert eine Binomialfilterung. Die Unterabtastung wird durch Schreiben der Pixel in einen *double–buffering* Viertelbildspeicher mit jedem zweiten Takt und nur bei geraden Linien vorgenommen. Der korrespondierende zweite Viertelbildspeicher ist mit einem weiteren Viertelbildspeicher, der adreßmäßig darüber liegt, zu einem Halbbildspeicher zusammengefaßt. Nach einem Umschalten des beschriebenen *double–buffering* Bildspeichers liegt insgesamt nun ein Halbbildspeicher vor, der in der unteren umgeschalteten Hälfte mit der ersten Pyramidenstufe gefüllt ist. Ein zweiter ASIC liefert ebenfalls Binomialfilterungen, aber mit halbem Takt: Er liest in der ersten Hälfte von Adresse 0 das erste Pyramidenhalbbild und schreibt in der zweiten Hälfte seines halben Taktes das Ergebnis in die erste freie Speicherzelle des Halbbildspeichers. Die Adresse des Halbbildspeichers wird für das Lesen mit jedem Takt inkrementiert, für das Schreiben nur bei geraden Zeilen und nur mit jedem zweiten Takt.

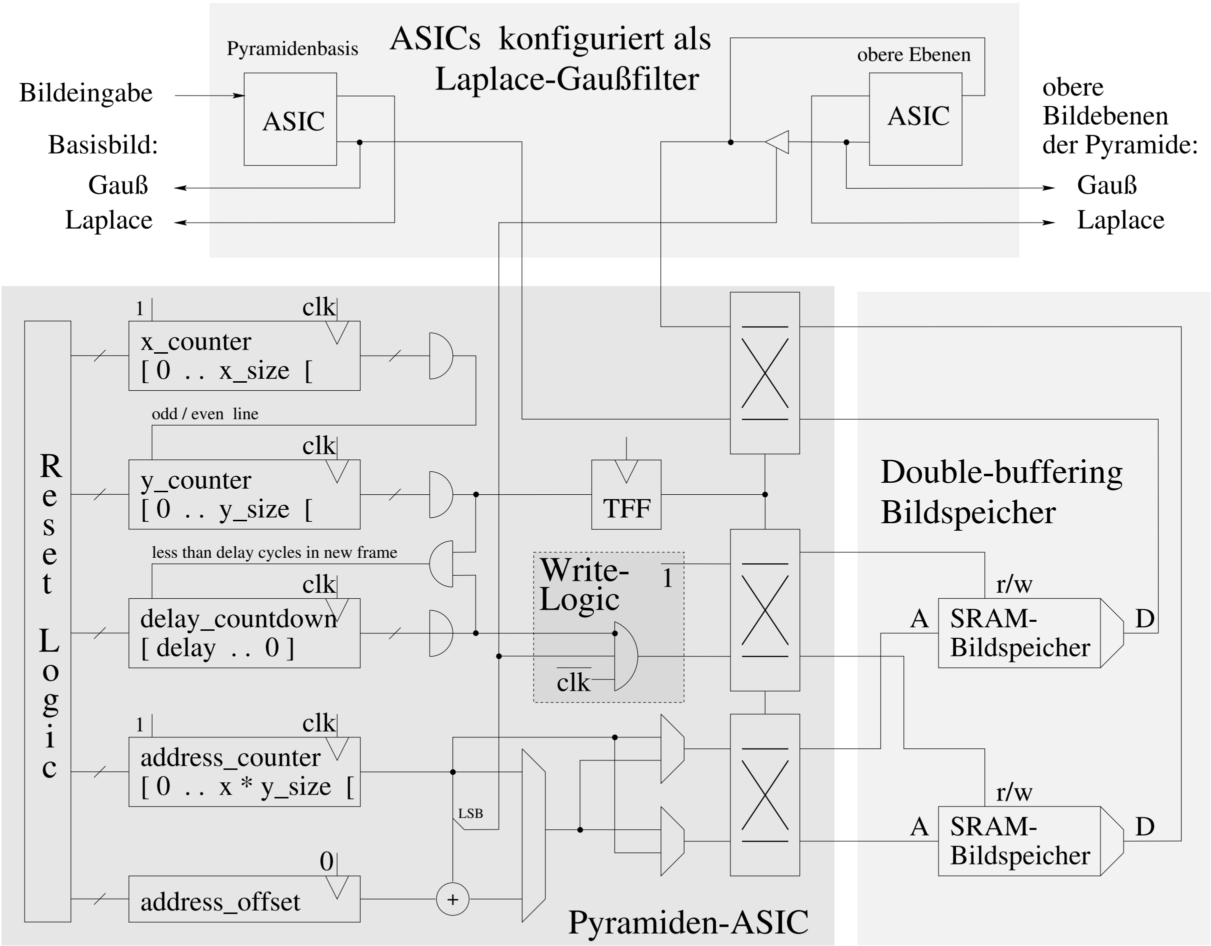


Abbildung 8.2: Pyramidenberechnung mit Pyramiden–ASIC

Es erscheinen damit in den später beschriebenen Speicherbereichen immer doppelt soviele Bilder von jeweils einem Viertel der Fläche wie vorher; die oberen Ebenen der Pyramide kommen entsprechend mehrfach nebeneinander zu liegen. In ihrer Anzahl äußert sich der Umstand, daß pro Unterabtastung und damit pro Pyramidenstufe ein Freiheitsgrad von vier bezüglich der Abtastung bestand. Da jeweils zwei Ergebnisse nacheinander unter derselben Adresse gespeichert wurden, ergibt sich jedoch jedesmal nur eine Verdoppelung der Anzahl von Bildern. Entscheidet man sich durch eine Einteilung in logische Bildbereiche z. B. für die Reihe linksbündig oder rechtsbündig übereinander liegender Bilder, so liegen in ihnen die Ebenen der Bildpyramide vor. (In allen Beispielen wurde der Einfachheit halber immer die linksbündige Reihe von Bilder als Pyramide dargestellt).

Die SRAM–Ansteuerungen müssen für das Schreiben und Lesen unabhängig gestartet werden können, um die Pipeline– und Zeilenspeicherzeitverluste zu kompensieren. Die Ansteuerlogik des *double–buffering* Speichers läßt sich diskret, als FPGA oder als weiteres Pyramiden–ASIC, realisieren. Einen Vorschlag der Komponenten auf Hauptblockebene für das Pyramiden–ASIC zeigt Abb. 8.2. Der grau unterlegte große Block links unten entspricht dem zu entwickelnden Pyramiden–ASIC. Rechts daneben sind die beiden als *double–buffering*–Bildspeicher eingesetzten SRAMs dargestellt. (Zur Vereinfachung, wurden hier jedoch beide als gleichgroße Halbbildspeicher dargestellt. Prinzipiell ließen sich auch drei Viertelbildspeicher einsetzen. Zwei davon nähmen die Rolle der dargestellten Halbbildspeicher ein, der dritte wäre adreßmäßig immer unter demjenigen einzublenden, auf dem die Berechnung der oberen Ebenen der Bildpyramide abläuft.).

Oben sind zwei ASICs des hier entworfenen Typs dargestellt, die die Rechenoperationen zur Berechnung der Pyramidendaten durchführen. Die einzelnen Komponenten lassen sich in Abb. 5.3 wiederfinden. Hier sind die Halbbildspeicher links angeordnet ("SRAM"), und die beiden ASICs zur Datenberechnung tragen die Bezeichnung "Gaußfilter & Laplacefilter". Der zweite Pixelausgang ist durch die Differenzbildung des Ergebnisses des ersten Ausgangs mit dem Zentralpixel so konfiguriert, daß der Pixelstrom der eingebetteten Laplacebildpyramide am zweiten Ausgang zur Verfügung steht. Das Pyramiden–ASIC selbst ist dort als *black–box*–Komponente dargestellt.

Die anschließende Bildverarbeitung erfolgt einmal auf dem Basisbild der Bildpyramiden, zum anderen auf dem Bild, in das deren obere Ebenen eingebettet sind. Eine Einbettung der kompletten Bildpyramide in ein Bild würde zu einer Heraufsetzung des Pixeltaktes um 50% geführt haben. Da dieses zu kritisch schien, wurde es nicht erwogen. Sobald die Bildgröße kleiner als die Faltungsmatrixgröße wird, versagt die angewandte Methode zur Randwerteinblendung und führt zu falschen Ergebnissen. Die Berechnung der allerobersten Pyramidenebenen muß also auf dem Hostrechner erfolgen, sofern sie überhaupt benötigt wird.

8.4.2 Benötigte Klassifikator–Komponenten

Abb. D.2 im Anhang zeigt den aktuellen Stand des Projektes. Weiterführende Arbeiten bestehen im Entwurf des Pyramiden–ASICs, dessen Hauptblockebenen in Abb. 8.2 skizziert ist. Zudem muß eventuell die oberste Pyramidenebene, die noch vom ASIC berechnet werden kann (die Randausblendung funktioniert ohne Einschränkung nur bei Bildern bis zur Matrixgröße), dem Rechner zur Weiterverarbeitung (Berechnungen bis zur obersten 1 Pixel auflösenden Pyramidenebene) angeboten werden. Mit einem linearen Klassifikator–ASIC oder einem LUT–ASIC, das mittels eines 16–MBit DRAM–Speicher–ICs (oder zweier Bausteine gleichen Inhaltes, um abwechselnd Daten auszugeben und einen Refreshzyklus durchzuführen) für drei 8–Bit Merkmale realisiert werden könnte, wären dann alle Komponenten verfügbar. Anhand einer größeren Anzahl von unter realen Umgebungsbedingungen eingescannten Bildern, die bereits bewertet wurden und für deren Fehlklassifikation jeweils die Kosten angegeben sind, muß nun eine Clusteranalyse erfolgen. In einem ersten Schritt gilt es dabei, die Form der Cluster im Merkmalsraum und die möglichen Änderungen der Cluster, die durch veränderte Umweltbedingungen auftreten und durch eine adaptive Steuerung kompensiert werden sollen, zu erforschen. Anschließend kann über eine entsprechende Modellierung bzw. Auswahl mathematischer Ansätze ein quantitativ optimaler Parametersatz für den Klassifikator berechnet werden, der in einigen Parametern adaptiv modifizierbar ist. Der nächste Schritt besteht darin, den Teil der höheren Bildverarbeitung, der auf einem Rechner als Programm abläuft, zu entwickeln und echtzeitfähig zu implementieren. Schließlich kann das hardwaregestützte System implementiert und zusammen mit der Software im realen Einsatz getestet werden.

8.4.3 Komplexere Bildverarbeitung

Mit der Detektion verschiedener Objektklassen oder der ermittelten Verteilung klassifizierender Merkmale des Originalbildes ist das Problem, ein Holz zu klassifizieren, noch nicht gelöst. Objekte befinden sich nicht nur an diskreten Pixelpositionen, sondern auch zwischen ihnen und haben ebensowenig nur Abmessungen, die zur Abbildung auf genau einer Ebene der Laplacepyramide führen. Deswegen müssen Operatoren jeder Bildverarbeitung eine gewisse Unschärfe besitzen, damit keine Objekte undetektiert bleiben. Als Folge werden Objekte an benachbarten Pixelpositionen und auf darüber oder darunter liegenden Ebenen der Bildpyramide mehrfach detektiert. Sie dürfen aber nicht mehrfach in die Bewertung eingehen. Verstärkt wird diese systeminhärente Unschärfe dadurch, daß sich der Rechenaufwand in Grenzen halten muß, so daß Filter mit sehr steiler Transferfunktion nicht realisierbar sind. Außerdem begrenzt zudem die Anwendungsdomäne selbst die Detektionsschärfe der einfacheren Bildverarbeitung: Holz als Naturprodukt läßt wegen der stärkeren Abweichungen der zu detektierenden Objekte nur unscharfe Objektdetektionsmethoden zu, die auch bei Abweichungen vom mittleren Erscheinungsbild eines Objektes noch funktionieren müssen. Die Mehrfachdetektion der Objekte muß als erster Schritt der komplexeren Bildverarbeitung sinnvoll gelöst werden. Neben der Erstellung einer Objektliste könnten klassifizierende Merkmale bei Bedarf auch noch der statistischen Auswertung dienen. Üblicherweise liegen sie als Histogramme der verschiedenen Frequenzbereiche der Pyramide vor und müssen je nach gewünschter Aussage interpretiert werden.

Anhang A

Batchjobs des Bildverarbeitungsprogramms

Bevor die Berechnung von Merkmalen beginnen konnte, mußten zuerst die eingescannten Bilddaten aufbereitet werden.

Bildaufbereitung

Die eingescannten Bilder lagen als Binärdateien vor. Ihr Name bestand aus einem Buchstaben für jedes Holz und einer Nummer. Es waren sechs Bilder pro Holz zuzuordnen. Die Ränder des Holzes waren oben und unten durch einen weißen Hintergrund abgehoben, der im Randbereich die Kamera übersteuerte (100%). Im Holzbereich des Bildes kamen nur Helligkeitswerte kleiner 80% vor. Die folgenden Batchjobs dienten dazu, Bildränder durch die mittlere Helligkeit vom Holz aller Bilder zu ersetzen und den Kontrast zur vollen Nutzung der Quantisierung anzuheben.

Die Befehle in den Batchjobs werden seriell verarbeitet. In der linken Spalte steht der Batchname, unter dem der Batchjob aufgerufen werden kann, die rechten Zeilen enthalten die einzelnen Befehle. Eine Befehlsübersicht findet sich im Anhang B:

randlos.job:	G/512,512 N/J R/randlos2,n
randlos1.job:	l/$#,b,u Y/a,0.8 V/g Y/a,0.52 P
	l/$#,b,u D/1 Y/a,1.3 P s/$#,p I/:
	R/randlos1,n
randlos2.job:	I/5 R/randlos1,n N/@ R/randlos2,n

Da die Bilder nur als Binärstrom vorlagen, mußte vor dem Einlesen zuerst die Bildgröße durch Skalieren des Quellbildes (seine Größe wird beim Einlesen für das neue Bild übernommen) auf die richtige Größe (512 * 512 Pixel) durch *G/512,512* festgelegt werden. Dies geschieht nur zu Beginn, da alle Bilder im weiteren 512 * 512 Pixel groß sind. Es lagen Bilder von zehn Hölzern, als Dateien *A0 – A5* bis *J0 – J5* vor. Mittels *N/J* wird die ASCII–Variable *$* auf J gesetzt; sie wird später zyklisch bis A dekrementiert, so daß alle Bilder der Reihe nach bearbeitet werden. Mit *R/randlos2,n* wird der Batchjob *randlos2.job* eingeladen. Er setzt als erstes zur Initialisierung des Batchjob *randlos1.job* die Integervariable # auf 5 und ruft diese dann als Unterroutine auf, die alle Bilder eines Holzes verarbeitet (s. u.). Danach wird die ASCII–Variable dekrementiert. Ist noch nicht das letzte Bild verarbeitet worden, war die Variable also größer A, wird *randlos2.job* erneut aufgerufen und bearbeitet das nächste Holz. Andernfalls kann die ASCII–Variable nicht weiter dekrementiert werden. Der Interpreter überspringt in diesem Fall immer den nächsten Batchjobaufruf, so daß hier die Verarbeitung endet.

Zur eigentlichen Bildverarbeitung wird zuerst das digitalisierte Bild als unsigned–Bild eingeladen (*l/$#,b,u*) und als Skalarwert der Absolutwert auf 0.8 gesetzt (*Y/a,0.8*). Ein Vergleich des Bildes mit dem Wert 0.8 über die Größer–Relation (*V/g*) führt zu einem Binärbild, das den Wert 0.00 für Holzbereiche und 1.00 (aufgerundet) für Randbereiche liefert. Eine Multiplikation (P) mit dem zuvor gesetzten Wert 0.52 (*Y/a,0.52*) liefert ein Bild, das für die Randbereiche genau den Wert 0.52 annimmt. Dieser Wert muß von den weißen Randwerten abgezogen werden, damit sie den Mittelwert der Helligkeit aller Hölzer 0.48 annehmen. Im Bereich des Holzes hat das Bild wieder den Wert null. Nun wird das Originalbild erneut eingeladen (*l/$#,b,u*) und von ihm das modifizierte Binärbild des Randes subtrahiert (*D/1*). Nach einer Anhebung (P) des Kontrastes um 30% (*Y/a,1.3*) wird das alte Bild durch das modifizierte Bild, inklusive seiner Bildparameter, ersetzt: *s/$#,p*. Um die weiteren Bilder desselben Holzes zu verarbeiten, wird die Integervariable, die Teil des Dateinamen *$#* ist, dekrementiert (*I/:*). Soweit sie vorher nicht schon null war, wird derselbe Batchjob erneut gestartet. Ist dies nicht der Fall, terminiert das Unterprogramm, und es wird das nächste Holz selektiert (s. o.).

Die Bilder eines Holzes waren nach dem Aufruf dieses Batches jedoch noch auf verschiedene Dateien verteilt. Als nächster Schritt wurden die Bilder aneinandergefügt und in einem entsprechend großen Gesamtbild eines Holzes abgespeichert. Da die Bildaufnahme auf einer ebenen Fläche mit vermessenen Anhaltemarken für die Verschiebung des Holzes durchgeführt wurde, ergaben sich bei der Verknüpfung der Einzelbilder im wesentlichen systematische Fehler, die sich kompensieren ließen: Die Ebene, auf der das Holz an der Kamera vorbeigeschoben wurde, lag um wenige Grad schief zum Kamerabild, so daß die Einzelbilder in der Vertikalen um wenige Pixel verschoben werden mußten. Durch die Schräglage steigerte sich die Bildhöhe des Gesamtbildes um 7%. Da zu Beginn der Arbeit nur 4 MB Speicher auf dem PC zur Verfügung standen, wurden nur jeweils 4 Bilder zu einem mit 2048 * 527 Pixel zusammengefaßt (zwei 3072 * 527 Pixel große Bildspeicher sind mit 4 MB Speicher maximal möglich, wobei Pyramidenbilder 50% breiter sind als ihre Ursprungsbilder).

```
mkquart   job:   N/K   R/mkquart1,n
mkquart1  job:   l/$3,p   Y/a,0.621   O/0,0,-27,-12   Y/a,0
                 O/-1536,0,0,0   s/tmp,b   I/:
                 l/$2,p   Y/a,0.621   O/0,0,-14,1   Y/a,0
                 O/-1024,512,0,0   l/tmp,b,u   S/1   s/tmp,b
                 l/$1,p   Y/a,0.621   O/0,0, -1,14   Y/a,0
                 O/-512,1024,0,0   l/tmp,b,u   S/1   s/tmp,b
                 l/$0,p   Y/a,0.621   O/0,0, 12,27   Y/a,0
                 O/ 0,1536,0,0   l/tmp,b,u   S/1   s/$_quart,p
```

Der Aufruf der Batchroutine *mk_quart1.job* mittels *mk_quart.job* erfolgt genau nach demselben Muster, wie in *randlos.job* die Batchroutine *randlos1.job* aufgerufen wird. In ihr werden zuerst die Teilbilder im eigenen Format geladen (*l/$#,p*) und die Konstante auf den Mittelwert der Bilder 0.621 gesetzt. Danach werden die Teilbilder in der Vertikalen oben und unten mit individuellem Rand versehen, um die geringe Schieflage bei der Aufnahme auszugleichen. Anschließend wird jedes Teilbild durch Anfügen von schwarzen Rändern zum Vollbildformat expandiert; um ein gesamtes Vollbild zu erhalten, werden alle aufsummiert. Damit das Programm die Bilder unabhängig von ihrem ehemaligen Ursprung unverschoben addiert, werden sie im parameterlosen Format gespeichert. Für jedes Bild wird die bisherige Überlagerung geladen (*l/tmp,b,u*), das neue Bild hinzuaddiert (*S/1*) und wieder abgespeichert. Das erste Bild wird direkt, die letzte Summe parametrisiert und unter dem Namen des Ergebnisbildes abgespeichert.

Angewandte Merkmalsberechnung

Auf der Basis dieser Bilder erfolgte nun die Berechnung der Merkmale. Die Orientierungsvektorberechnung erfolgte für alle Bilder (*orientrg.job*, vgl. o.). Dazu wird im Batchjob *orientr1.job* das Holzbild geladen (*l/$_quart,p*) und einer initialen Filterung mit einem kleinen Gaußfilter (3 * 2 Matrix) unterzogen. Dies geschieht separiert mit eindimensionalen Binomialmatrizen in X–Richtung (*b/x,2,0*) und in Y–Richtung (*b/y,1,0*). Anschließend wird die Laplacepyramide mit einem Gaußfilter achter Ordnung mit 9 * 9 Koeffizienten berechnet, wobei von Ebene zu Ebene der Kontrast um 4/3 angehoben wird (*w/8,2,1.33*), damit die (tieffrequenten) Strukturen der oberen Ebenen nicht im Quantisierungsrauschen untergehen. Das Ergebnis, ein vorzeichenbehaftetes Bild, liegt nach dem Mitteln (*S/0*) mit der Konstante 1 (*Y/a,1*) als vorzeichenloses Grauwertbild vor. Da es für sich bereits ein Merkmal bildet und außerdem noch ein weiteres Mal benötigt wird, wird es gespeichert (*s/$_laplc,p*).

Zur Orientierungsvektorberechnung werden die Differentiale in X– und Y–Richtung benötigt. Mit (*d/x,1,1*) wird das symmetrische Differential gebildet und mit (*s/dif_x,p*) abgespeichert (es wird wegen seiner geringen Aussteuerung dabei noch implizit verdoppelt, was zu der Matrix $(1, -1)$ anstatt $(0.5, -0.5)$) führt. Dasselbe folgt für das Differential in Y–Richtung. Um die Produkte der Differentiale zu bilden, wird zuerst das Y–Differential, das noch im zweiten Bildspeicher steht, kopiert (es steht danach in beiden) und durch Multiplikation (*P*) quadriert abgespeichert (*s/dy_dy,p*). Durch Bildspeichertausch (*x*) wird nun wieder das Y–Differential in den zweiten Bildspeicher geholt und danach das X–Differential eingeladen. Nach der Multiplikation (*P*) wird nun das berechnete gemischte Produkt der Differentiale gespeichert (*s/dx_dy,p*). Nach einem weiteren Bildspeichertausch und Kopieren desselben wird mit dem dritten Produkt (*P*) das Quadrat des X–Differentials berechnet und gespeichert. Der zweite Teil der Verarbeitung ist in einem weiteren Batchjob ausgelagert, der zum Schluß aufgerufen wird (*R/orientrg2,n*).

```
orientrg.job:   N/K   R/orientr1,n
orientrg1.job:  l/$_quart,p   b/x,2,0   b/y,1,0
                w/8,2,1.33   Y/a,1.0   S/0   s/$_laplc,p
                d/x,1,1   s/dif_x,p   l/$_laplc,p
                d/y,1,1   s/dif_y,p   c
                P/3   s/dy_dy,p   x   l/dif_x,p
                P/3   s/dx_dy,p   x   c
                P/3   s/dx_dx,p   R/orientrg2,n
orientrg2.job:  l/dx_dx,p   l/dy_dy,p   S/0
                b/x,8,2   b/y,8,0   G/f,0.5,0.5   s/$_y_pl_x,p
                l/dx_dx,p   l/dy_dy,p   D/0
                b/x,8,2   b/y,8,1   G/f,0.5,0.5   s/$_y_mi_x,p
                l/dx_dy,p
                b/x,8,3   b/y,8,1   G/f,0.5,0.5   s/$_2_xy,p
                N/@   R/orientr1,n
```

In diesem werden die einzelnen Komponenten des Orientierungsvektors und das Orientierungsintensitätsmaß gebildet. Dies letzte Merkmal wird zuerst durch eine separierte 9 * 9 Pixel Gaußfilterung (*b/x,8,2 b/y,8,0*) über die Summe (*S/0*) der beiden Quadrate des X– und Y–Differentials (*l/dx_dx,p l/dy_dy,p*) gebildet. Die erste Komponente des Vektors wird genauso berechnet, nur mit dem Unterschied, daß statt der Summe die Differenz gebildet wird. Die zweite Komponente des Vektors besteht aus der Gaußfilterung über dem Quadrat des gemischten Differentialproduktes (*l/dx_dy,p*). Der zusätzliche Faktor zwei (vgl. 3.12) ist hier im Shiftfaktor des ersten Binomialfilters enthalten (*b/x,8,3* anstatt *b/x,8,2*). Da alle Orientierungsmaße zum Schluß tiefpaßgefiltert werden, enthalten sie keine hohen Frequenzen mehr. Vor dem Abspeichern konnten sie somit ohne Informationsverlust um den Faktor zwei in beiden Richtungen unterabgetastet (*G/f,0.5,0.5*) werden (um Speicherplatz einzusparen).

```
dilatat.job:   N/K   R/orientr1,n
dilatat1.job:  l/$_quart,p   b/x,2,0   b/y,1,0   g/8 s/$_gauss,p
               r/9,9,1   t   t/a,1   D/1   s/$_dilat,p
               N/@   R/dilatat1,n
```

Im Batchjob *dilatat1.job* wird anstatt der Laplacepyramide, die in *orientr1.job* berechnet wurde, nun die Gaußpyramide nach der initialen Eingangsfilterung (*b/x,2,0   b/y,1,0*) erzeugt. Die beiden Befehle dazu unterscheiden sich darin, daß der Befehl zur Gaußpyramidenberechnung nur den ersten Parameter (Filtergröße) kennt (*g/8*). Die Werte des Ergebnisbildes sind vorzeichenlos und werden direkt als ein klassifizierendes Merkmal abgespeichert. Zur Berechnung des letzten Merkmals wird die Rangordnung auf einer 9 * 9 Umgebung berechnet, das erst–größte Element (*r/9,9,1*) im Ergebnisbild abgelegt und der Randwert auf den Maximalwert gesetzt (erstes *t*: Randwert abschalten, zweites *t/a,1*: auf absoluten Wert 1 setzen). Im zweiten Bildspeicher steht noch das Originalpyramidenbild, das mit dem folgenden Befehl *D/1* nun von dem Dilatationsbild abgezogen wird. Das Ergebnis wird als letztes Merkmal abgespeichert (*s/$_dilat,p*) und der Vorgang für den Rest der Holzbilder wiederholt (*N/@   R/dilatat1,n*).

Anhang B

Dokumentation zur Software

B.1 Anleitung zum Bildverarbeitungsprogramm

B.1.1 Hardwareanforderungen

Das Programm ist für PCs mit dem Betriebssystem MS–DOS geschrieben. Bis zu 5 MByte Arbeitsspeicher wird je nach gewünschter maximaler Bildgröße benötigt. Alle vorgesehenen Videomodi — bis auf einen — sind mit einer 1 MByte Grafikkarte nach dem VESA–Standard nutzbar; mit einer Standard–VGA–Karte kann man nur zwei Modi benutzen (näheres: siehe letzter Befehl der nachfolgenden Befehls–Liste).

EMS–Speicher

Die zu Programmbeginn allokierte Bildspeichergröße ist beim Programmstart anzugeben und begrenzt die nachher im Programm mögliche dynamische Bildgröße. Ab etwa 640 * 350 Pixel Bildgröße wird EMS–Erweiterungsspeicher benötigt. Das Programm benutzt pro 16 KB ein EMS–Handle, also eventuell mehr, als emm386.exe defaultmäßig bereitstellt. Die Handleanzahl–Option beim Installieren des EMS–Speicher Memory–Managers sollte daher auf ihren Maximalwert 255 hochgesetzt und es sollte ein 255 * 16 = 4080 KByte EMS–Speicher eingerichtet werden.

B.1.2 Besonderheiten

Wertinterpretation

Das zur Implementation verwendete Divisionsergebnis von Integerzahlen ist um den Nullpunkt herum symmetrisch, so daß bei Divisionen eine Häufung im Histogramm um den Nullpunkt auftritt. Bei den Filtern und bei Wechseln in die *signed*–Darstellung wird dieser Effekt jedoch unterdrückt, so daß den diskreten Werten Intervalle gleicher Breite der reellen Zahlen entsprechen.

Quantisierungsverluste

Manche Operationen erlauben optional ein arithmetisches Linksshiften des Ergebnisses um bis zu 8 Bit. Die maximal mögliche Verschiebung ist so gewählt, daß nicht unnötigerweise signifikante Stellen verlorengehen, darüber hinaus gehendes Shiften ist durch eine nachfolgende Produktbildung mit einem Skalarwert durchzuführen: Linksshiften nach Binomialfilterungen ist eigentlich nur für vorzeichenbehaftete Bilder sinnvoll, bei vorzeichenlosen führt dieses meistens — für ein Großteil der Pixel — zu übergelaufenen (geclippten) Werten. Weil Differentialfilter in der Regel zu vorzeichenbehafteten Bildern führen, ist besonders bei solchen mit integriertem Binomialfilter (Sobeloperator) Shiften sehr sinnvoll. Sonst würden mehrere Nachkommastellen ungenutzt dadurch verloren gingen, weil der Wertebereich normalerweise kaum ausgeschöpft wird. Für die Laplacepyramide gilt dasselbe.

Summen– und Differenzbildungen können zum Wechsel zwischen *signed*– und *unsigned*–Format benutzt werden. Speziell in dieser Richtung würde nach der Addition von eins fast zwangsläufig ein Überlauf stattfinden. Ein Halbieren der Werte des *signed*–Bildes und des Summanden vor der Addition würde zu einem Quantisierungsverlust von einem Bit führen. Ein arithmetisches Rechtsshiften um ein Bit vor dem Abspeichern ist in dieser Konvertierungsrichtung wünschenswert. Zur Wandlung von *unsigned*– ins das *signed*–Format erfolgt dagegen eine Subtraktion von 0.5 mit anschließendem Verdoppeln der Werte, entsprechend einem Linksshiften vor dem Abspeichern. Dies ist möglichst in einem Arbeitsgang auszuführen.

Um beide Forderungen zu erfüllen, ohne neue Parametertypen einführen zu müssen, werden die Ergebnisse von Summen– oder Differenzoperatoren um den Wert eines Parameters nach links– und danach generell um ein Bit nach rechts geshifted. Durch den Parameter eins ergeben sich also die Funktionen im mathematischen Sinne.

B.1.3 Befehle

Alle Befehle werden durch Druck der entsprechenden Buchstabentaste ausgeführt. Soweit noch Optionen zur Verfügung stehen oder Parameter einzugeben sind, werden sie anschließend durch weitere Buchstaben oder Zahleingaben interaktiv getroffen. Einzelne Buchstaben stehen dabei als Kürzel für Parameter, die meistens durch einzelne Ziffern nacheinander eingegeben werden. Mit dem Oder–Symbol "—" verknüpfte Buchstaben stellen verschiedene Optionen aus. In der Batchliste werden alle Eingaben durch Formatierzeichen getrennt dargestellt. Verlassen kann man das Programm durch Eingabe von Q im Hauptmenü. Die Befehle sind im folgenden ebenso wie im Menü (s. Tafel D) in thematische Bereiche gegliedert.

Ein–/ Ausgabebefehle

Bild laden/speichern `l / s: p|b [s|u]`

Zum binären Laden von Bildern (b) müssen deren Größe und das abgespeicherte Format (ob die Bilder im signed– (s) oder unsigned–Format (u) vorliegen) angegeben werden. Beim binären Speichern wird automatisch das im Menü hinter "destin: " mit dem Kürzel s oder u angegebene Format verwendet. Als Bildgröße der zu ladenden Datei wird beim binären Laden die des zuletzt errechneten Bildes angenommen. Zur Vorgabe genügt es, irgendein Bild auf die erwartete Bildgröße zu skalieren. Die Extremwertangaben des Bildes beziehen sich nach dem binären Laden nicht wie sonst auf die letzte Operation, sondern nur auf die tatsächlich gelesenen Pixel. Damit kann beim Einladen des Bildes z. B. kein Überlauf der vorherigen Operation angezeigt werden. Das Laden von Bildern, die im parametrisierten eigenen Bildformat gespeichert wurden (p), benötigt keine weiteren Angaben, da beim Speichern zuvor der komplette Zustand des Bildes gesichert wurde.

Bild kopieren `c`

Dieser Befehl kopiert den Destination–Bildspeicher. Er ist als Operation aufzufassen. Deswegen enthält der neue Destination–Bildspeicher keine Über– oder Unterläufe mehr; beim Kopieren können solche nicht auftreten.

Bildpuffer tauschen `x`

Tauscht Destination– und Source–Bild und alle assoziierten Parameter gegeneinander aus. Ein zweites Tauschen stellt den Ursprungszustand wieder her.

Bild linear anzeigen `v: cursor|CR`

Dieser Befehl stellt die Bilder im gewählten Grafikmodus dar. Bilder im unsigned–Format werden als Graustufenbild ausgegeben, Schwarz entspricht dem Pixelwert 0.00 und Weiß dem Wert 1.00. Bei Bildern im signed–Format werden positive Werte statt in Grau in Grün dargestellt. Negative Werte werden in Rot ausgegeben. Die maximale Farbintensität entspricht dabei den Extremwerten 1.00 und -1.00. Pixel mit betragsmäßig kleinen Werten werden entsprechend dunkel bis schwarz dargestellt. Zum Ergebnisbild der letzten Operation läßt sich optional das letzte Quellbild anzeigen. Es wird dabei im Menü immer die mögliche Alternative zum aktuell gesetzten Anzeigemodus genannt. Für beide Bilder steht dann die linke bzw. rechte Bildschirmhälfte zur Verfügung. Wird ein signed– und ein unsigned–Bild nebeneinander angezeigt, halbiert sich die zur Verfügung stehende Farbpalette. Mit der Leertaste wechselt man zum anderen Bild. Ist das aktuelle Bild zu groß, kann der sichtbare Ausschnitt mit den Cursortasten verschoben werden. Mit Escape gelangt man zum Hauptmenü zurück.

Hat man im Menü zuvor eine Merkmalsdatei und eine Anzahl von Merkmalen angegeben, kann aus dem Anzeige– (mit Return) in einen Markierungsmodus gewechselt werden. Dort läßt sich ein Cursorkreuz mit den Cursortasten auf eine Bildposition verschieben, wobei mit den Zahltasten die Pixel pro Cursorschritt (in logarithmischer Skala) gewählt werden können. Danach lassen sich jedem Merkmal mit einem Schieber Werte zuweisen. Alle Einzelschritte werden mit Return bestätigt; die ganze Eingabeprozedur kann zudem jederzeit mit Escape abgebrochen werden. Die Pixelpositionen und Merkmalswerte werden jeweils als Zeile an die selektierte Datei angehängt. Dieser Modus dient dazu, Merkmale vom Benutzer, z. B. vorgegebene Klassifikationen, mit einem externen Programm den berechneten Merkmalen gegenüberstellen zu können. Die Merkmalsvorgabe des Benutzers kann als Sollklassifizierung im Merkmalsraum als Colorierung der Merkmalsvektoren wiedergegeben werden, was eine grafische Diskriminanz– und Clusteranalyse ermöglicht.

Bild eingeebnet anzeigen `e: cursor|CR`

Dieser Modus ist — abgesehen von der Programmierung der LUT — der Grafikkarte zur Farbdarstellung absolut identisch zum vorherigen. Die Darstellung des Bildes erfolgt statt mit linearer Grau– bzw. Rot–/Grünstufen Look–Up–Table so, daß alle darstellbaren Farbwerte gleichhäufig benutzt werden. In einem Histogramm, denke man sich unter der Werteskala, einen linearen Grauwertkeil, bestehend aus allen möglichen Farbstufen, gezeichnet. Werden die Histogrammlinien nun so verschoben, daß die Summe aller Pixel über je einer Farbstufe der unteren Skala gleich ist, erhält man bezüglich der Farbstufen (nicht der Pixelwerte!) ein eingeebnetes Histogramm. Das Verfahren ist als Histogrammausgleich bekannt und wird u. a. zur Darstellung von Tomographiedaten verwendet. In [*c't, 9.91 1991*] wird hergeleitet, daß der Histogrammausgleich einer LUT–Darstellung mit der Verteilungsfunktion der Grauwerte als LUT–Inhalt entspricht (die zugehörige Dichtefunktion der Grauwerte ist das Histogramm selbst). Bei signed–Formaten erfolgt der Histogrammausgleich für den positiven und negativen Wertebereich, also die Rot– und Gründarstellung, getrennt. Ist im Extremfall nur eine Histogrammlinie vorhanden, ist das Ergebnis eines Histogrammausgleichs nicht eindeutig, sondern implementationsabhängig. Die Bilddaten selbst bleiben auf jeden Fall unbeeinflußt. Eine entsprechende Umrechnung würde als nicht–lokale Operation nicht ins Konzept aller anderen lokalen Funktionen passen.

Histogramm/ Statistik `h`

Bezüglich der jeweils letzten Bildoperation (außer binärem Laden) stehen der Maximal– und Minimalwert sowie die Anzahl der Pixel mit Über– bzw. Unterlauf (die geclippt werden mußten) sowie der Mittelwert und die Standardabweichung der Bilddaten zur Verfügung. Diese lassen sich zusammen mit dem Histogramm in linearer und logarithmischer Darstellung anzeigen. Ausgenommen sind dabei Bilder mit null Pixeln Größe.

Stapelverarbeitung

Batchjob speichern `J`

Alle Befehle, die in der unteren Batch–Zeile momentan angezeigt werden, lassen sich als ASCII–String abspeichern. Obwohl dabei keine Sonderzeichen benutzt werden, ist der String nicht ohne weiteres ohne das Bildverarbeitungsprogramms editierbar, da kein CR oder LF Sonderzeichen vorkommen darf, sondern nach jedem, auch dem letzten Befehl ein Leerzeichen folgen muß. Das Programm nimmt nach dem Einladen eines Batchjobs keine Plausibilitätsprüfungen mehr vor, sondern unterstützt stattdessen selbst das Editieren von Batchjobs.

Batchjob laden `R: a|n`

Dieser Befehl lädt den zuvor gespeicherten Batch–Job wieder ein. Dabei werden die dem Ladebefehl noch folgenden Batch–Befehle verworfen, wenn die Option Neuladen (n) gewählt wurde. Ansonsten wird der Ladebefehl selbst durch den geladenen Batchjob ersetzt (a). Soll der Ladebefehl uninterpretiert als Batchjobbefehl aufgenommen werden, so ist, wie auch bei allen anderen Bildbearbeitungsbefehlen, der Protokollmode zu wählen. Batch–Jobs können sich als Unterprogramme aufrufen. Dabei müssen Zyklen unbedingt vermieden werden, weil man aus ihnen heraus keine Kontrolle mehr über das Programm zurückgewinnen kann. Da bei einem Programmabbruch mittels CTRL–BREAK der EMS–Speicher nicht wieder freigegeben werden kann, ist ein Rechnerreset unausweichlich.

Einträge löschen/ zyklisch tauschen `F / U / Z`

Der erste (F) und letzte (U) Befehl des Batch–Strings, der permanent unten im Hauptmenü angezeigt wird, läßt sich direkt löschen. Um mittlere Befehle zu modifizieren, ist zyklisch zu rotieren (Z) bis der Befehl am Ende steht, dann zu ändern und bis zur Ausgangsstellung weiter zu rotieren.

Batchjob ausführen `A`

Der Befehl A schaltet in den Executemode und startet die Batchverarbeitung. Jeder Befehl wird nach der Ausführung am Anfang der batch–queue gelöscht, nach der Befehlsausführung ist die batch–queue immer leer, man sollte sie also bei Bedarf vorher sichern. Wird ein Dekrementparameter, der an beliebiger Stelle für einen ASCII oder numerisches Zeichen in jedem Eingabeparameter stehen kann, auf “A” oder 1 zurückgezählt, so wird der nächste batch–Ladebefehl übersprungen. Damit lassen sich repeat ... until Schleifen nachbilden. Da manche Eingabewertbereiche kontextabhängig sind, kann es zu Situationen kommen, in denen batch–Eingaben nicht im erlaubten Wertebereich sind oder Dateien nicht existieren etc.. Die Verarbeitung wird dann mit einer Fehlermeldung angehalten und die ausstehenden Batch–Kommandos nach dem nächsten Tastendruck gelöscht.

Faltungs–/Rangoperationen

Die Matrizen–/ bzw. Fenstergröße kann bis zu $9*9$ Pixel betragen. Einer Erweiterung steht lediglich die komfortable Eingaberoutine ("1" bis "9" als Tastendruck) entgegen. Weil ein Binomialfilter der Ordnung neun zehn Koeffizienten hat, beträgt die Maximalgröße für Binomialfilter ausnahmsweise zehn Pixel. Werden keine Randwerte eingeblendet, werden die Bilder durch Fensteroperatoren um die $Breite\ der\ Fenster\ -\ 1$ kleiner. Außerdem verschiebt sich der Ursprung, weil für den ersten berechenbaren Bildpunkt die Matrix schon vollständig ins Bild geschoben werden mußte. Als Zentralpixel von Matrizenoperationen wird der Schwerpunkt der Beträge der Matrix gewählt. Die Verschiebung des Ursprungs wird mit einer Auflösung von 1/2 Pixel mitprotokolliert. Beim Filtern mit symmetrischen Matrizen gerader Länge treten 1/2 Pixel als Ursprungsoffset auf. Es handelt sich dann um Zwischengitterbilder. Bei doppelter Filterung mit solchen Operatoren kann das Ergebnis wieder für Gitterplätze gültig sein. Damit es bei dann wieder ganzzahligen Ursprungsoffsets nicht zu Rundungsfehlern kommt, ist die halbzahlige Pixelauflösung des Ursprungsoffsets nötig. So lassen sich die gefilterten Bilder wieder pixelgenau mit ihren Ursprungsbildern verknüpfen (z. B. zum Maskieren über ein Merkmal).

Differential– und Binomialfilter `d / b: x|y o n`

Vordefinierte Matrizen existieren bereits zur Differentiation $(-1, 1)$ mit anschließender Binomialfilterung (d) und zur Binomialfilterung selbst (b) jeweils mit wählbarer Matrixgröße in X– und Y–Richtung. Zweidimensionale Binomialfilter werden nicht vordefiniert unterstützt, sie sollten aus Effizienzgründen in X– und Y–Richtung separiert ausgeführt werden. Eine Differentiation 1. Ordnung im obigen Sinne ist z. B. der in X– Richtung separierte Teil eines X–Sobelfilters, zu dessen Vollendung dann noch mit einem Binomialfilter 2.Ordnung in Y–Richtung zu filtern ist.

Rangordnungsfilter `r: b h o`

Neben linearen Filtern wurden noch Rangordnungsfilter (r) aufgenommen. Das Filterergebnis ist das o–t größte Element (o) eines b breiten und h Pixel hohen Fensters. o ist im Bereich $0 < o <= b*h$ wählbar. Sortiert wird mittels Heapsort (die Sortierimplementation wurde aus [*Wirth 1986*] übernommen).

Allgemeine Matrixfaltung `m: b h t M`

Eine Faltungsmatrix kann auch diskret eingegeben werden. Nach der Matrixbreite b und –höhe h ist vor den Koeffizienten noch ein gemeinsamer Teiler t einzugeben. Danach werden die Koeffizienten der Matrix der Reihe nach als Integer eingegeben (M).

Gauß– und Laplacepyramiden `g / w: o [n e]`

Um Pyramiden effizient berechnen zu können, werden die Einzelbilder ihrer Ebenen hier in einem Bild zusammengefaßt, dessen Berechnung intern mit separierten Binomialmatrizen der Ordnung o erfolgt. Die Basis der zweiten Pyramidenstufe wird rechtsbündig neben das Originalbild gesetzt, die folgenden Stufen wiederum rechtsbündig darunter. Außer der Gaußpyramide (g) kann alternativ auch die Laplacepyramide (l) berechnet werden. In diesem Fall kann neben einem Shiftwert, der die Werte des Ergebnisbildes um Zweierpotenzfaktoren anhebt, noch ein reeller Verstärkungsfaktor angegeben werden, der die Kontrastanhebung zwischen den einzelnen Ebenen der Laplacepyramide (e) bestimmt (1.00 = Neutralstellung). Bei der Laplacepyramidenberechnung geht der Bildinhalt des Quellbildes verloren, bei der Gaußpyramide nur, falls das zweite Bild in X–Richtung größer als das Ursprungsbild der Gaußpyramide war. Neben den Pyramidenbilderinhalten kommen in den Bildern undefinierte Zonen vor, die immer gelöscht werden und bei folgenden Filterungen durch Randwerteffekte breiter werden können. Im Histogramm werden diese Pixel unterdrückt. Die Extremwertzählung wird jedoch verfälscht: Im *signed*–Bild entsprechen sie dem Wert 0 und im *unsigned*–Bild dem Wert 0.5 . Sind dies bezüglich der übrigen Pixel Extremwerte, verfälschen sie die Extremwerte und ihre Pixelzählung.

Randwert `t: a|r`

Mit dem Befehl (t) kann für Umgebungsoperatoren eine Einblendung von Randersatzwerten erreicht werden. Der am Rand einzublendende Wert kann absolut (a) oder als Linearkombination des Mittelwertes und der Standardabweichung des Destination–Bildes (r) angegeben werden. Da nur Ergebnispixel definiert sind, in deren Umgebung wiederum alle Pixel definiert sind, wird das Bild ohne Randersatzwerteinblendung bei Faltungen oder Rangordnungsoperationen stetig kleiner. War vorher die Randwertoption schon aktiv, wird sie wieder zurückgesetzt.

Zweistellige arithmetische Punktoperationen

Arithmetische Operationen `S / D / P / T: n`

Beim Teilen durch 0 wird auf den vorzeichenrichtigen Maximalwert umgeschaltet. Ansonsten entspricht die Operation den angegebenen Ausdrücken im Hauptmenü (Summe, Differenz, Produkt und Quotient), wobei d für das Destination– und s für das Source–Bild steht. Vor der Ergebnisquantisierung lassen sich die Werte noch je nach Funktion arithmetisch nach links oder rechts shiften (n).

Schwellenwert–/ Rangordnungsoperator `V / X: g|k`

Es findet bei beiden Funktionen ein Vergleich statt. Dieser kann je nach gesetztem zweiten Operand zwischen Bildern oder zwischen einem Bild und einem Schwellenwert stattfinden. Als Vergleichsoperator steht größer (g) oder kleiner (k) zur Verfügung. Das Ergebnis des

Vergleichs ist beim Compare–Befehl (V) eins für wahr und null für falsch. Beim Clip–Befehl wird stattdessen immer der Pixel rechts vom Vergleichsoperator übernommen, wenn der Vergleich wahr ist, ansonsten der linke. Der Effekt dabei ist, daß beim Größer–Vergleich (g) der größere Wert, beim Kleiner–Vergleich der kleinere Wert (k) übernommen wird. Mit diesen beiden Funktionen lassen sich somit Rangordnungen und Vergleiche zweier Bilder (Binärantwort) berechnen, bzw. Extremwerte in Bildern clippen und Schwellenwertvergleiche anstellen.

Konstante als zweiter Funktionswert `Y: a|r`

Für die jeweils folgende Punktoperation kann vom Source–Bild als zweiter Operand auf einen festen Skalarwert Y umgeschaltet werden, der dann als f(Bild, s = Y) angezeigt wird. Er kann absolut (a) oder als Linearkombination des Mittelwertes und der Standardabweichung des Destination–Bildes (r) angegeben werden. Ein doppeltes Aufrufen dieser Skalarwertoption setzt sie zurück; als zweiter Operand dient wieder das Source–Bild der letzten Operation.

parameterlose Punktoperationen

Betrags–, Exponential– und Logarithmusfunktion `B / E / L`

Diese Funktionen wurden der Vollständigkeit halber mit aufgenommen. Der Aufruf der Betragsfunktion (B) ist nur auf signed–Format Bilder sinnvoll, ansonsten entspricht er dem Befehl Kopieren (c). Für die Exponential– und Logarithmusfunktion wurden Kurvenausschnitte gewählt, die für alle Bildwerte definiert sind (die Konstante der Formeln betragen genaugenommen $1 + 1/512$) und weiterhin zueinander die Umkehrfunktion bilden.

Sonstiges

Die aktuellen Parameter beider Bilder im Hauptspeicher, das Bildformat signed oder unsigned (s/u), die Bildgröße und der eventuell halbzahlige Offset zum Ursprungsbild werden im Hauptmenü ständig angezeigt.

Bildgröße umrechnen `G: a|f b h`

Das Bild wird mittels Unter– bzw. Überabtastung auf eine neue Bildauflösung gebracht. Die neue Auflösung kann absolut (a) oder relativ (r) zur bisherigen Auflösung angegeben werden. Die Breite (b) und Höhe (h) wird entsprechend in Pixeln oder als Faktor eingegeben. Durch Herunterrechnen und anschließendes Zurückrechnen der Auflösung kann auch eine Rasterung eines Bildes erreicht werden. Dieser Befehl kann zu aliasing–Effekten führen. Ohne vorherige Filterung zur Einhaltung des Abtasttheorems sollte man ihn nicht unbedingt zum Umskalieren von Bildern gebrauchen.

Bild verschieben `O: l r o u`

Um beliebige Bildausschnitte isolieren oder das Bild um eine neutrale Umgebung vergrößern zu können, müssen hier neue Koordinaten für alle 4 Bildseiten relativ zu den alten angegeben werden. Für den linken Rand (l) bedeuten positive Werte, daß das Bild erst ab diesem Pixel beginnen soll, also links abgeschnitten wird; negative fügen links einen Rand hinzu, 0 ist der Neutralwert. Für den rechten Bildrand (r) verhält es sich genau umgekehrt. Für den unteren Rand (u) bedeuten negative Werte ein Abschneiden — die Y–Koordinatenachse zeigt nach unten —, positive das Hinzufügen eines Randes; für den oberen (o) gilt es wieder entsprechend umgekehrt. Falls das Bild zu groß wäre oder sich aus den Werten negative Randbreiten ergäben, werden die Werte für den rechten und unteren Bildrand auf die erlaubten entsprechenden Extremwerte korrigiert. Zum Auffüllen wird der unter den Punktoperationen gewählte Wert genommen, oder falls dort keiner gesetzt ist, der arithmetische Mittelwert des aktuellen Wertebereichs, 0 für signed– bzw. 0.5 für unsigned–Bilder.

zum Protokollmode `M`

Das Programm kann sich im Protokollmode oder im Execute–Mode befinden. Der Protokollmode dient dazu, daß Kommandos in die batch–Datei geschrieben werden können, ohne ihre Ausführung abwarten zu müssen. Es wird die eigentliche Berechnung von Pixeln übersprungen. Die Bildlagekoordinaten werden nichtsdestotrotz nach jeder Operation aktualisiert. Die Bildanzeigeoptionen werden natürlich auch nur protokolliert. Kehrt man zum Executemode zurück, werden alle Bilder, die keinen sinnvollen Inhalt mehr haben, auf 0 Pixel Größe gesetzt.

Integer–/Charakter–Dekrementparameter `I / N`

Diese Parameter ersetzen in allen alphanumerischen Eingaben von Parametern Zahlzeichen oder alphabetische Zeichen. Sie können neu gesetzt werden oder mit dem in der Eingabezeile angezeigten Sonderzeichen dekrementiert werden. Beide Parameter können nicht nur als Variable z. B. in Dateinamen oder Filterordnungsparametern zur Berechnung von Bildreihen oder Parameterstudien dienen, sondern haben auch noch den Seiteneffekt, daß, falls von 1 oder “A”, ihrem Minimalwert, aus versucht wird, sie zu dekrementieren, stattdessen der nächste Batch–Ladebefehl übersprungen wird. Damit lassen sich Batch–Schleifen zu Parameterstudien oder zum automatischen Verarbeiten mehrerer Bilder schreiben. Ist dieser Effekt unerwünscht, empfiehlt es sich, nachfolgend eine leere batch–Datei im Einfügemodus zu laden.

Current Directory `C`

Der Befehl setzt das aktuelle Verzeichnis für alle Dateibezüge des Programms. Es ist der absolute Pfad oder der relative — bezüglich des vor Programmstart aktuellen Verzeichnisses — anzugeben.

Merkmalsfile `W`

Dieser Befehl definiert eine Merkmalsdatei, in die Bildkoordinaten zusammen mit einer vorgegebenen Anzahl (W) von Werten geschrieben werden. Der Zahlparameter legt fest, wieviele Benutzer–Merkmale zu jeder Koordinate abgefragt werden. Existierte die Datei nicht, wird sie mit einem fileheader als Inhalt neu angelegt, ansonsten wird sie als ".bak" Datei kopiert und fortgeschrieben. Die Datei dient als Eingabe für das Programm *cluster* zur graphischen Darstellung von Merkmalsvektoren im Merkmalsraum.

Dazu müssen die Pixel der darzustellenden Merkmalsvektoren zusammen mit einer Sollklassifikation im Bildanzeigemodus (v) eingegeben und bis zu drei Dateinamen bereits berechneter klassifizierender Merkmalsbilder im Header des Merkmalsfiles nachgetragen werden. Die entsprechenden Merkmalsvektoren werden dann aus den Bilddateien ausgelesen und als Projektion dargestellt.

VGA–/VESA–Modus `K`

Wechselt zyklisch zwischen einigen VGA– und VESA–Modi, die zur Bildanzeige dienen. Die VGA–Modi 320 * 200 Pixel in 256 und 640 * 480 in 16 Farben können auf allen PCs genutzt werden. Mit 2 MByte VESA–Grafikkarten sind bis zu 1280 * 1024 Pixel, bei 1 MByte Karten bis zu 1024 * 768 Pixel in 256 Farben möglich. VESA–Modi können auf allen VESA–Karten mit 64 KB Schreibpuffer genutzt werden (getestet auf Spea–Mirage und ELSA–Winner 1000). Der Bildspeicher wird aus Geschwindigkeitsgründen direkt angesprochen. (Alle Modi ab 640 * 480 Pixel in 256 Farben sind VESA–Modi und lassen sich nur mit VESA–Karten korrekt darstellen!). Jeder Graphikmodus existiert zweimal. Einmal wird nur das Destination–Bild (bezüglich der letzten Operation) angezeigt, das andere mal beide Bildspeicher nebeneinander. Nach jedem anderen Befehl toggelt dieser Befehl zuerst beide Varianten, bevor er die weiteren Videomodi durchgeht.

Anhang C

Beweis zur Multiplikationsrekursion

Lemma 1: Alle Knoten auf demselben Niveau haben als Bezeichner k oder $k+1$.

Induktionsbeweis: $b(n)$ sei die Menge der Bezeichner der Knoten mit Niveau n.

Verankerung:

$$\begin{array}{rcllll} b(0) & = & \{g\} & & & f\ddot{u}r \quad k=g \\ b(1) & = & \{g/2, g/2+1\} & : & g \quad ungerade & f\ddot{u}r \quad k=g/2 \\ & = & \{g/2\} & : & g \quad gerade & f\ddot{u}r \quad k=g/2 \end{array} \tag{C.1}$$

Annahme:

$$b(n) \subseteq \{h, h+1\} \tag{C.2}$$

Schritt:

$$b(n+1) \subseteq \left\{ \begin{array}{llll} \{h/2, h/2+1\} & : & h \quad ungerade & f\ddot{u}r \quad k=h/2 \\ \{h/2, h/2+1\} & : & h \quad gerade & f\ddot{u}r \quad k=h/2 \end{array} \right. \tag{C.3}$$

Die Aufteilung von h ist nur zum Beweis sinnvoll: Folgende Tabelle von Fallunterscheidungen für alle sich ergebenden Knotenbezeichner des Niveau $n+1$, abhängig davon, ob der kleinere Bezeichner auf Niveau n gerade oder ungerade ist, beweist den Induktionsschritt:

$b(n) \subseteq \{h, h+1\}$	$\{h,$		$h+1\}$		$b(n+1) \subseteq$
Zerlegung	$(h+1)/2$	$h/2$	$(h+1)/2$	$h/2+1$	$\{h/2, h/2+1\}$
h *gerade*	$h/2$	$h/2$	$h/2$	$h/2+1$	$\{h/2, h/2+1\}$
h *ungerade*	$h/2+1$	$h/2$	$h/2+1$	$h/2+1$	$\{h/2, h/2+1\}$

(C.4)

(Bemerkung: Der Fall $b(n) = \{h\}$, h gerade, führt als Spezialfall wieder zu einer einelementigen Menge $b(n+1) = \{h/2\}$. Ist k = 2^n, gibt es im ganzen Zerlegungsbaum pro Niveau nur einen Bezeichner für die Knoten.) □

Behauptung: $b(n) \in \{\{0,1\},\{1\}\}\ :\ n >= ceiling(ld(g)),\ g > 0$

Beweis: Die Summe aller Knotenbezeichner desselben Niveaus ist eine Invariante (Zerlegungsknoten: $h/2 + (h+1)/2 = h$). Erst wenn die Knotenanzahl 2^n eines Niveaus n größer als g ist, muß es Knoten dieses Niveaus mit dem Bezeichner 0 geben. Für $g > 0$ muß es auch Knoten dieses Niveaus mit Bezeichnern größer gleich 1 geben. Nach Lemma 1 gibt es keine anderen Bezeichner für Knoten dieses Niveaus als 0 und 1. Ist die Knotenzahl 2^n gleich g, so sind alle Knoten des Niveaus n mit 1 bezeichnet, da $b(n) = \{g/2^n\}$ (s. o. Spezialfall)

Gilt für eine Niveau n $b(n) \in \{\{0,1\},\{1\}\}$, können Knoten des Niveaus $n-1$ nur die Bezeichnermengen $b(n-1) \in \{\{1,2\},\{2\}\}$ haben. Die Bezeichnermenge $b(n-2) \in \{\{2,3\}, \{3\}, \{3,4\}, \{4\}\}$, enthält also keine Einsen mehr. Aufgrund von Lemma 1 lassen sich die drei letzten Schritte aller möglichen Zerlegungen aufzählen. Man findet die Behauptung dabei bestätigt:

$$\begin{array}{ccccc} \{4\} & \rightarrow & \{2\} & \rightarrow & \{1\} \\ \{3,4\} & \rightarrow & \{2,1\} & \rightarrow & \{1,0\} \\ \{3\} & \rightarrow & \{2,1\} & \rightarrow & \{1,0\} \\ \{2,3\} & \rightarrow & \{2,1\} & \rightarrow & \{1,0\} \end{array} \tag{C.5}$$

Damit ist die Behauptung durch vollständige Aufzählung der möglichen Zerlegungen der letzten 3 Niveaus bewiesen. □

Das Beispiel einer Multiplikatorzerlegung in Abb. C.1 greift die Beispielmultiplikation aus Abb. 6.1 auf und zeigt den Zusammenhang mit den im Beweis verwandten Variablen.

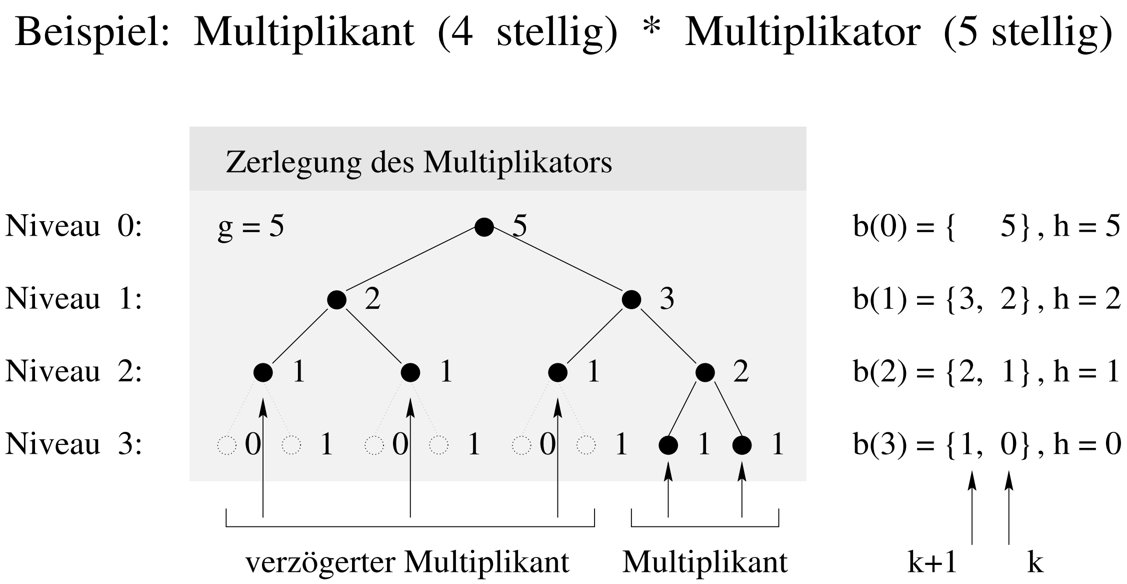


Abbildung C.1: Multiplikationszerlegung

Anhang D

Bildtafeln

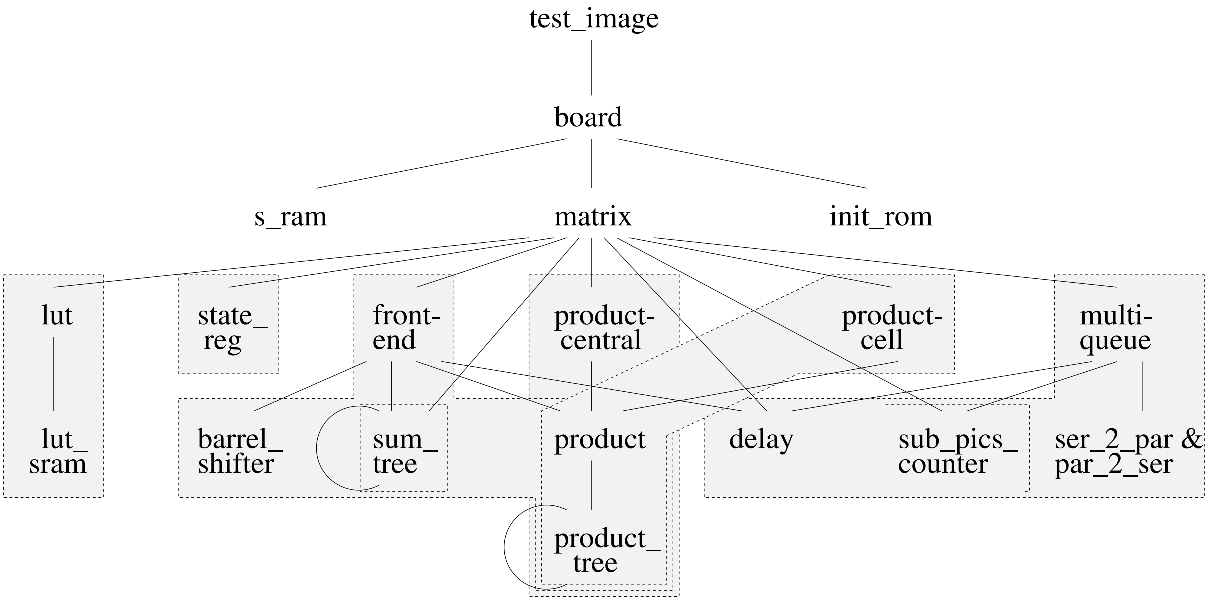


Abbildung D.1: Komponentenhierarchie des ASIC

```
Bildverarbeitung im Ortsraum: (8 Bit, max 3072 * 527) Ver 1.82  Q to EXIT

Eingabe/Ausgabe                         Stapelverarbeitung
--------------------------------        ---------------------------------
l. Bild laden          p|b [s|u]        J. Save Batchjob
s. Bild speichern      p|b              R. Load Batchjob                  a|n
c. Bild kopieren                        F. Ersten  Eintrag löschen
x. Bildpuffer tauschen                  U. Letzten Eintrag löschen
v. Bild linear anzeigen cursr|CR        A. Batchjob ausführen
e. Bild eingeebnet -"-  cursr|CR        Z. Zyklisch tauschen
h. Histogramm/ Statistik

                                        Punkt-Punkt/Skalaroperationen
Faltungs-/Rangoperationen               ---------------------------------
--------------------------------        S. Summe   (d + s) / 2 * 2^n:   n
d. Different. / 2 * 2^n: x|y o n        D. Differ. (d - s) / 2 * 2^n:   n
b. Binomi.-filter * 2^n: x|y o n        P. Produkt (d * s)     * 2^n:   n
r. Rangordnungsfilter:   b h o          T. Teilen  (d / s)     / 2^n:   n
m. all. Matrixfaltung:   b h t M        V. Compare (d o s) ? 1 : 0  : g|k
g. Gaußpyramide berechnen: o            X. Clippen (d o s) ? s : d  : g|k
w. Laplacepyramide berech: o n e        Y. f(Bild, s = zweites Bild): a|r
t. Randwert ( ja )  0.500  : a|r

                                        Sonstiges
Parameterlose Punktoperationen          ---------------------------------
--------------------------------        source: u (0.0,0.0) (3072,527)
B. Betrag bilden                        destin: s (1.5,1.0) (3069,525)
E. Exponential exp a - 1                G. Bildgröße umrechnen:   a|f b h
L. Logarithmus ln (a + 1)               O. Bild verschieben   :   l r o u
                                        M. zum Protokollmode
                                        I. Intg.dekr.param: '#' = 0
Wahl: Q                                 N. Char.dekr.param: '$' = A
                                        C. Curr.Dir.: d:\holz_pic\
                                        W. Merk.File: none
                                        K. Destination in: 1280*1024, 256


Batch:

l\$_quart,p b/x,2,0 b/y,1,0 w/8,2,1.33 Y/a,1.0 S/0 s/\$_laplc,p  _ _ _ _
_ _ _ _ _ _ d/x,1,1 s/dif_x,p l/\$_laplc,p  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ d/y,1,1 s/dif_y,p c P/3 s/dy_dy,p _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ x l/dif_x,p P/3 s/dx_dy,p _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ x c _ _ _ _ P/3 s/dx_dx,p R/orientr2,n v
```

Menü des Bildverarbeitungsprogramms

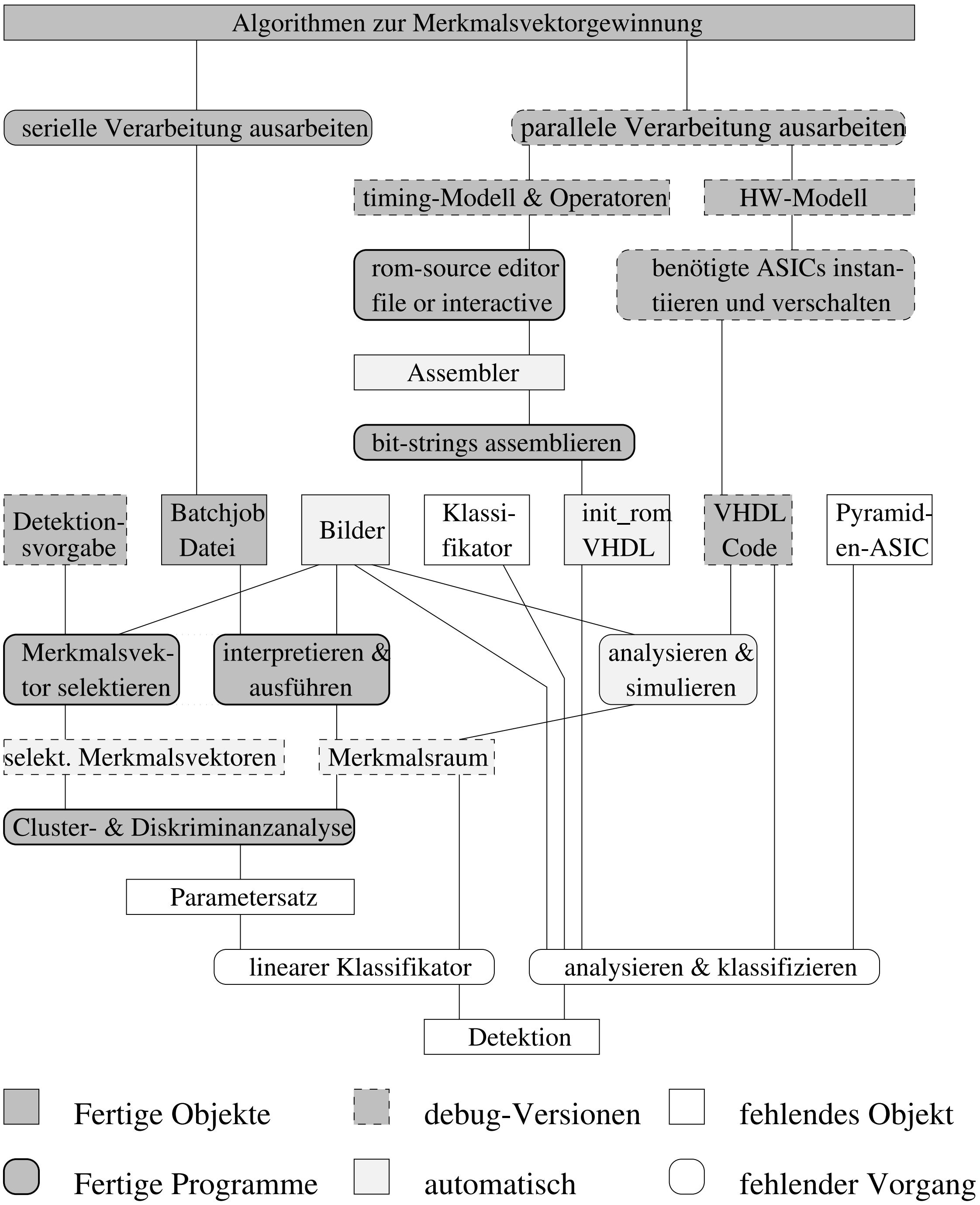


Abbildung D.2: Arbeiten und Stand des Gesamtprojektes

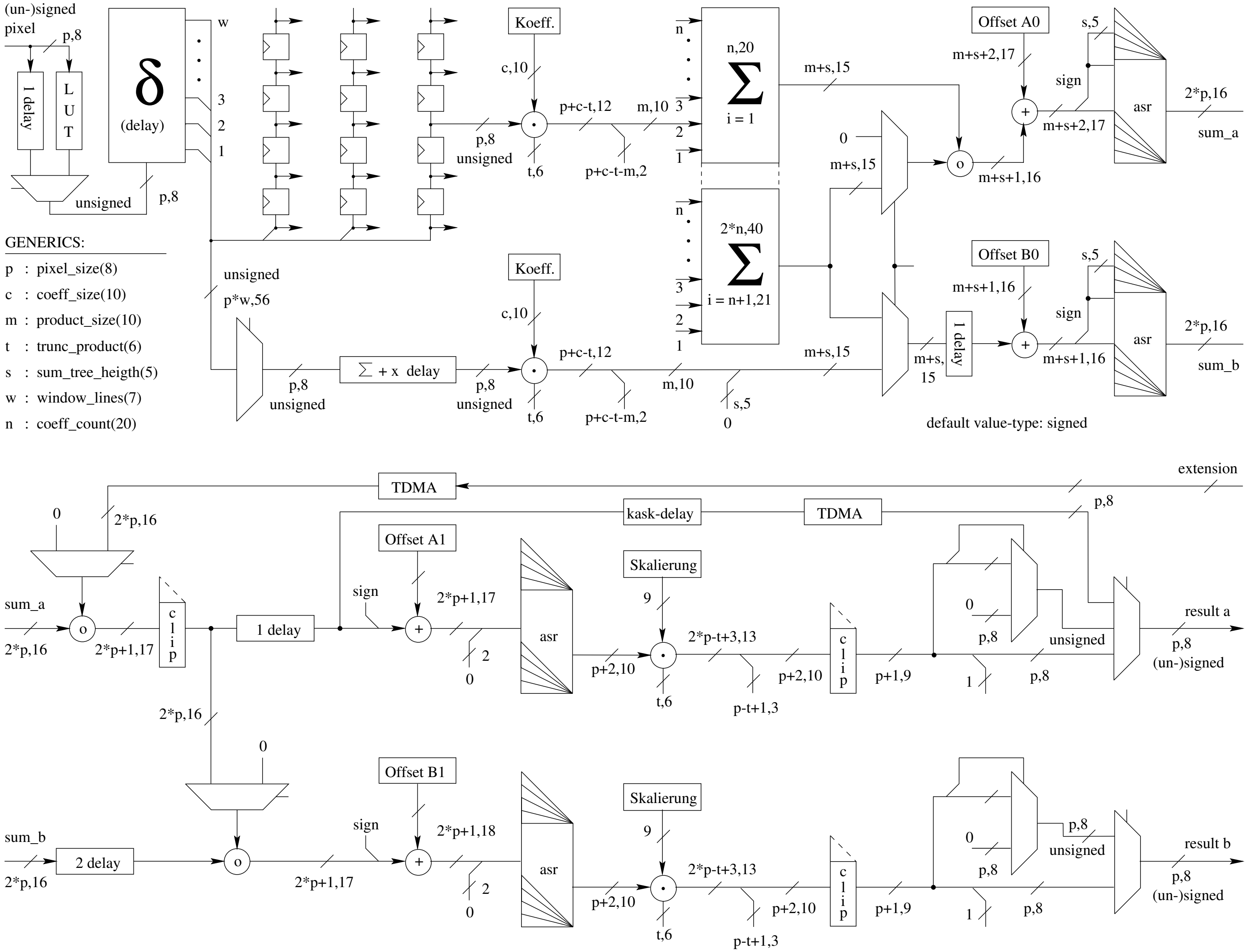


Abbildung D.3: Datenfluß (Matrix konfiguriert dargestellt)

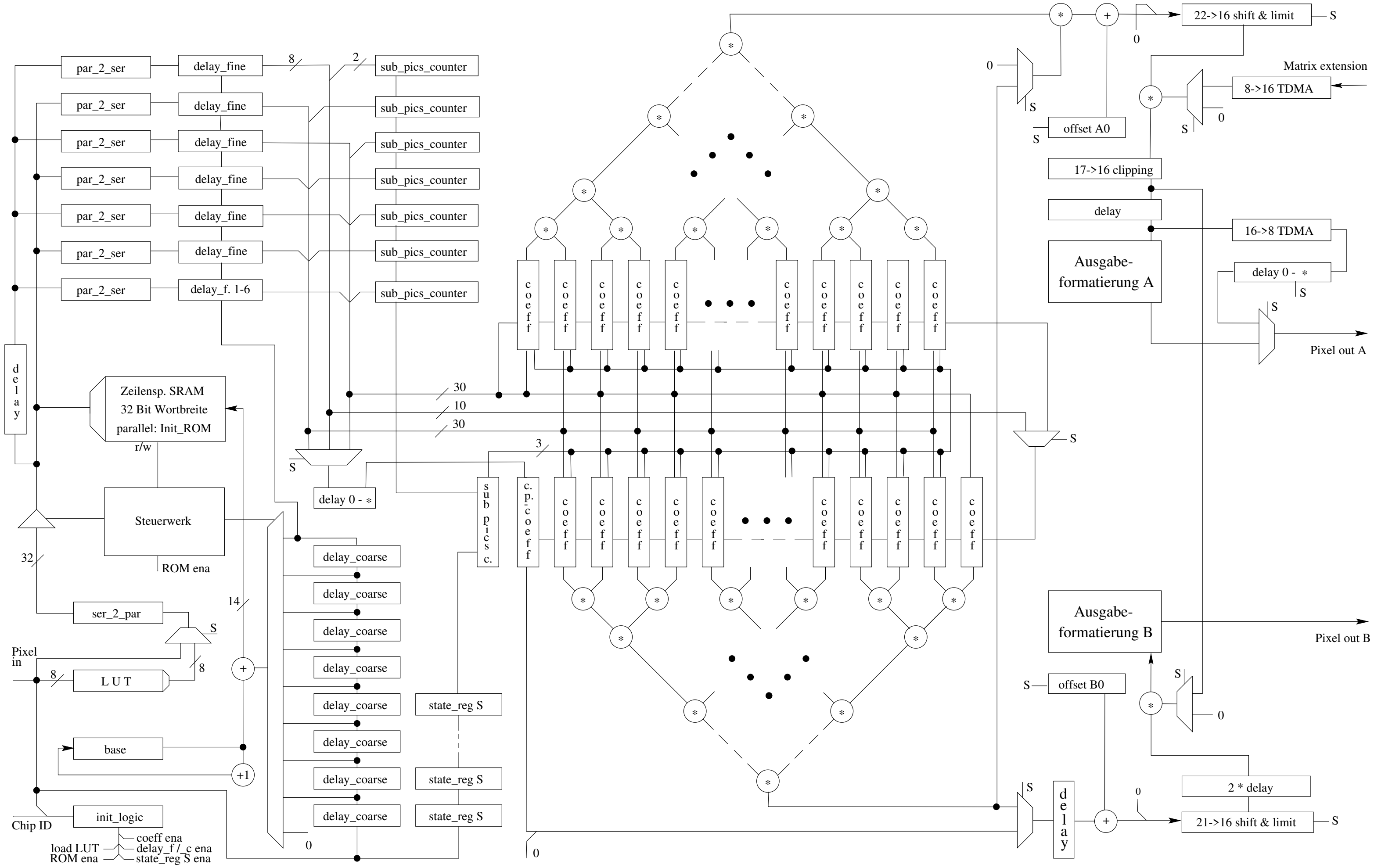


Abbildung D.4: Strukturbild

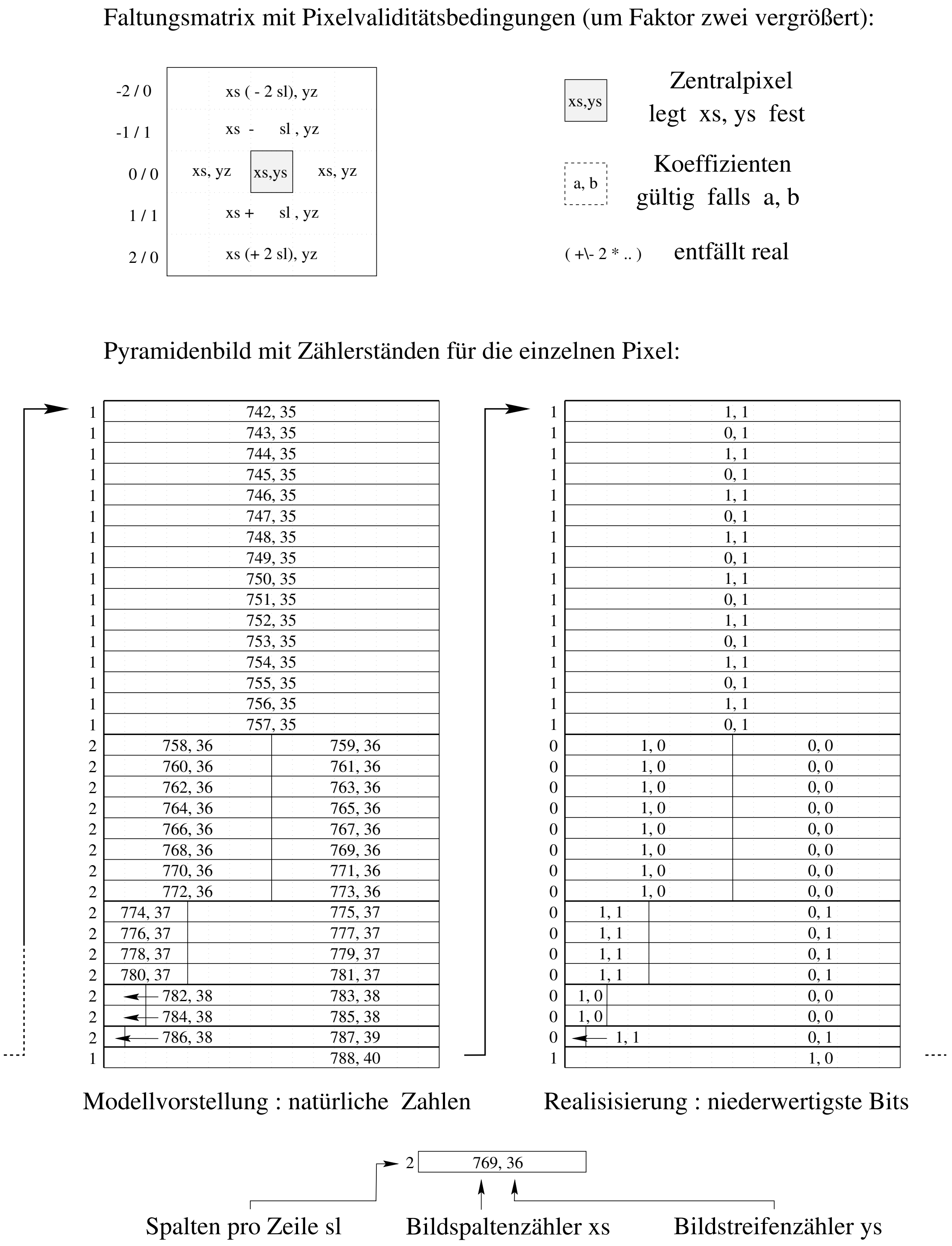
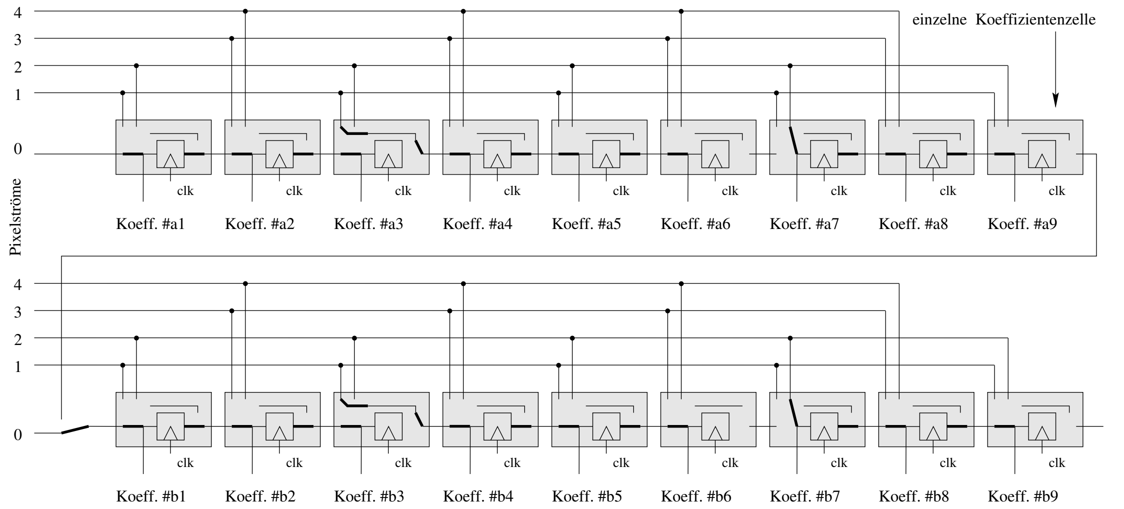


Abbildung D.5: Bildbereichszähler



Konfiguration für einen 18 Koeffizienten-ASIC:

Zwei 3 * 3 Matrizen

9a	8a	7a	← Pixelstrom 2
6a	5a	4a	← Pixelstrom 1
3a	2a	1a	← Pixelstrom 0, unverzögert

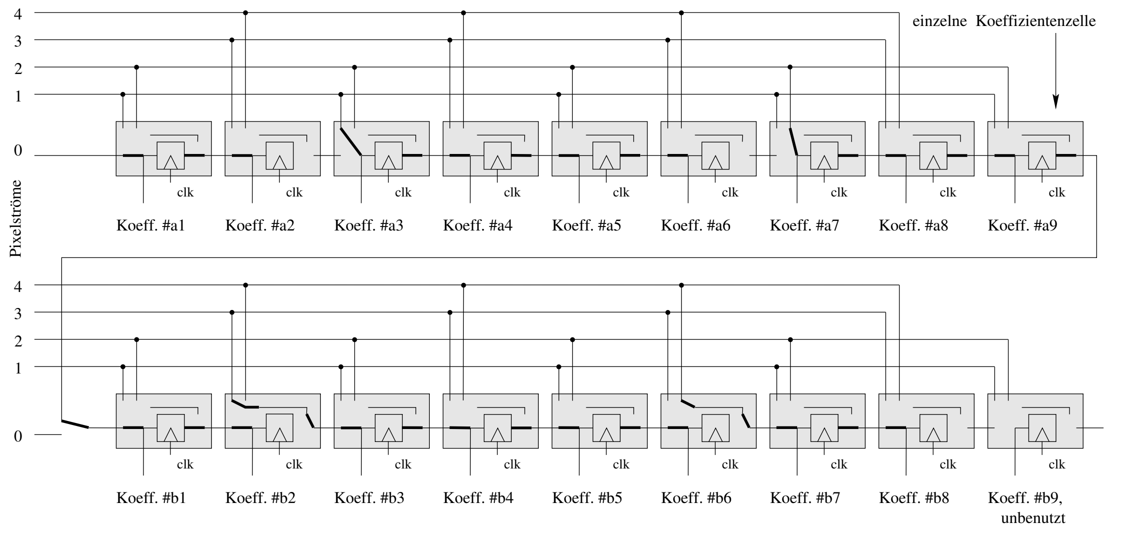
Koeffizienten Teilmatrix a

9b	8b	7b	← Pixelstrom 2
6b	5b	4b	← Pixelstrom 1
3b	2b	1b	← Pixelstrom 0, unverzögert

Koeffizienten Teilmatrix b

Pixelstrom 2: verzögert um zwei Zeilen
Pixelstrom 1: verzögert um eine Zeile
Pixelstrom 0: unverzögerter Eingabepixelstrom

— Hardwaremäßig festhelegte Verdrahtung
— Über Multiplexer konfigurierbare Verbindungen



Konfiguration für einen 18 Koeffizienten-ASIC:

Radiale 5 * 5 Matrix

		8b	7b		← Pixelstrom 4
6b	5b	4b	3b		← Pixelstrom 3
2b	1b	9a	8a	7a	← Pixelstrom 2
	6a	5a	4a	3a	← Pixelstrom 1
	2a	1a			← Pixelstrom 0, unverzögert

Koeffizienten der Gesamtmatrix

Pixelstrom 4: verzögert um zwei Zeilen
Pixelstrom 3: verzögert um zwei Zeilen
Pixelstrom 2: verzögert um zwei Zeilen
Pixelstrom 1: verzögert um eine Zeile
Pixelstrom 0: unverzögerter Eingabepixelstrom

— Hardwaremäßig festgelegte Verdrahtung
— Über Multiplexer konfigurierbare Verbindungen

Abbildung D.6: Konfiguration der Koeffizientenzellen

Abbildung D.7: 1. Orientierungsvektorkomponente, klassifizierendes Merkmal #1 (auf einem PC mit dem Programm *bild_ver* berechnet)

Abbildung D.8: 2. Orientierungsvektorkomponente, klassifizierendes Merkmal #2 (auf einem PC mit dem Programm *bild_ver* berechnet)

Abbildung D.9: Texturintensität, klassifierendes Merkmal #3
(auf einem PC mit dem Programm *bild_ver* berechnet)

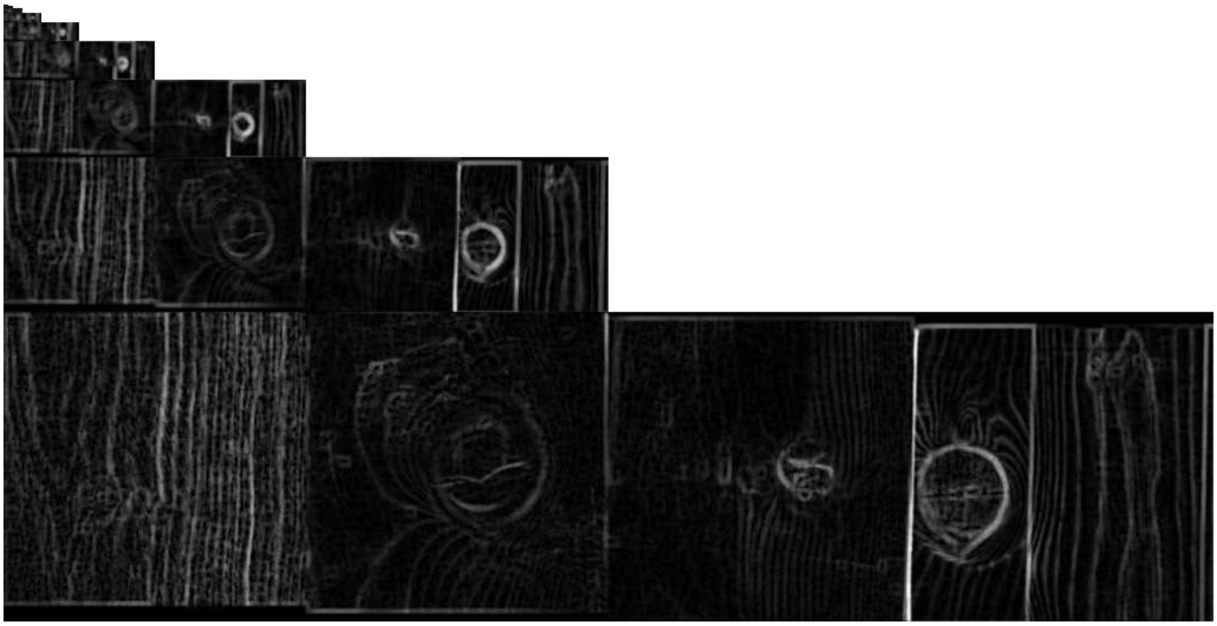


Abbildung D.10: Drittes klassifizierendes Merkmal: Dilatationsverfahren
(Ersten beiden Merkmale: s. Abb. 5.1 u. 5.2)
(auf einem PC mit dem Programm *bild_ver* berechnet)

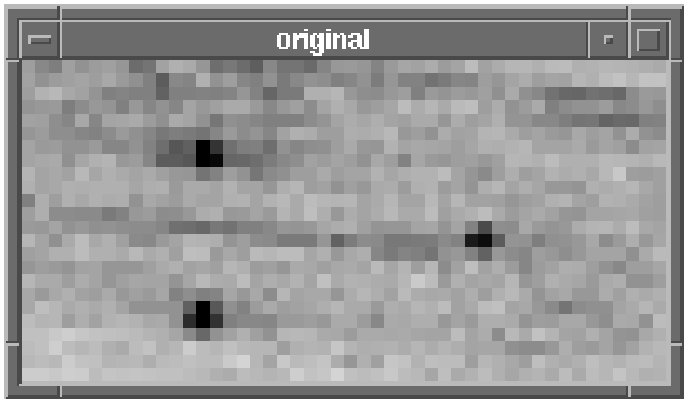


Abbildung D.11: Eingabebild (unangepaßter Randeinblendewert)

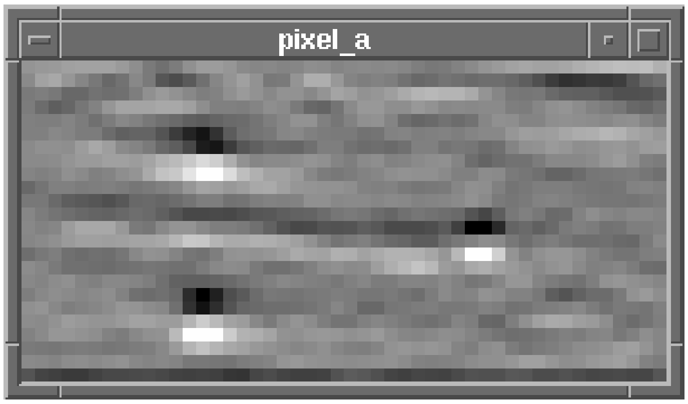


Abbildung D.12: Y–Sobelfilterung (von dem simulierten ASIC berechnet!)

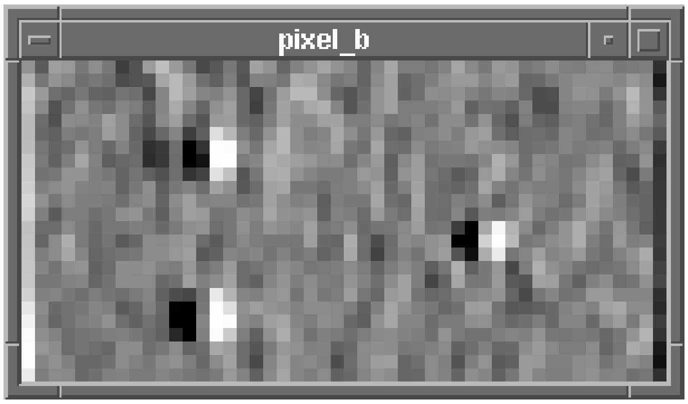


Abbildung D.13: X–Sobelfilterung (von dem simulierten ASIC berechnet!)

Literaturverzeichnis

[*Bronstein 1962*] Bronstein, I. N.; Semendjajew, K. A.:
Taschenbuch der Mathematik.
Frankfurt/Main: Harri Deutsch, 1962.

[*c't, 9.91 1991*] Herausg. Heise, C.:
c't. 9.91. magazin für computer technik.
Hannover: Heinz Heise, 1991.

[*Cadence 1994*] Cadence Design Systems, Inc:
9401 Cadence Online Library.
(Cadence DFW2 v4.3. sub–version 4.3.130)
San Jose, CA 95134,
USA: Cadence Design Systems,
Inc. 555 River Oaks Parkway, 1994.

[*Deichsel, Trampisch 1985*] Deichsel, G.; Trampisch, H.J.:
(Biometrie) Clusteranalyse und Diskriminanzanalyse.
Stuttgart, New York: Gustav Fischer, 1985.

[*Dreschler–Fischer 1994*] Dreschler–Fischer, L.:
Bildverstehen.
Prüfungsunterlagen für die Vorlesung im SoSe 1994.
Labor für künstliche Intelligenz,
Universität Hamburg, SoSe 1994.

[*Dwersteg 1995*] Dwersteg B.:
Konzeption und Entwicklung eines Histogrammers als ASIC.
Diplomarbeit Informatik, Universität Hamburg, 1995.

[*Eschermann 1993*] Eschermann, B.:
Funktionaler Entwurf digitaler Schaltungen.
Berlin, Heidelberg: Springer–Verlag, 1993.

[*Fleischer 1993*] Fleischer, A. G.:
Modellgesteuerte Detektion von optischen Oberflächenstrukturen des Holzes.
Interner Bericht des Arbeitsbereiches.
Arbeitswissenschaft, Universität Hamburg, 1993.

[*Hendrich 1994*] Hendrich, N.:
XCLI VHDL–Erweiterung: outputWindow.vhd.
Universität Hamburg,
Arbeitbereich TECH: interne Arbeit, 1994.

[*Hübner 1990*] Hübner, G.:
Stochastik.
Vorlesungsskript: Mathematik für Informatiker.
Institut für Mathematische Stochastik,
Universität Hamburg, WS 1990/91.

[*Jähne 1991*] Jähne, B.:
Digitale Bildverarbeitung.
Berlin, Heidelberg: Springer, 1991.

[*Kempendorf 1993*] Kempendorf, O.:
Optische Detektion mit Hilfe digitaler Bildverarbeitung.
Diplomarbeit Biologie, Universität Hamburg, 1993.

[*Klette, Zamperoni 1992*] Klette, R.; Zamperoni. P.:
Handbuch der Operatoren für die Bildverarbeitung
Braunschweig, Wiesbaden: Vieweg, 1992.

[*Kolla et al. 1989*] Kolla, R.; Molitor, P.; Osthof, H.:
Einführung in den VLSI–Entwurf.
Stuttgart: B. G. Teubner, 1989.

[*Lagemann 1987*] Lagemann, K.:
Rechnerstrukturen.
Verhaltensbeschreibung und Entwurfsebenen.
Berlin, Heidelberg, New York: Springer, 1987.

[*Rosenstiehl, Camposano 1989*] Rosenstiehl, W.; Camposano, R.:
Rechergestützter Entwurf hochintegrierter MOS–Schaltungen.
Berlin, Heidelberg: Springer, 1989.

[*Synopsys 1995*] Saliba, D. R.; Robiola, J.:
Synopsys Online Documentation. Version Number 3.3a.
Mountain View, CA 94043–4033, USA: Synopsys, Inc. 700 East Middlefield Road, 1995.

[*Wirth 1986*] Wirth, N.:
Algorithmen und Datenstrukturen mit Modula–2.
Stuttgart: B. G. Teubner, 1986.