5.5.4 Synchronisation externer Signale

Der Histogrammer führt sämtliche internen Vorgänge synchron zu seiner eigenen Taktfrequenz durch und liefert Ausgangssignale synchron zu diesem Takt. Alle Eingangssignale des Histogrammers dürfen jedoch asynchron zum Takt des Histogrammers liegen. So werden die Kameradaten z.B. mit der kameraspezifischen Pixelfrequenz geliefert. Der Hostrechner greift synchron zu seinem eigenen Bustakt auf die Busschnittstelle des Histogrammers zu. Deshalb müssen die Signale dieser Schnittstellen innerhalb des Histogrammer synchronisiert werden. In den Synchronisations-Flip-Flops können jedoch metastabile Zustände auftreten. Wie in Kapitel 4.6.3 erläutert, ist eine möglichst lange Einstellzeit für die Synchronisations-Flip-Flops vorzusehen, um Fehler durch metastabile Zustände unwahrscheinlich zu machen.

Der Histogrammer wertet Daten an der Pixelschnittstelle nur aus, wenn zuvor eine steigende Flanke des Pixeltaktes erkannt wurde und das Start-Signal aktiv ist. Deshalb kann bei der Pixelschnittstelle nur Metastabilität in den Synchronisationsschaltungen für das Start-Signal und den Pixeltakt zu Fehlern führen (s. Abbildung 4.14, Abbildung 4.15). Da der synchronisierte Pixeltakt in sehr vielen Komponenten des Histogrammers ausgewertet wird, ist es kaum möglich, eine niedrige maximale Laufzeit dieses Signals zu allen darüber gesteuerten Flip-Flop-Dateneingängen zu garantieren. Um eine definierte Zeit für die Einsynchronisation des Pixeltaktes zu garantieren, wird in die Synchronisationsschaltung eine zusätzliche Pipelinestufe eingefügt, die das Synchronisations-Flip-Flop enthält (zwischen FF1 und FF2 in Abbildung 4.19). Damit die übrigen Eingangssignale der Pixelschnittstelle weiterhin im gleichen zeitlichen Verhältnis zum Pixeltakt stehen, muß auch hinter die Dateneingänge der Pixelschnittstelle eine zusätzliche Pipelinestufe geschaltet werden. Diese zusätzliche Pipelinestufe verzögert die Bildauswertung lediglich um einen Histogrammer-Takt.

Entsprechende Überlegungen wie für die Steuersignale der Pixelschnittstelle gelten auch für die Steuersignale der Busschnittstelle (/cs, /rd, /wr, s. Abbildung 4.10, Abbildung 4.12). Die hier für Synchronisation benötigte Zeit addiert sich allerdings zu der Zugriffszeit für einen Schreib- oder Lesezugriff und macht den Zugriff auf den Histogrammer langsamer. Deshalb wird nur eine halbe Taktperiode abzüglich der Laufzeit der den Synchronisations-Flip-Flops nachgeschalteten Logik für die Synchronisation von Eingangssignalen zur Verfügung gestellt (s. Kapitel 4.6.3). Anhand der Unterlagen zu den im Compass-System verfügbaren Synchronisations-Flip-Flops [33] soll im folgenden geprüft werden, ob so eine genügend hohe Betriebssicherheit erreicht wird.
In dem Handbuch zur Standardzellbibliothek wird für die mittlere Zeit zwischen zwei Fehlern durch Metastabilität bei Verwendung des Synchronisations-Flip-Flops VSC450 DFNTNS folgende Formel angegeben (MTBF = Mean Time Between Failures):

$$MTBF[\text{sec}] = \frac{\exp(T_{PD} / \tau)}{(wo) \cdot (f_{CLK}) \cdot (2 f_{DATA})}$$

In der Gleichung ist:

wo	eine experimentell ermittelte Zeitkonstante, die sowohl die nominelle Laufzeit innerhalb des Flip-Flops zwischen Takteingang und Ausgang berücksichtigt als auch die Empfindlichkeit des Flip-Flops für metastabile Zustände.
τ	die experimentell ermittelte exponentielle Metastabilitäts-Abklingzeitkonstante des Flip-Flops.
$2\,f_{DATA}$	die mittlere Änderungsrate des Dateneingangs. Dies ist das Doppelte der Frequenz, die am Dateneingang anliegt.
f_{CLK}	die Taktfrequenz des Flip-Flops.
T_{PD}	die in der Schaltung maximal zulässige Dauer eines metastabilen Zustandes. Ein Überschreiten dieser Dauer kann zu einem Fehler in der nachfolgenden Logik führen.

Die Gleichung zeigt die negativ exponentielle Abhängigkeit der Fehlerwahrscheinlichkeit von der zur Synchronisation zur Verfügung stehenden Einstellzeit und die lineare Abhängigkeit von der Häufigkeit der Einsynchronisation eines neuen Pegels. In die Konstanten fließt auch die Länge des Zeitbereiches ein, innerhalb der eine Pegeländerung am Dateneingang zur Einspeicherung eines undefinierten Pegels führen kann.

Von Compass werden folgende Parameter für den *Worst-Case* angegeben:

wo = 140 ns

τ = 0,081 ns

Um den Minimalwert der MTBF zu ermitteln, müssen *Worst-Case*-Werte eingesetzt werden, d.h. maximale Frequenzen, schlechtestes Tastverhältnis und maximale Signallaufzeit.
Im Businterface des Histogrammers können folgende Parameter als *Worst-Case*-Werte angenommen werden:

f_{CLK} = 50 MHz (maximale Taktfrequenz)

$2\,f_{DATA}$ = 20 MHz (maximale Zugriffshäufigkeit)

T_{PD} = 4 ns (Mit Timing-Verifier ermittelter Wert: Periodendauer - 4 ns= 8 ns - 4 ns.)

Mit diesen Werten ergibt sich für die MTBF:

$$\frac{\exp(4 / 0{,}081)}{140ns \cdot 50MHz \cdot 20MHz} = 1{,}99 \cdot 10^{13}\,Sekunden = 6{,}33 \cdot 10^{5}\,Jahre$$

Diese Zeit ist um viele Zehnerpotenzen höher als die MTBF aufgrund anderer Fehlerursachen. Viel häufiger treten beispielsweise sporadische Fehler durch Höhenstrahlung auf oder dauerhafte Defekte durch Leiterbahnmigration oder Abreißen der Bonddrähtchen aufgrund thermischer Schwankungen. Damit ist die ermittelte *Worst-Case*-Zeit für T_{PD} genügend groß.

5.6 Datenpfad-Compiler

Alternativ zur Erstellung einer Schaltung über ein VHDL-Verhaltensmodell kann eine Schaltung direkt innerhalb von Compass auch grafisch definiert werden. Effiziente Schaltungsentwürfe von Datenpfaden können mit dem Datenpfad-Compiler erstellt werden. Dieses Werkzeug erlaubt eine hardwarenahe Definition von Einheiten, die häufig in Datenpfaden Verwendung finden, wie ALUs, Schieberegister, Addierer usw.. Jeweils eine einzelne Funktionseinheit eines Designs mit einem Datenpfad konstanter Breite kann innerhalb einer Zeichnung entworfen werden. Die Funktionseinheit wird erstellt, indem Schaltungselemente aus einer Bibliothek entnommen und in der Zeichnung plaziert und verdrahtet werden. Alle Elemente und alle Leitungen werden gemäß einer für die Zeichnung spezifizierten Breite mehrfach parallel angelegt. Kontrollsignale wie beispielsweise Steuersignale für Multiplexer

werden für jedes Bit parallel geschaltet und stehen automatisch als Anschlüsse der Funktionseinheit zur Verfügung. Ein Icon der Funktionseinheit mit einem Symbol für eine Datenpfad-Compiler-Einheit und mit sämtlichen Anschlüssen wird automatisch erstellt (s. Anhang C).
Ein solcher Entwurf kann grafisch manuell eingegeben werden oder automatisch aus Teilen eines VHDL-Entwurfes generiert werden. Bei automatischer Erzeugung von Datenpfad-Einheiten aus einer VHDL-Beschreibung legt der Compiler anhand vorgegebener Kriterien Datenpfad-Compiler-Einheiten an.
Die Option der automatischen Erzeugung von Datenpfad-Compiler-Einheiten war nur vorübergehend am MAZ verfügbar und konnte daher für den Chip nicht genutzt werden.

Die im Datenpfad-Compiler entworfenen Einheiten können entweder direkt in einen Hardware-Block auf dem Chip umgewandelt werden oder auf eine Netzliste abgebildet und über Standardzellen realisiert werden. Für direkte Abbildung auf Hardware sind im Handbuch Zeitverhalten und Platzbedarf der Bausteine in Abhängigkeit von der Anzahl parallel zu bearbeitender Bits aufgeschlüsselt. Anhand dessen kann die Schaltung für die gewünschte Taktfrequenz ausgelegt werden. Jeder so entworfene Datenpfad ergibt einen rechteckigen Block auf dem Chip. Die Anordnung, Anzahl und der Typ der Einheiten innerhalb des Blockes auf dem Chip gehen direkt aus der der grafischen Beschreibung hervor. Die Breite des Blockes hängt von der Anzahl der benötigten unterschiedlichen Einheiten und deren jeweiligem Platzbedarf ab. Die Höhe eines Blockes hängt nur von der Anzahl der parallel zu errechnenden Bits ab. Diese festen Geometrien bieten den Vorteil kalkulierbarer Laufzeiten und kompakter Blöcke, stellen jedoch beim Layout eines Chips unter Umständen ein Problem dar, da sie nicht in Arrays der konstant gleichhohen Standardzellen einzugliedern sind. Gegenüber einer VHDL-Beschreibung ergibt sich auch bei Abbildung der Datenpfad-Einheiten auf Netzlisten der Vorteil, daß der Chip-Designer die Struktur der Schaltung und die zu verwendenden Standardzellen entsprechend seinem Verständnis der Funktion des Schaltungsteils wählen kann. So kann sich ein günstigeres Ergebnis bezüglich Flächenbedarf oder Laufzeit ergeben, als bei automatischer Optimierung.

Der Datenpfad-Compiler eignet sich nur für Teile einer Schaltung, in denen mehrere Einheiten, die jeweils in gleicher Weise parallel auf allen Bits eines Busses arbeiten, sequentiell miteinander verschaltet werden. Auch wenn eine solche Einheit nur auf einem Teil der Bits arbeiten soll, kann aufgrund der festgelegten Geometrie der Platz für die nicht genutzten Bits bei direkter Umwandlung in einen Hardware-Block nicht eingespart werden.

Innerhalb des Histogrammers treten keine längeren regulären Datenpfade auf. In den meisten Einheiten ist ein großer Anteil von Steuerlogik notwendig. Dort ist eine Anwendung des Datenpfad-Compilers nicht effizient. Die Akkumulatoren des Statistikteils werden jedoch jeweils vierfach instantiiert und rechnen mit sehr breiten Werten (34 bzw. 44 Bit). Da diese Akkumulatoren relativ einfach aufgebaut sind, bietet sich der Versuch einer Optimierung durch Einsatz des Datenpfad-Compilers an. Deshalb wurden in einem alternativen Entwurf des Statistikteils die Akkumulatoren für Grauwertsumme und Quadratsumme mit dem Datenpfad-Compiler noch einmal manuell entworfen. Der Rest des Statistikteils wurde wieder über VHDL implementiert. Diese Realisierung wurde hinsichtlich des Platzaufwandes mit der reinen direkt aus VHDL gewonnenen Standardzell-Realisierung verglichen (s. Kapitel 5.8). Es ergab sich für den gesamten Statistikteil bei Implementation der Akkumulatoren als Hardwareblock eine Einsparung von etwa 25%.

Für den Akkumulator zur Berechnung der Quadratsumme muß eine Datenpfad-Einheit entsprechend der maximalen Breite innerhalb dieses Schaltungsteils ausgelegt werden, also 44 Bit (s. Anhang C). Ein Ripple-Carry-Addierer hat eine Laufzeit von knapp 40 ns für 44 Bit,

ist also zur Aufsummierung der Grauwertquadrate bereits ausreichend schnell. Durch Verwendung des Ripple-Carry-Addierers ist gegenüber dem defaultmäßig vom ASIC-Syntesizer verwendeten Addierertyp bereits eine Einsparung an Gattern möglich. Obwohl die höherwertigen 24 Bit des Addierers nur als Inkrementer genutzt werden, können nur einheitlich Volladdierer für jedes Bit des Datenpfades implementiert werden. Es stehen Flip-Flops mit einem internen Multiplexer zur Verfügung, der es ermöglicht, wahlweise ein Eingangssignal oder den aktuellen Ausgangswert zu übernehmen. Diese Flip-Flops eignen sich für das Akkumulations-Register, da es mit dem Systemtakt getaktet wird, aber nur bei Eintreffen eines Pixeltaktes ein neuer Wert übernommen werden soll.
Der eigentliche Akkumulator wurde aus einem Ripple-Carry-Addierer und Flip-Flops mit vorgeschalteten Multiplexern aufgebaut. Ebenfalls wurde ein Ausgangsmultiplexer mit in die Datenpfad-Einheit aufgenommen, um das 44 Bit breite Register auf den 16 Bit breiten Datenbus zu multiplexen. Dieser Multiplexer wird als 16 Bit breiter Multiplexer mit je einem Eingang für die unteren 16 Bit des Registers, die mittleren 16 Bit und die restlichen 12 Bit ausgelegt. Über spezielle Verdrahtungs-Direktiven des Datenpfad-Compilers, sogenannte Bus-Swizzles, kann die dazu notwendige Aufsplittung der Bits der Summe vorgenommen werden. Ebenso können mit Bus-Swizzles Eingänge auf feste Potentiale gelegt oder Bits als nicht verwendet deklariert werden. Aufgrund der Struktur der Datenpfad-Einheiten wird innerhalb des Blocks auf dem Chip der Platz für einen 44 Bit breiten Multiplexer besetzt, obwohl nur 16 Bit benötigt werden.
Entsprechend wurde der Akkumulator für die Grauwertsumme entworfen.
Die restlichen Funktionen des Statistikteils wurden über eine entsprechend modifizierte VHDL-Komponente implementiert. Diese Komponente gibt zusätzlich alle Signale zur Ansteuerung der Datenpfad-Einheiten aus.

Ein Entwurf des Histogramminkrementers bzw. der 4 und 12 Bit breiten Einzelinkrementer mit dem Datenpfad-Compiler liegt ebenfalls nahe, da diese Inkrementer mehrfach instantiiert werden und eine geringe Laufzeit gefordert ist. Als Datenpfad-Einheit für Inkrementer steht ein Ripple-Carry-Inkrementer oder der vom ASIC-Synthesizer genutzte Addierertyp zur Verfügung. Da der Ripple-Carry-Inkrementer jedoch für den Einsatz im Histogramm-inkrementer zu langsam ist, hätte sich keine Verbesserung gegenüber der Synthese aus VHDL ergeben. Es zeigte sich sogar, daß der ASIC-Synthesizer bei entsprechenden Timing-Constraints Inkrementer erzeugt, die dem manuellen Entwurf entsprechen.

5.7 Einbau von Padzellen

Um die Herstellung des Chips zu ermöglichen, müssen an sämtliche Ein- und Ausgänge des Histogrammers Pad-Zellen angeschlossen werden. Die Pad-Zellen sehen Anschlußpunkte für Bond-Drähtchen vor und enthalten Pegelwandler für Eingänge und Treiberstufen für Ausgänge des Chips.
Um diese Padzellen einzubauen, wurde oberhalb des grafischen Top-Levels eine weitere Ebene erstellt, innerhalb der der Histogrammer mit geeigneten Padzellen verbunden wird (s. Anhang C). Bei der Auswahl der Pad-Zellen standen unterschiedliche Treiberstärken und Pegelwandler für TTL- und CMOS-Logik zur Verfügung. Auf Wunsch des MAZ wurden für alle Signale TTL-Pegel und Treiberstärken im Bereich von 4 bis 8 mA vorgesehen. Für den Datenbus wurden bidirektionale Treiber-Pads (Tristate-Pads) eingesetzt. Diese verfügen über einen gemeinsamen Ein-/ Ausgang zum Pin und getrennte Ein- und Ausgänge zu den Komponenten des Chips. Deshalb war es notwendig, separate Ein- und Ausgänge in der Bus-interface-Einheit vorzusehen. Weiterhin muß die Datenrichtung der Tristate-Pads von der Businterface-Einheit gesteuert werden.

5.8 Ergebnisse der Netzlistensimulation

Die Realisierung aller spezifizierten Funktionen des Histogrammers auf Netzlistenebene ist eine wesentliche Ziellinie dieser Arbeit. Der Funktionsnachweis kann durch Simulation des gesamten Histogrammers geführt werden.

Um eine Simulation des gesamten Histogrammers zu ermöglichen, wurden nach Einbau der Padzellen erneut Netzlisten aller Teile des Histogrammers erstellt. Diese Simulation wurde im VHDL-Simulator mit der für das Verhaltensmodell erstellten Simulationsumgebung (histosti.vhd, s. Anhang D) durchgeführt.
Um Timing-Probleme entdecken zu können und um unterschiedliche Signallaufzeiten deutlich sichtbar zu machen, wurde eine Taktung des Histogrammers mit 50 MHz simuliert. Als Pixeltaktrate wurde 2/5 dieser Frequenz gewählt, also 20 MHz. Mit dieser Frequenzkonstellation konnten Setup- und Hold-Zeitverletzungen in den Eingangs-Flip-Flops vermieden werden, da beide Frequenzen in einem festen Phasenverhältnis zueinander stehen. Bei diesem Taktraten-Verhältnis erkennt der Histogrammer abwechselnd zwei Pixeltakte im minimal zulässigen Abstand von zwei Systemtakten und im Abstand von drei Systemtakten. Auf diese Weise wird sowohl der Fall des maximalen Verhältnisses von Pixeltaktrate zu Systemtaktrate geprüft als auch der Fall, daß eine Verarbeitungszeit von mehr als zwei Takten pro Pixel zur Verfügung steht. Bei der Simulation wurden Ein- und Ausgabesignale des Histogrammers und Signalverläufe innerhalb des Histogrammers aufgezeichnet. Insbesondere die Signale an der Schnittstelle zum Histogrammspeicher zeigen bei Operationen wie der Histogrammerstellung und der Egalisierung sofort, ob die wesentlichen Komponenten des Histogrammers korrekt arbeiten. Die aufgezeichneten Ergebnisse des Simulationslaufes wurden mit Hilfe des VHDL-Simulators als Kurvendiagramm ausgegeben. Die so erhaltenen Signalverläufe wurden kontrolliert und auch mit den Signalverläufen bei der reinen Verhaltenssimulation verglichen.
In Anhang E sind eine Beschreibung des Simulationsablaufs und ein Ausdruck der wesentlichen Passagen des Simulationslaufes beigefügt.

Die Simulation zeigt, daß der Histogrammer alle grundlegenden Funktionen auf der Netzlistenebene spezifikationsgemäß ausführt.

Es sind jedoch weitere unabhängige Prüfungen des Histogrammers erforderlich. Es wurden nicht sämtliche Modi bei allen Speicheraufteilungen durchgetestet. Der Histogrammer verfügt nämlich über eine große Anzahl unterschiedlicher Speicheraufteilungen. Bei jeder dieser Speicheraufteilungen werden die Inkrementer und Addierer innerhalb der ALU in unterschiedlicher Weise zusammengeschaltet. Auch der Multiplexer zur Ankopplung des Speichers an das Hostinterface wird jeweils unterschiedlich konfiguriert.
Damit weitere Tests vom MAZ durchgeführt werden können, wurde die vom MAZ bereits für die Validation des Verhaltensmodells erstellte Testumgebung an die Erfordernisse der Netzlistensimulation adaptiert. Dazu waren geringfügige Anpassungen des Timings der Busschnittstelle notwendig, um Setup- und Hold-Zeitverletzungen zu vermeiden. Die für diese Überprüfung notwendigen Testsequenzen können ebenfalls in späteren Entwicklungsstadien verwendet werden. Darauf aufbauend kann beispielsweise eine Testsequenz für Prototypen des Chips erstellt werden.
Ein Probelauf mit der MAZ-Simulationsumgebung, bei dem Histogramme über 3 Fenster innerhalb eines Bildes mit 4096 Pixeln erstellt und ausgelesen wurden, benötigte auf dem vom MAZ zur Verfügung gestellten SPARC 2-Rechner eine Simulationszeit von ca. 30 Stunden. In Anbetracht dieser langen Simulationszeit sind gezielte Prüfmuster dem Test mit realistischen Bildgrößen vorzuziehen.
Eine Testsequenz, die durch Einbringung von Prüfmustern in den Histogrammspeicher Übertrags- und Überlaufbehandlung der ALU bei Histogrammerstellung prüft, wurde

beispielhaft für die Simulationsumgebung des MAZ erstellt und simuliert. Diese Testsequenz ist in Anhang F abgedruckt. Sie zeigt, daß der Histogrammer in dem getesteten Modus spezifikationsgemäß arbeitet.
Unter Einsatz des IKOS-Hardwarebeschleunigers zum Voyager-VHDL-Simulator sollte auch ein Probelauf mit realistischen Bildgrößen bei angemessener Laufzeit möglich sein.

5.9 Vorläufige Layout-Ergebnisse

Anhand der Netzliste kann innerhalb des Compass-Systems bereits eine Flächenabschätzung für die einzelnen Einheiten des Histogrammers oder den gesamten Histogrammer vorgenommen werden. Dazu plaziert das System die verwendeten Standardzellen und sonstigen Einheiten und ermittelt den für die Verdrahtung notwendigen Platz. Tabelle 5.3 zeigt die Anzahl der benötigten Standardzellen und den Flächenbedarf der einzelnen Einheiten.
Als weitere Information ist die Anzahl der Gatteräquivalente angegeben. Ein Gatteräquivalent entspricht bei Compass dem Aufwand für ein NOR- oder NAND-Gatter mit zwei Eingängen und minimaler Treiberstärke. Diese Gatter bestehen aus 4 Transistoren. Auch ein Inverter doppelter Treiberstärke wird mit 4 Gatteräquivalenten angegeben, da er zwei Transistoren doppelter Größe enthält. Die Anzahl der Gatteräquivalente ist wichtig für die Berechnung der dynamischen Stromaufnahme des Chips. Dies ist die Stromaufnahme, die durch das Umladen von Kapazitäten entsteht. Aus diesem Grund entspricht die Anzahl der Gatteräquivalente der RAMs nicht der Anzahl, die sich aus einer Berechnung der Transistoranzahl ergeben würde. Die RAMs haben insgesamt 49152 Speicherzellen, von denen jede über 6 Transistoren verfügt. Der Histogrammer enthält folglich insgesamt etwa 450 000 Transistoren, davon 140 000 in Standardzellen.

Tabelle 5.3: Syntheseergebnisse und vorläufige Layoutergebnisse

Komponente (Filename)	Standardzellen, (Gatteräquivalente)	Flächenbedarf [mm²]
ALU (histadd)	791 (2158)	2,60
RAM-Adreßmultiplexer (histadr)	63 (179)	0,24
Businterface (histbus)	93 (298)	0,45
Steuereinheit (histctrl)	114 (265)	0,38
Adreßdecoder (histdecd)	207 (403)	0,59
Pixeleingang (histea)	104 (471)	0,40
Intervallkomparator (histintv)	67 (302)	0,40
Pixelausgang (histpout)	95 (286)	0,37
Ram-Bus-Multiplexer (histrmux)	525 (1272)	1,70
Steuerregister (histsfr)	37 (352)	0,53
Statistikteil a) nur VHDL (histstat)	7136 (13483)	19,60
b) mit Datenpfadeinh., Blöcke (statteil2)	3623 (?)	15,60
c) mit Datenpfadeinh., Standardzelloption	4875 (?)	17,30
Bildzähler, Fensterkomp. (histwin)	801 (3984)	5,30
Speicherlatches (hhstmem)	60 (265)	0,41
Speicher (4*RAM1024x12)	4 (15390)	23,00
Insgesamt (Statistikteil c), Layoutfläche)	8000 (35000)	ca. 70

5.10 Fehler der Entwurfssysteme

Chip-Entwurfssysteme sind sehr große Programme. Insbesondere die Synthese von Chips aus VHDL-Beschreibungen wird erst seit wenigen Jahren unterstützt. Synthese und Optimierung einer Standardzell-Logik sind mit zunehmender Gatteranzahl schnell aufwendiger werdende Aufgaben. Chip-Entwurfssysteme unterliegen deshalb einer ständigen Anpassung an wachsende Komplexität der Designs, wachsende Rechenleistungen der verwendeten Rechner und Anforderungen an Standardisierung, um die Datenübergabe zwischen unterschiedlichen Entwurfssystemen und Komponenten von Entwurfssystemen zu ermöglichen. Deshalb kommt es auch immer wieder zu Fehlern in den Systemen oder im Zusammenspiel der Werkzeuge unterschiedlicher Hersteller. Weiterhin ist die Dokumentation oft unzureichend und in Details ungenau. Gründe hierfür bestehen teilweise darin, daß der Hersteller Interna wie beispielsweise die verwendeten Algorithmen nicht offenlegen möchte.
Für den Benutzer sind die Ursachen für Probleme nicht immer klar erkennbar. Es kann ein Benutzerfehler oder ein Fehler im Entwurfssystem vorliegen. Daher muß oft ein Weg gefunden werden, um bestimmte Situationen zu umgehen.

5.10.1 Compass-System

Jede VHDL-Komponente innerhalb des grafischen Top-Levels muß über ein separates VHDL-File definiert werden. Innerhalb jedes Files können jedoch mehrere hierarchisch miteinander verknüpfte Subkomponenten (*Entities*) enthalten sein. Bei Übernahme eines solchen Files in den grafischen Top-Level soll das Compass-System die in der Hierarchie an oberster Stelle stehende Subkomponente automatisch erkennen, damit das Icon für die Grafik mit den richtigen Anschlüssen versehen wird. Dennoch wurden vom Compass-System teilweise die falschen Subkomponenten der Files in den Top-Level übernommen. Eine manuelle Selektion schlug ebenfalls fehl. Nur durch Probieren wurde herausgefunden, daß Abhilfe durch Herstellung von Namensgleichheit zwischen der obersten Subkomponente und dem Filenamen möglich ist.

Der zum Compass-System gehörige RAM-Generator kann laut Handbuch sowohl RAMs mit kombiniertem Schreib- und Leseport als auch solche mit getrenntem Schreib- und Leseport in unterschiedlichen maximalen Konfigurationen generieren. Die in der Dokumentation vorgesehenen RAM-Größen konnten bei RAMs mit getrennten Ein- und Ausgängen jedoch nicht erreicht werden. So war eine Implementation des 1024*48-Speichers nur in vier Blöcken mit je 1024*12 Bit anstatt von zwei Blöcken mit je 1024*24 Bit möglich. Eine Anfrage bei Compass ergab, daß es sich hierbei um einen Fehler des RAM-Generators handelt.

Weitere mit dem VHDL-Quellcode zusammenhängende Probleme bei Compass:

1. Benutzung von Tristate-Bussen

Die Verwendung eines Datenbusses, der von mehreren Stellen her getrieben und gelesen werden kann, wird durch VHDL unterstützt. In der *Entity*-Deklaration von Komponenten, die auf den Bus schreibend und lesend zugreifen, muß der Bus als *inout* deklariert werden, während bei Einheiten, die nur schreibend oder nur lesend auf den Bus zugreifen, auch eine Deklaration als *out* bzw. *in* hinreichend ist. Bei der Erzeugung von VHDL-Code aus der synthetisierten Netzliste heraus beachtet das Compass-System diese drei Möglichkeiten für jede einzelne Leitung des Busses getrennt und erzeugt fehlerhaften VHDL-Code für Einheiten, die sowohl lesend als auch schreibend auf den Bus zugreifen, dabei aber nicht alle Leitungen

des Busses lesen oder nicht alle Leitungen des Busses treiben. Die unterschiedlich behandelten Leitungen des Busses werden separat deklariert, jedoch unter dem selben Namen (vgl. folgendes Beispiel). Dies führt bei Analyse durch den VHDL-Simulator zu einer Fehlermeldung wegen Mehrfach-Deklaration.

Beispiel:
Original VHDL-Code:

```
entity BusTest is
  port(Clk,Write,Read: in std_logic;
       Datenbus: inout std_logic_vector (7 downto 0);
      );
end BusTest;

architecture CompassTest of BusTest is
  signal Register: std_logic_vector (3 downto 0);
begin
  DatenBus<="00000000" when Read='1' else "ZZZZZZZZ";

  Lesen:Process(Write,DatenBus(3 downto 0))
  begin
    if Write='1' then
      Register<=DatenBus(3 downto 0);
    end if;
  end Process;
end BusTest;
```

Nach Synthese von Compass aus der Netzliste generierte VHDL-Entity (Ausschnitt):

```
entity BusTest is
  port(Clk,Write,Read: in std_logic ...;
       Datenbus: out std_logic_vector (7 downto 4) ...;
       Datenbus: inout std_logic_vector (3 downto 0) ...; -- Fehler !!!
      );
end BusTest;
```

2. Durchreichen von Signalen innerhalb eines Modules

Manchmal ist es aus Gründen der Übersicht sinnvoll, einen Bus in mehrere Teile aufzusplitten. Findet diese Aufteilung innerhalb einer Einheit statt, die die Bussignale nicht erzeugt, aber die abgespaltenen Teile an andere Einheiten weiterreicht, ist innerhalb der Einheit keine Logik zum Abspalten des Signals notwendig. Das Compass-System synthestisiert diesen Fall zwar korrekt, erzeugt jedoch wiederum falschen VHDL-Code, indem auch die durchgereichten Signale als *in* anstatt von *out* deklariert werden.
Das Synopsys-System meldet in diesem Fall bei der Synthese die Erzeugung eines „Feed through“, erzeugt jedoch korrekten VHDL-Code aus der Netzliste.

5.10.2 Synopsys-System

Mit Synopsys ergab sich bei der Arbeit mit *std_logic*-Datentypen das Problem, daß der Vergleich zwischen zwei Bit-Vektoren gleicher Länge doppelt definiert wird, nämlich einmal im Standard-Sprachumfang und einmal in der Zusatzlibrary „std_logic_arithmetic“. Dieses Problem wird nicht korrekt gelöst und führt bei der Syntaxprüfung mit dem Programm VHDLAN zur Meldung einer Mehrdeutigkeit. Um die Fehlermeldung unter Benutzung der gewünschten Funktion zu umgehen, muß die Vergleichsoperation in der Präfix-Form

```
std_logic_arithmetic.=(Argument1,Argument2)
```

aufgerufen werden. Eine weitere Möglichkeit ist die Umwandlung eines der Argumente in eine Integer-Variable, da der Vergleich zwischen einem Bit-String und einer Integer-Variable im Standard-Sprachumfang nicht definiert ist. Bei Verwendung vieler Vergleichsoperationen machen jedoch beide Lösungen den Quelltext unübersichtlich. Um das Problem zumindest für das Verhaltensmodell zu umgehen, wurden veränderte Versionen der Zusatzlibraries angelegt, in denen die mehrdeutigen Funktionen auskommentiert sind. Es zeigte sich jedoch, daß Synopsys mit diesen veränderten Libraries nicht synthetisieren kann.

Eine Synthese des gesamten Histogrammers wurde mit Synopsys nicht vorgenommen.

6 Ausblick

Kapitel 5 zeigt, wie ein Chip mit VHDL entwickelt werden kann und welche Besonderheiten bei der Realisierung des Histogrammers mit dem Compass-Entwurfssystem auftraten. Der Histogrammer wurde zur Validation seiner Funktion erfolgreich auf Gatterebene simuliert und es wurden bereits Padzellen eingefügt. Probeweise Layoutversuche ergaben, daß der Gesamt-Flächenbedarf des Histogrammers bei etwa 70 mm² liegt. Dies ist eine für ein ASIC noch akzeptable Größe.
Bevor der Histogrammer zur Fertigung an einen ASIC-Hersteller übergegeben werden kann, muß ein Testkonzept für den Funktionstest bei der Fertigung erarbeitet und ein Layout des Chips erstellt werden.

6.1 Fertigstellung des Histogrammers

6.1.1 Testbarkeit

Nachdem eine integrierte Schaltung entworfen und der Entwurf für fehlerfrei befunden wurde, kann sie gefertigt werden. Bei der Produktion integrierter Schaltungen wird jedoch ein großer Anteil von fehlerhaften Exemplaren hergestellt (bei komplexen Schaltungen oft über 60 Prozent). Ein Großteil der dabei auftretenden Fehler resultiert aus zufällig auf dem Wafer oder einer der Masken verteilten Staubpartikeln oder anderen Fehlerquellen, die zu Unterbrechung von Leiterbahnen oder Kurzschlüssen zwischen Leiterbahnen führen können. Aufgrund der geringen Abmessungen und der großen Anzahl von Prozeßschritten bei der Herstellung eines Chips können Fehler nur selten optisch erkannt werden. Deshalb muß der Chip elektrisch getestet werden, um Fehler zu detektieren. Bei komplexen Schaltungen ist es jedoch unmöglich, sämtliche im Betrieb auftretenden Situationen zu erproben. Deshalb muß versucht werden, die Funktion sämtlicher Gatter innerhalb des Chips zu testen. Es gibt jedoch nur eine geringe Anzahl von Punkten des Chips, auf die von außen elektrisch zugegriffen werden kann. Die interne Funktionalität zwischen Ein- und Ausgängen des Chips ist meist so kompliziert, daß es unmöglich ist, einzelne Gatter zu testen. Daher muß ein Testkonzept erarbeitet werden, über das durch den Test längerer Signalpfade innerhalb des Chips zugleich bzw. fortschreitend eine große Anzahl von Gattern einer kombinatorischen Logik getestet wird.
Eine häufig ausreichende Annahme für die Erstellung von Testmustern ist, daß innerhalb jeder Kette von Gattern nur eine Leitung fehlerhaft einen Kurzschluß gegen die positive oder die negative Versogungsspannung hat bzw. unterbrochen ist (Stuck at-Fehlermodell).
Um eine Anzahl von Gattern innerhalb einer Einheit ausnahmslos testen zu können, muß eine Anzahl von Prüfmustern erstellt werden, bei denen die Ausgabe der Einheit mindestens zweimal von dem Pegel jedes Gatterein- und ausganges abhängt, nämlich einmal bei positivem Pegel und einmal bei negativem Pegel. Voraussetzung für diesen Test einer kombinatorischen Logik ist, daß Ein- und Ausgänge der Gatter gut mit neuen Testmustern zu laden bzw. auszulesen sind. Um alle Ein- und Ausgänge kombinatorischer Logik im Chip erreichen zu können, werden häufig sämtliche Flip-Flops innerhalb des Chips in einem speziellen Testmodus zu einem Schieberegister (Scan-Path) zusammengeschaltet. Die dazu notwendige Logik muß in den Chip integriert werden. Eingang und Ausgang des Scan-Paths sowie Steuersignale für den Testmodus müssen über zusätzliche Anschlüsse zugängig gemacht werden. Der Scan-Path ermöglicht auch den Test aller darin zusammengeschalteten Flip-Flops. Über den Scan-Path kann jeweils ein Testmuster für jede kombinatorische Logik in den Chip geladen werden.

Aufgrund dieses Testmusters generieren sämtliche kombinatorischen Logikeinheiten Ausgangssignale. Diese Ausgangssignale werden dann in die Scan-Path-Flip-Flops getaktet und ausgelesen, während ein neues Testmuster in den Scan-Path geschoben wird.

Ein Chip, der kommerziell gefertigt und verkauft werden soll, muß vor der Weitergabe an Kunden so intensiv getestet werden, daß die restliche Fehlerwahrscheinlichkeit sehr gering ist. Eine Restfehlerwahrscheinlichkeit von deutlich unterhalb einem Prozent ist anzustreben. Dazu sollte der Prüfmustersatz eine Fehlerüberdeckung von deutlich über 95 Prozent erzielen. Das bedeutet, daß mindestens 95 Prozent der innerhalb des Fehlermodells möglichen Fehlersituatuionen durch den Test erkannt werden müssen. Der dazu notwendige Test soll nach Möglichkeit schnell, d.h. mit möglichst wenigen Prüfmustern durchführbar sein, um die Testkosten je Exemplar gering zu halten.

Für alle Komponenten des Histogrammers sollte ein eigenes Testkonzept erarbeitet werden, um dem modularen Aufbau gerecht zu werden.

Das Testen von Speichern kann anhand von Informationen über den internen Aufbau der Speicherarrays mit einer Anzahl von wenigen Prüfmustern je Speicherstelle vorgenommen werden (üblich sind ca. 2-20 Prüfmuster je Speicherstelle). Der Speicher des Histogrammers kann über das Hostinterface beschrieben und ausgelesen werden, so daß ein Speichertest über diese Schnittstelle einfach möglich ist. Der Einbau eines automatischen Testvektorgenerators, der von den meisten ASIC-Tools angeboten wird, empfiehlt sich beim Histogrammer nicht, da die dann notwendige zusätzliche Logik die Geschwindigkeit jedes Speicherzugriffs verringern würde.

Durch Einbringung von ausgewählten Prüfmustern in den Speicher und anschließende Erstellung eines Histogramms eines Testbildes kann der Histogramminkrementer getestet werden. Das Testbild muß einen Grauwert für jede mit einem Prüfmuster geladene Speicherstelle enthalten. Der Addierer der ALU, der zur Bildung der Summenhäufigkeiten und der Kachelsummen dient, kann durch Erstellung der Summenhäufigkeiten über zuvor in den Speicher geladene Testmuster geprüft werden.

Bildzähler und Fensterkomparatoren können getestet werden, indem je ein Bild des Formats 4096*1 und ein Bild des Formats 1*4096 einprogrammiert und eingespeist werden. Währenddessen können die Fensterpositionen mehrfach geeignet umprogrammiert werden und die Vergleichsergebnisse der Fensterkomparatoren an den entsprechenden Ausgangsleitungen (WinActive) überprüft werden.

Zum Test des Statistikteils kann ein Scan-Path durch sämtliche Statistikregister gelegt werden. Wahlweise könnte eine Funktion implementiert werden, über die alle Statistikeinheiten in zwei separate Hälften aufgespalten werden, die dann in insgesamt 4096 (=2^12) Schritten (anstatt von $4096^2 = 16\,777\,216$ Schritten) ausgehend von Null bis zum Maximalwert gezählt werden können. Die obere Hälfte jeder Einheit besteht dann aus einem einfachen 12 Bit-Zähler. Die untere Hälfte besteht bei den Zählern für die Schwellensummen ebenfalls aus einem 12 Bit-Zähler und bei den Summen-Registern aus je einem 10 bzw. 20 Bit breiten Akkumulator und Addierer. Da die Summen und Zählerstände jederzeit auslesbar sind, könnten alle Statistikregister und die vorgeschalteten Recheneinheiten parallel mit einigen tausend Schritten getestet werden. Eine weitere Möglichkeit, um den Statistikteil testbar zu machen, wäre es, ein Beschreiben der Statistikregister über den Datenbus vorzusehen. Darüber könnten ausgewählte Testvektoren geladen werden.

Durch Implementation zusätzlicher Stellungen des Multiplexers für den Pixelausgang können beliebige weitere Signale innerhalb des Histogrammers zum Test zugängig gemacht werden. Im Histogrammer sind viele Einheiten vierfach vorhanden. Dies könnte genutzt werden, um bei gleicher Programmierung aller vier Fenster intern Signale sämtlicher dann redundanten Einheiten zu vergleichen und das Vergleichsergebnis verfügbar zu machen. Damit könnte die Anzahl der Kontroll-Lesezugriffe deutlich reduziert werden.

Aufgrund dieser Betrachtung der Testmöglichkeiten aller wesentlichen Einheiten des Histogrammers scheint nur eine geringe Anzahl von Erweiterungen notwendig zu sein, um sämtliche Funktionen ausreichend gut testbar zu machen.

6.1.2 Abschließende Synthese, Layout und Fertigung

Wenn nach Hinzufügung von Testlogik der VHDL-Code komplett fertiggestellt ist, kann die abschließende Synthese erfolgen. Dabei sollte maximale Optimierung des Synthesizers angewählt werden, um das bestmögliche Ergebnis bezüglich Laufzeit und Gatteraufwand zu erzielen. Ein globaler Optimierungslauf ist notwendig, um die Treiberstärken der einzelnen Einheiten an die jeweiligen Lastkapazitäten innerhalb des Chips anzupassen. In diesem Schritt muß die angestrebte maximale Taktfrequenz des Histogrammers als globales Constraint vorgegeben werden, damit die Signalpfade zwischen den Einheiten entsprechend optimiert werden. Dabei sind die Recheneinheiten des Statistikteils als „don't touch“ zu deklarieren, da hier weniger strenge Constraints ausreichend sind.
Anschließend muß die Einhaltung sämtlicher Timing-Constraints überprüft werden und sichergestellt werden, daß das Timing im Zusammenspiel der Einheiten eingehalten wird. Dazu sind die Laufzeiten zwischen sämtlichen Flip-Flops unter Beachtung der Polarität der Taktung zu überprüfen. So darf beispielsweise die Verzögerung einer Logik zwischen einem mit negativer Flanke getakteten Flip-Flop und einem mit positiver Flanke getakteten Flip-Flop nur ca. 40 Prozent der Taktdauer betragen, wenn eine Abweichung des Takt-Tastverhältnisses vom idealen Tastverhältnis (50 Prozent) um 10 Prozent zulässig sein soll. Zusätzlich ist die maximale Verschiebung des Taktes zwischen den Flip-Flops zu berücksichtigen (Clock-Skew). Dies gilt ebenfalls für die Setup-Zeit und bei Synchronisations-Flip-Flops die Dauer metastabiler Zustände (s. Kapitel 5.5.4).
Dieser Test kann auch mit spezieller Software automatisch durchgeführt werden.

Bei Anfertigung des synthetisierbaren VHDL-Codes wurde versucht, manuelle Eingriffe in die Netzliste überflüssig zu machen. Es war jedoch nicht möglich, dem Synthesizer anzuzeigen, daß an bestimmten Stellen Synchronisations-Flip-Flops verwendet werden müssen. Deshalb müssen die betroffenen Flip-Flops manuell in der Netzliste gegen Synchronisations-Flip-Flops ausgetauscht werden. Diese Veränderungen in der Netzliste müssen nach der abschließenden Synthese vorgenommen werden.
Betroffen sind die zwei Eingangs-Flip-Flops für die asynchronen Signale „Read“ und „Write“ im Businterface und die Synchronisations-Flip-Flops für die optionell ebenfalls zum Takt des Histogrammers asynchronen Signale „Start“ und „PClkIn“. Diese Flip-Flops sind im VHDL-Quellcode mit einem entsprechenden Kommentar versehen. Da alle anderen (asynchronen) Eingangssignale wie Datenbus und Pixeleingang nur aufgrund der Pegel dieser vier synchronisierten Signale ausgewertet werden, müssen keine weiteren Synchronisations-Flip-Flops eingesetzt werden.

Nach der abschließenden Synthese kann ein Layout des Chips angefertigt werden. Auch nach der automatischen Anfertigung eines Layouts und gegebenenfalls manueller Optimierung muß ein eingehender Test sämtlicher Laufzeiten erfolgen, um sicherzustellen, daß alle Signalpfade den Laufzeitanforderungen weiterhin genügen. Erst anhand der hier erzielbaren Laufzeiten kann die erreichbare Taktfrequenz des Histogrammers endgültig abgeschätzt werden.
Nach Validation dieses Layouts wird der Chip mit dem erarbeiteten Satz von Testmustern simuliert. Die Ausgabe des Histogrammers wird als Dokumentation zusammen mit den Stimuli in einer Datei abgespeichert. Diese Simulation muß mit dem im Compass-System integrierten Simulator (QSIM) erfolgen, damit der Chip-Hersteller garantiert, daß das Verhalten der gefertigten Chips mit dem im Simulationslauf dokumentierten Verhalten übereinstimmt.

Die Layoutdaten können dann zusammen mit den Prüfdaten zur Fertigung einiger weniger Testexemplare des Histogrammers an den Chiphersteller übergeben werden.
Erst nach eingehender Prüfung der Funktion der Testexemplare und ggf. Einbringung von Verbesserungen wird im allgemeinen eine größere Anzahl von Probeexemplaren hergestellt und an Anwender des Histogrammers weitergegeben. Nach Evaluation durch die Anwender kann eine Fertigung in Stückzahlen beginnen.

6.2 Mögliche Erweiterungen und Verbesserungen des Histogrammers

Beim praktischen Einsatz eines Chips ergibt sich oft eine Reihe von Wünschen nach zusätzlichen oder veränderten Funktionen. Es kann sich auch herausstellen, daß bestimmte Funktionen oder Modi nur selten genutzt werden und in zukünftigen Versionen entfallen können.
Während solche Erfahrungen erst beim Einsatz des Histogrammers gemacht werden, ist eine Reihe allgemeiner Trends und Anforderungen an zukünftige Versionen des Histogrammers absehbar.
In der digitalen Bildverarbeitung besteht der Trend zu höheren Auflösungen und parallel dazu zu höheren Pixelfrequenzen. Weiterhin ist mit einer zunehmenden Bedeutung von Farbbildverarbeitung zu rechnen, um neue Einsatzgebiete für die Bildverarbeitung zu erschließen.

Erhöhung der zulässigen Pixelfrequenz

Der Histogrammer unterstützt mit bis zu 4096*4096 Pixeln bereits sehr hohe Auflösungen. Um bei so hohen Auflösungen jedoch für Echtzeitverarbeitung ausreichende Bildfrequenzen zu erreichen, sind höhere Pixelfrequenzen erforderlich als der Histogrammer unterstützt. Die maximale Pixelfrequenz des Histogrammers kann jedoch durch Übergang auf eine schnellere Technologie (z.B. 0,6 µm-Strukturen statt 0,8 µm) heraufgesetzt werden. Eine weitere deutliche Steigerung der Pixelfrequenz ist durch Verwendung eines Dual-Port-RAMs anstatt des Single-Port-RAMs möglich (vgl. Kapitel 4.3.1).

Erhöhung der Auflösungen

Um die maximale zulässige Pixelanzahl des Histogrammers je Bild zu vergrößern, müssen sämtliche Registerbreiten erhöht werden. Eine Auflösung von maximal 8192*8192 Pixeln würde eine Verbreiterung der Positionsregister um ein Bit auf 13 Bit und eine Verbreiterung der Statistikregister um je zwei Bit erfordern.
Mit einer Grauwertauflösung von zehn Bit bietet der Histogrammer auf längere Sicht eine genügend hohe Grauwertauflösung, da es kaum Anwendungen gibt, die eine derart hohe

Grauwertauflösung bei gleichzeitig großer Pixelanzahl erfordern. Auch das menschliche Auge kann nur deutlich weniger als 1024 Helligkeitswerte unterscheiden. Die Aufzeichnung eines Histogramms über ein Vollbild mit 11 Bit Grauwertauflösung ist jedoch bereits möglich. Dazu muß externe Fenstersteuerung angewählt werden und das MSB des Grauwertes als Enable-Signal für je eines von zwei Fenstern A und B verwendet werden. Die Pixel mit den Grauwerten 0-1023 werden dann in Fenster A aufgezeichnet, die Pixel mit den Grauwerten 1024-2047 in Fenster B.
Der für Histogramme benötigte Speicherplatz ist der Grauwertauflösung proportional. Eine Erhöhung der Grauwertauflösung erfordert eine entsprechende Vergrößerung des RAM-Adreßbereiches.
Eine auf Wunsch des MAZ angefertigte detaillierte Aufstellung der notwendigen Veränderungen bei Erhöhung von Grauwert- oder Positionsauflösung ist in Anhang G aufgeführt.

Verbesserte Nutzung des vorhandenen Histogrammspeichers

Da für die meisten Anwendungen Grauwertauflösungen von acht Bit ausreichend sind, liegt eine bessere Nutzung des vorhandenen Histogrammspeichers bei geringerer Grauwert-auflösung nahe. Beispielsweise könnte bereits bei einer Grauwertauflösung von neun Bit für alle vier Fenster ein 24 Bit breiter Speicherbereich zur Verfügung gestellt werden. Bei dieser Konfiguration entfallen die Einschränkungen der Fenstergrößen (s. Kapitel 3.2.2).
Um bei 10 Bit Grauwertauflösung ebenfalls die Histogrammerstellung über vier beliebig große Fenster zuzulassen, müßte der Speicherplatz auf 1024*96 Bit verdoppelt werden. In diesem Fall würden sich ALU (Histogramminkrementer) und der Multiplexer zur Ankopplung des RAMs an das Hostinterface deutlich vereinfachen lassen, da auf die große Anzahl unterschiedlicher Speicheraufteilungen verzichtet werden könnte. Der Anteil des Speichers an der gesamten Chipfläche würde dann etwa 40 Prozent betragen.

Erstellung von Histogrammen über Farbbilder

Die Verarbeitung von RGB-Farbbildern ist möglich, indem drei Histogrammer verwendet werden. Dabei zeichnet jeder Histogrammer Histogramme über je einen Farbkanal auf.
Unter Einschränkung der Anzahl separater Fenster pro Farbbild kann jedoch auch die Möglichkeit geschaffen werden, mit nur einem Histogrammer parallel über jeden der drei Farbkanäle je ein Histogramm zu erstellen. Dazu muß eine der oben genannten Speicher-aufteilungen realisiert werden, bei der für mindestens drei Fenster ein je 24 Bit breiter Speicherbereich zur Verfügung steht. Durch separate Adressierung der Speicherblöcke für drei Fenster über je einen Pixeleingang können dann Histogramme über RGB-Farbbilder aufgezeichnet werden.
Eine interne Adressierbarkeit von mindestens zwei Speicherhälften über separate Adressen würde es weiterhin auf einfache Weise erlauben, zugleich Histogramme aufzuzeichnen und Kachelsummen zu bilden. Wahlweise wäre auch ein Zugriff über das Hostinterface auf eine Speicherhälfte möglich, während in der jeweils anderen Speicherhälfte ein Histogramm erstellt wird.

Verbesserte Unterstützung der Erstellung von Cooccurence-Matrizen

Der Histogrammer sieht intern keine gesonderte Unterstützung für die Erstellung von Cooccurence-Matrizen vor. Um eine Cooccurence-Matrix zu erstellen, muß die Nachbarschaftsrelation zwischen jeweils zwei Pixeln (mit einer Grauwertauflösung von maximal 5 Bit) über externe Hardware hergestellt werden. Die Herstellung einfacher Nachbar-schaftsrelationen könnte auch innerhalb des Histogrammers vorgesehen werden. Dazu muß ein

als Ringpuffer mit programmierbarer Länge geschalteter Zeilenspeicher mit 4097*5 Bit in den Histogrammer integriert werden. Dieser Ringpuffer muß so geschaltet werden, daß die oberen (oder unteren) fünf Bit des 10 Bit breiten Pixeleinganges durch diesen Ringpuffer um Null bis zu 4097 Takte verzögert werden können.

Wie hier gezeigt wurde, erfordern viele der möglichen Verbesserungen des Histogrammers eine Vergrößerung der internen Speicherkapazität. Da sich bei Realisierung der aktuellen Version des Histogrammers gezeigt hat, daß der Speicher einen Anteil von nur etwa 30 Prozent an der Chipfläche von 70 mm² hat, wäre die Vergrößerung des Chips auch bei einer Verdopplung der Speicherkapazität noch in Kauf zu nehmen. Dies gilt insbesondere dann, wenn eine Nachfolgeversion des Chips mit feineren Strukturgrößen gefertigt wird.

7 Danksagung

Meinen ganz besonderen Dank möchte ich meinem Betreuer Herrn Dr. Reinhard Rauscher vom Fachbereich Informatik der Universität Hamburg, Arbeitsbereich Technische Grundlagen der Informatik, aussprechen. Er stellte den Kontakt zum MAZ her, der zum Diplomarbeitsthema führte. Darüber hinaus war er ein stets ansprechbarer Ratgeber, der nicht nur die Zusammenarbeit mit dem MAZ koordinierte, sondern auch bei Interessenkonflikten, wie sie bei industriellen Aufgabestellungen auftreten können, engagiert blieb.
Dem MAZ möchte ich für die interessante Aufgabe im Rahmen eines realen Projektes, die Bereitstellung eines Arbeitsplatzes und der notwendigen technischen Hilfsmittel danken.
Für die Einführung in VHDL und die Hilfestellung bei der Verwendung des Synopsys-Systems, mit dem die erste Version des Verhaltensmodells erstellt wurde, möchte ich Herrn Andreas Mäder und den übrigen Assistenten des Fachbereich TECH danken.
Für die Hilfestellung bei der Benutzung des Compass-Entwurfssystems möchte ich den Herren Müller und Troike vom MAZ danken.
Der Diplomandin Birgit Schulte möchte ich für die gute Zusammenarbeit bei der Inbetriebnahme der vom MAZ erstellten Simulationsumgebung danken.

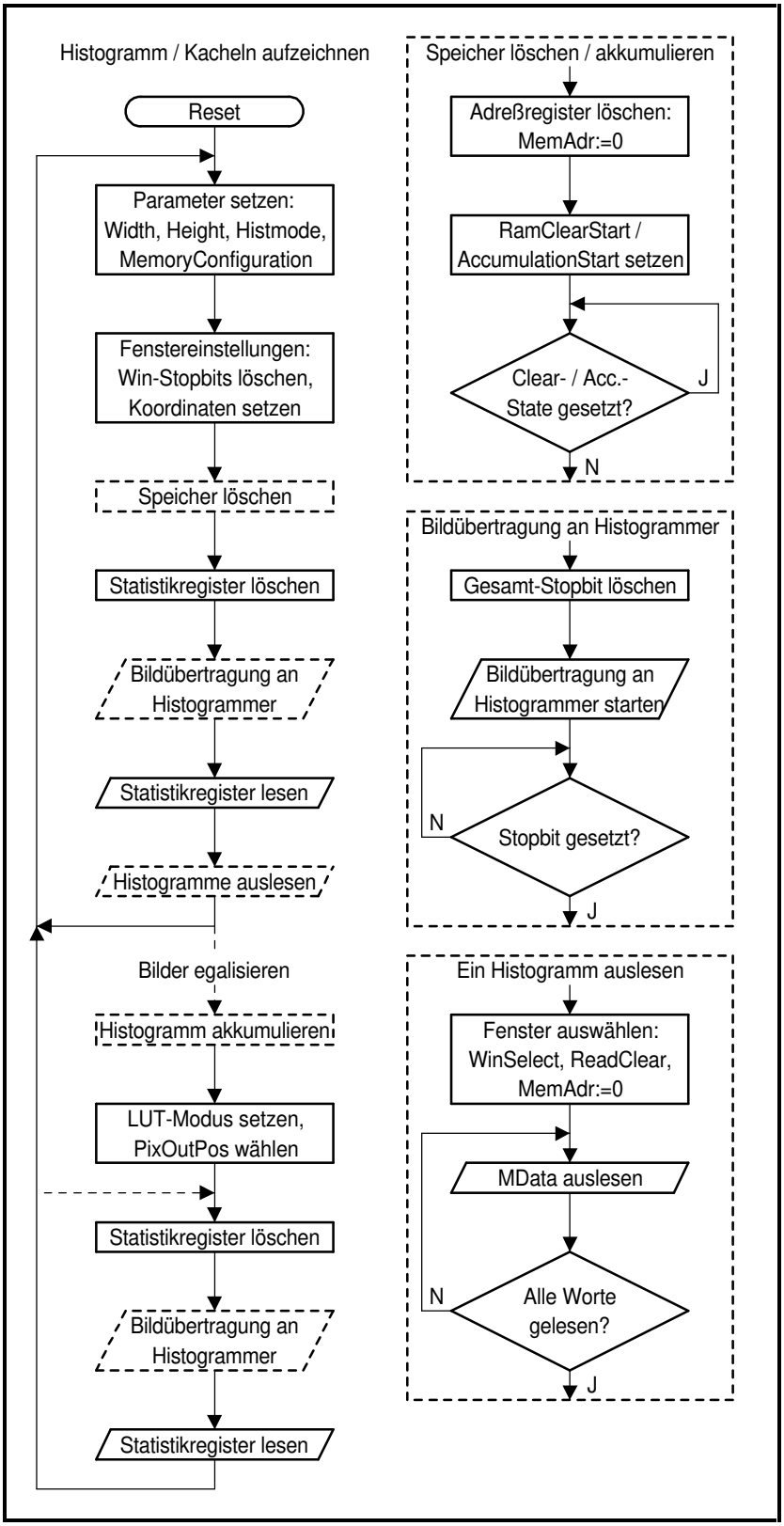
8 Literaturverzeichnis

[1] Ashenden, P.J.: The VHDL Cookbook. University of Adelaide, South Australia 1990
[2] Ballard, D.H., Brown, C.M.: Computer vision. Englewood Cliffs, Prentice-Hall 1982
[3] Bässmann, H., Besslich, Ph.W.: Bildverarbeitung Ad Oculos. (Programm Ad Oculos und Begleitbuch), Springer-Verlag 1991
[4] Compass: 0.8-Micron CMOSVCC4RA3 Cell Compiler - Rev.B. Compass 1994
[5] Compass: ASIC Synthesizer - V8R4.6, Compass 1994
[6] Dreschler-Fischer, L.: Skript Bildverstehen. Universität Hamburg, SoSe 1994
[7] ES2: ECPD10 Library Databook. ES2 1991
[8] Haberäcker, P.: Digitale Bildverarbeitung. Hanser 1989
[9] Haralick, R.M.: Statistical image texture analysis. In: T.Y. Young, K.S. Fu: Handbook of pattern recognition and image processing. Academic Press 1986
[10] Harris: Datenblatt zum HSP48410 Histogrammer/Accumulating Buffer, Harris 1992
[11] IEEE: Standard VHDL Language Reference Manual (IEEE Std 1076-1987). IEEE 1988
[12] Intel: 486 DX Microprocessor Data Book. Intel 1991
[13] Intel: MCS®-96 809XBH/839XBH Advanced 16-Bit Microcontroller with 8- or 16-Bit external Bus, Data Sheet. Intel 1987
[14] Jähne, B.: Digitale Bildverarbeitung. Springer-Verlag 1991
[15] Kittler, J. ,Duff, M.: Image Processing System Architectures. Research Studies Press Ltd. 1985
[16] Koch, J.: Der 16bit-Mikroprozessor SC 68000 - Eigenschaften. Valvo 1982
[17] Lattice: Kapitel „GAL Metastability Report“ in: GAL Data Book. Lattice 1990
[18] Legler, W.: Kapitel „Rückkopplung und Stabilität“ in: Vorlesungsmanuskript Physik / Elektronik I. Universität Hamburg 1987
[19] Limann, O.: Fernsehtechnik ohne Ballast. 9. Auflage, Franzis 1971
[20] LOSP: Beschreibung zur KAMA-10 CCD-Matrixkamera, LOSP GmbH 1994
[21] LSI Logic: Datenblatt zum L64250 Histogram/Hough Transform Processor, LSI Logic 1991
[22] Micron: SRAM 1993 Data Book. Micron Semiconductor, Inc. 1993
[23] Müller, St.: Arbeitpapier „Mögliche Einsatzgebiete eines Histogramm-Prozessors“. MAZ Hamburg GmbH, 10.1.1994
[24] Müller, St.: Arbeitspapier „Konzept eines Histogramm-Prozessors“. MAZ Hamburg GmbH, 31.3.1994
[25] Müller, St.: Arbeitspapier „Konzept eines Histogramm-Prozessors Kap. IV Ein-/Ausgabe von Daten, Pixelstrom“. MAZ Hamburg GmbH, 27.7.1994
[26] PCI SIG: PCI Local Bus Specification, Production Version. Rev. 2.0 April 30, 1993. PCI SIG 1993
[27] Seiko Epson: Quarzprodukte von Seiko Epson - The Crystalmaster’93. Spezial-Electronic KG 1993
[28] Shaoo, P.K. , Soltani, S., Wong, A.K.C.: A survey of thresholding techniques. Computer Vision Graphics and Image Processing 41, (1988) 233-260
[29] Spaniol, O.: Arithmetik in Rechenanlagen, Teubner 1976
[30] Stiller, A.: AT-Bus, Die Busspezifikation des PC/AT gemäß IEEE P996. c’t 1991 Heft 11, Heft 12
[31] Tavana, D.M.: „Metastability - A study of the Anomalous Behavior of Synchronizer Circuits“ in: Programmable Array Logic Handbook. Monolithic Memories Inc. 1986
[32] Thomson-CSF Semiconducteurs Specifiques: Anzeige für CCD-Sensoren in Electronic Product News April 1994

[33] VLSI: 0.8-Micron Library Synchronizer - Rev.A. VLSI Technology, Inc 1994

9 Anhang A: Programmierbeispiel für den Histogrammer

Anhand eines Flußdiagrammes soll die Programmierung des Histogrammers schematisch dargestellt werden. Der dargestellte Funktionsablauf ist jedoch nicht bindend.



Um die Programmierung des Histogrammers zu demonstrieren, sei hier ein Beispiel in Pseudocode (PASCAL) gezeigt. Eine Definition der in der Registerbelegung definierten Register als Konstanten, der definierten Bitpositionen als Zweierpotenzen und Routinen zur Ausgabe von Werten an den Histogrammer seien vorausgesetzt. Im Beispiel soll ein Histogramm über ein 1024*1024-Bild und über ein 100*100-Fenster innerhalb des Bildes aufgezeichnet werden. Die Statistikwerte über das Bild und das Fenster sollen ebenfalls gebildet werden.

```
Procedure SchreibFunkReg(FReg,Wert:Word); {Beschreibt Funktionsregister}
begin
  WriteHisto(FSubAdr,FReg);
  WriteHisto(FData,Wert);
end;

Procedure SchreibWinPos(FReg,X1,Y1,X2,Y2:Word); {Setzt Fensterkoordinaten}
begin
  WriteHisto(FSubAdr,FReg);
  WriteHisto(FData,X1);
  WriteHisto(FData,Y1);
  WriteHisto(FData,X2);
  WriteHisto(FData,Y2);
end;

Function LiesFunkReg(FReg:Word):Word;   {Liest ein Funktionsregister aus}
begin
  WriteHisto(FSubAdr,FReg);
  LiesFunkReg:=ReadHisto(FData);
end;

begin {Hauptprogramm}
  {*** Vorbereitungen ***}

  SchreibFunkReg(Width,1024);           {Bildgröße definieren}
  SchreibFunkReg(Height,1024);

  {Speicherkonfiguration mit 20 Bit für Fenster 1, 16 Bit für Fenster 3
   und 12 Bit für Fenster 4 setzen. Fenster 1 bleibt fest auf Bildgröße.}
  SchreibFunkReg(MemoryConfiguration,k5043+0*Win1Var);

  {Fenster 1 und Fenster 3 sollen benutzt werden. Fenster 1 reicht für
   1024² Pixel, Fenster 3 für 256² Pixel, Fenster 4 für 64² Pixel.
   Die Position für Fenster 3 ist noch zu definieren:}
  SchreibWinPos(Win3X1,200,150,299,249); {Angabe der inneren Eckpunkte}

  {Aktivierung der (außer Fenster 1) zu benutzenden Fenster:}
  SchreibFunkReg(WinControl,ClrWinStop3);

  {Jetzt muß der Speicher für die Histogramme gelöscht werden:}
  WriteHisto(MemAdr,0);                 {Löschen ab Adresse 0 vorbereiten}
  {Histogrammodus einstellen und Löschen starten:}
  SchreibFunkReg(ModeControl,Histmode00+RamClearStart);

  {Statistikregister löschen:}
  SchreibFunkReg(ClrStat,$ffff);        {Alle Statistikregister löschen}

  {Warten, bis Speicher gelöscht ist:}
  repeat
  until (LiesFunkReg(ModeControl)       {RAMClearState gelöscht?}
         and RAMClearState)<>RAMClearState;

  {***Bild aufnehmen***}

  SchreibFunkReg(WinControl,Start);     {Nächstes Bild wird ausgewertet}
  repeat
```

```
  until (LiesFunkReg(WinControl)        {Stopbit gesetzt?}
         and Stop)=Stop;

  {***Ergebnisse auslesen***}

  {1. Histogramm für Fenster 1 auslesen und Speicher dabei nicht löschen:}
  SchreibFunkReg(ModeControl,WinSelect00+NoReadClear);
  WriteHisto(MemAdr,0);                 {Ab Adresse 0 auslesen}
  for i:=0 to 1023 do begin             {Für alle (1024) Grauwerte}
    Histogramm1[i]:=ReadHisto(MData)    {Lo- und Hi- Word der 20 Bit lesen}
                    +65536*ReadHisto(MData);
  end;
  ...
end.
```

10 Anhang B: Vergleich mit handelsüblichen Histogrammern

Eigenschaft	MAZ-Histogrammer	Harris HSP48410	LSI Logic L64250
Grauwertauflösung	10 Bit	10 Bit	9 Bit
Bildauflösung [Pixel]	4096*4096	4096*4096	4096*4096
max. Pixelfrequenz	25 MHz?	33 bzw. 40 MHz	15 bzw. 20 MHz
Anzahl Fenster	4	1	1
Fensterdefinition	intern/extern	nur Vollbild	intern, nur Vollbild
X/Y-Adreßzähler	ja	-	ja
Histogramm-akkumulation	ja	ja	ja
LUT-Funktion	10 Bit aus 48 Bit	24 Bit	4*9 Bit, parallel zu Histogrammerstellung
Kachelsummen	ja	externe Adr. nötig	ja
Statistikfunktionen	über 4 Fenster: Min., Max., Summe, Quadrats., Schwellensumme	-	-
Interface CPU-kompatibel, Typ	ja, Register	externe Logik erforderlich, RAM	ja, Register
Sonstige Funktionen	Fensterkomparator	RGB-Delay-Line 4..1028 Zyklen; 16/24 Bit RAM; Subtraktion möglich	Hough-Transformation (Hilfstabellen werden in LUTs geladen)

Der MAZ-Histogrammer hebt sich von den beiden kommerziell erhältlichen Typen HSP48410 und L64250 insbesondere durch die Fähigkeit der parallelen Histogrammerstellung in mehreren Fenstern ab. Er verfügt als einziger Baustein über Statistikfunktionen.
Der L64250 bietet als zusätzliche Funktion eine (modifizierte) Hough-Transformation an. Die Hough-Transformation dient der Suche von Geraden in Bildern. Diese Funktion erfordert zusätzliche Vorverarbeitungshardware, da als Eingangsbild an den Histogrammer bereits ein Gradientenbild (d.h. ein Bild der Kontouren) geliefert werden muß. Eine komplette Bearbeitung größerer Bilder ist nur in mehreren Durchläufen möglich.
Der HSP48410 erlaubt besonders hohe Pixelfrequenzen. Der Histogrammspeicher kann über eine Flash-Clear-Funktion in kürzester Zeit insgesamt gelöscht werden. Das RAM-ähnliche Businterface kann vollkommen asynchron betrieben werden. Um alle Funktionen dieses Bausteins vom Hostrechner aus steuern zu können, ist jedoch zusätzlich ein externer Portbaustein notwendig.

11 Anhang C: Grafischer Top-Level und Datenpfad-Einheiten

Der mit dem Compass-System erstellte grafische Top-Level des Histogrammers hat die Funktion, sämtliche in VHDL entworfenen Komponenten untereinander zu verbinden. Weiterhin werden mit dem Datenpfad-Compiler erstellte Einheiten und Elemente wie RAMs, Bus-Repeater und Pads eingebunden.

Die Hauptebene des grafischen Top-Levels (histogrammer) verbindet die in VHDL definierten Komponenten. Der Speicher wird in einer Subebene (memory) eingebunden.
Als Statistikteil kann in den Top-Level wahlweise die komplett in VHDL erstellte Version (histstat) oder die alternative über VHDL und den Datenpfad-Compiler erstellte Version (statteil2) des Statistikteils eingebaut werden. Beide Versionen sind funktionsgleich und verfügen über die gleichen Anschlüsse.
Die alternative Version des Statistikteils (statteil2) ist ebenfalls über eine Subebene des grafischen Top-Levels definiert. In dieser Subebene werden die über den Datenpfad-Compiler erstellten Akkumulatoren für Grauwertsummen (statsum_dp) und Quadratsummen (statsqr_dp) und die VHDL-Komponente für die übrigen Statistikfunktionen eingebunden. Die Schemata der Akkumulatoren liegen wiederum als Subebene unterhalb des Statistikteils.

In der obersten Ebene des grafischen Top-Levels (histopads) wird die darunter liegende Hauptebene (histogrammer) als Symbol dargestellt. In der obersten Ebene sind Pad-Zellen für alle Anschlüsse des Histogrammers eingebunden. Insgesamt verfügt der Histogrammer über 59 Anschlüsse für Ein- und Ausgänge. Aufgrund der hohen Arbeitsfrequenz und der großen Anzahl von Ausgängen sind zusätzlich mindestens ca. 10 Paare von Anschlüssen für die 5 Volt-Versorgung vorzusehen. Deshalb wird der Histogrammer voraussichtlich in ein 84-poliges Pin-Grid-Array Keramikgehäuse eingebaut werden.

Übersicht über die Aufgliederung des grafischen Top-Levels in Ebenen:

Ob. Ebene	Hauptebene	Subebene 1	Subebene 2
histopads	histogrammer	*VHDL-Komponenten*	
		memory	*VHDL-Komponente* *RAMs*
		statteil2 (opt.)	*VHDL-Komponente* statsqr_dp *(Datenpfad-Compiler)* statsum_dp *(Datenpfad-Compiler)*

Die folgenden Grafiken zeigen alle Ebenen des mit dem Logic-Assistant des Compass-Systems erstellten grafischen Top-Levels des Histogrammers und die ebenfalls mit dem Logic-Assistant erstellte Datenpfad-Compiler-Komponente „statsqr_dp“.
Im Anschluß an die Grafiken des Top-Levels ist ebenfalls einer der synthetisierten 4-Bit-Inkrementer der ALU als Grafik abgebildet. Solche Grafiken werden nach Synthese automatisch aus der Netzliste generiert, sobald die Netzliste als Subkomponte des grafischen Top-Levels eingesehen wird.

12 Anhang D: Listing des VHDL-Verhaltensmodelles

Das Verhaltensmodell besteht aus den Files:

HISTADD.VHD	ALU: Inkremeter, Addierer, Halteregister
HISTADR.VHD	RAM-Adreßmultiplexer
HISTBUS.VHD	Bus-Interface zum Hostrechner
HISTCTRL.VHD	Steuerwerk
HISTDECD.VHD	Oberste Registerebene, Adreßdekoder
HISTEA.VHD	Synchronisation der Eingangssignale
HISTINTV.VHD	Intervall-Komparator
HISTMEM.VHD	Verhaltensmodell für ES2-RAM aus CRAM-ECPD 10 (simuliert Speicher bei Verhaltenssimulation ohne Timing)
HHSTMEM.VHD	Latches zur Ansteuerung der RAMs CRAM03A
HISTPOUT.VHD	Multiplexer für Pixelausgang
HISTRMUX.VHD	Multiplexer für Busanbindung des Histogrammspeichers
HISTSFR.VHD	Allgemeine Spezialfunktionsregister
HISTSTAT.VHD	Statistikteil
STAT_DP_VHD.VHD	Alternativer Statistikteil zur Einbindung externer Datenpfad-Compiler Einheiten für Grauwertsummen und Quadratsummen
HISTWIN.VHD	Bildzähler und Fensterkomparatoren
HISTKONST.VHD	Konstanten-Package für Zuordnung der Speicheraufteilungen
HISTOSTI.VHD	Stimulus-File für Gesamt-Simulation aller Funktionen
HISTOGRAMMER.VHD	Aus grafischem Top-Level erzeugtes VHDL-Top-Level File
MEMORY.VHD	s.o.
STATTEIL2.VHD	s.o.
HISTOPADS.VHD	s.o.

Die aufgeführten Quellcode-Files des Verhaltensmodells sind im folgenden abgedruckt. Zum Vergleich zwischen der endgültigen synthetisierbaren Version des Verhaltensmodells und einem ursprünglichen Entwurf, ist als Beispiel auch eine Entwurfsversion des Steuerwerks abgedruckt (histctrl.alt). Sämtliche Quellcode-Files sind in den grafischen Top-Level des Histogrammers eingebunden. Alternativ können sie über ein (hier nicht abgedrucktes) VHDL-File untereinander verbunden werden.

13 Anhang E: Simulationslauf mit Testumgebung „histosti“

Um die Funktion des Histogrammers beurteilen zu können, müssen bei einer Simulation sämtliche Komponenten aktiviert werden und alle Möglichkeiten des Zusammenspiels der Komponenten geprüft werden. Ein detaillierter Test sollte ebenfalls jede Komponente in jeder Konfiguration und für unterschiedliche Wertebereiche prüfen, so daß sämtliche innerhalb des Quellcodes auftretenden Fallunterscheidungen abgetestet werden.

Die Simulationsumgebung „histosti“ (s. Anhang D) wurde zur Validation des Verhaltensmodells des Histogrammers und zur Simulation des Histogrammers anhand der Netzliste erstellt. Dabei wurde Wert darauf gelegt, sämtliche Hauptfunktionen des Histogrammers zu aktivieren, um alle Einheiten und deren Zusammenspiel in allen Betriebsmodi zu prüfen. Die Anzahl der bei der Simulation benötigten Takte sollte gering sein, um kurze Simulationszeiten zu erzielen und die Ergebnisse anhand aufgezeichneter Signale schnell beurteilen zu können. Nur durch die Untersuchung von Signalen innerhalb des Histogrammers können auch Timingprobleme lokalisiert werden.
Durch probeweises Beschreiben und Lesen des Histogrammspeichers können Businterface und Adreßdecoder, RAM-Multiplexer, Speicheransteuerung und Adreßerzeugung getestet werden. Das Aktivieren der Löschfunktion (Flash-Clear) prüft zusätzlich Teile des Steuerwerks. Bei einer Histogrammerstellung werden weiterhin die Pixelschnittstellen, Positionszähler und Fensterkomparatoren, der Statistikteil sowie der Intervallkomparator aktiviert. Dabei können auch die Funktion der Stopbits und der Read-Modify-Write-Zyklus über den Histogramminkrementer überprüft werden. Ein Auslesen der Statistikwerte zeigt das Zusammenspiel zwischen Statistikteil und Businterface. Bei Erstellung von Kachelsummen wird der Addierer der ALU aktiviert. Die Bildung von Summenhäufigkeiten zeigt das Zusammenspiel zwischen dem Akkumulator und dem Addierer der ALU.
Innerhalb der Simulationsumgebung „histosti“ werden die genannten Funktionen anhand eines 4*4-Bildes mit 16 Pixeln ausgeführt. Eine Benutzung nur der hinteren 16 Speicherstellen des Speichers bei der Histogrammerstellung erlaubt es, auch den Löschvorgang auf diese Speicherstellen zu beschränken. Sämtliche Ergebnisse des Simulationslaufes können durch Aufzeichnung der Signale an den Ein- und Ausgängen des Histogrammers sowie durch Aufzeichnung der internen Signale zur Ansteuerung des RAM-Moduls (memory) sichtbar gemacht werden. Zum Test des Statistikteils wurden zeitweise auch Summen und Quadratsummen aufgezeichnet.

Im Anschluß folgt eine tabellarische Beschreibung des Simulationsablaufes. Auf den folgenden Seiten sind wesentliche Passagen des Simulationslaufes mit der Netzliste des Histogrammers ausgedruckt. Die dargestellten Signalverläufe wurden direkt mit dem VHDL-Simulator Voyager generiert.

Die maximale Taktfrequenz des Histogrammers von 50 MHz wurde zur Simulation gewählt, um eventuelle Timing-Probleme deutlich sichtbar zu machen. Für den Pixeltakt wurden 20 MHz gewählt, um unterschiedlich viele Systemtakte für die Verarbeitung jedes Pixels innerhalb des Histogrammers zur Verfügung zu stellen.

Ablauf der Netzlistensimulation des Histogrammers mit der Testumgebung „histosti“:

Zeit [µs]	Aktion	Ergebnisse
0,0..0,04	Initialisierung der Eingangssignale, Reset des Histogrammers	Einstellung des Default-Modus für Speicheraufteilung
0,0..0,4	Bildauflösung 4*4 einprogrammieren	Breite und Höhe werden auf 3 gesetzt
0,4..1,1	Histogramm-Speicher beschreiben: 3F0h: 555555h 3F1h: AAAAAAh	Untere 24 Bit der Speicherstellen (Fenster 1) werden beschrieben
1,1..1,4	Alle Fenster-Stopbits löschen	(Speicherstellen werden dann bei Auslesen gelöscht)
1,4..2,0	Speicherstellen 3F0h und 3F1h (je 24 Bit) probeweise auslesen	Werte: 5555h, 0055h, AAAAh, 00AAh; Speicherstelle wird bei jeweils zweitem Zugriff insgesamt gelöscht
2,0..2,7	Eckkoordinaten für Fenster 3 auf (1,0), (3,2) setzen	(Fenster 3 ist 3*3-Fenster rechts oben im 4*4-Bild, Fenster 1: Vollbild, Fenster 4: nur Pixel 0 (links oben))
2,7..3,9	Automatisches Löschen der Speicherstellen 3F2h bis 3FFh starten und abwarten	Alle Speicherstellen von 3F2h bis 3FFh werden insgesamt gelöscht Ready-Ausgang ist dabei negativ
3,9..4,2	Löschen des Haupt-Stopbits	Ready-Ausgang wird negativ
4,2..5,1	Übertragung eines 4*4-Bildes an den Histogrammer (Pixelwerte: 3*3F0h, 3*3F1h, 3*3F2h, 3*3F3h, 3*3F4h, 1*3F5h)	Pixel in den Histogrammen: Fenster 1 (24 Bit): alle Pixel Fenster 3 (12 Bit): 2*3F0h, 1*3F1h, ... Fenster 4 (12 Bit): 3F0h Grauwertsumme (Fenster 1): 3F23h Quadratsumme (Fenster 1): F92413h Nach Bildübertragung wird Ready-Ausgang positiv
5,1..5,5	Auslesen von Minimum und Maximum für Fenster 3	Werte: 3F0h, 3F3h
5,5..6,2	Kachelmodus setzen, Kantenlänge 8 einstellen, untere 24 Bit von Speicherstelle 0 löschen (durch Auslesen)	Untere 24 Bit von Speicherstelle 0 werden gelöscht: XXXXXX000000h, da Fenster 1 auf Vollbild
6,2..6,5	Löschen des Haupt-Stopbits	Ready-Ausgang wird negativ
6,5..7,4	Erneute Übertragung des 4*4-Bildes an Histogrammer (Pixelwerte: s.o.)	Alle Pixelwerte werden in Speicherstelle 0 (24 Bit) aufsummiert. Summen: 0, 3F0h, 7E0h, BD0h, ..., 3F23h Nach Bildübertragung wird Ready-Ausgang positiv
7,4..8,6	Bildung von Summenhäufigkeiten über Histogramm ab Speicherstelle 3F0h starten und abwarten	Summenhäufigkeit über Fenster 1 wird gebildet. Werte: 3, 6, 9, Ch, Fh, 10*10h Ready-Ausgang ist dabei negativ

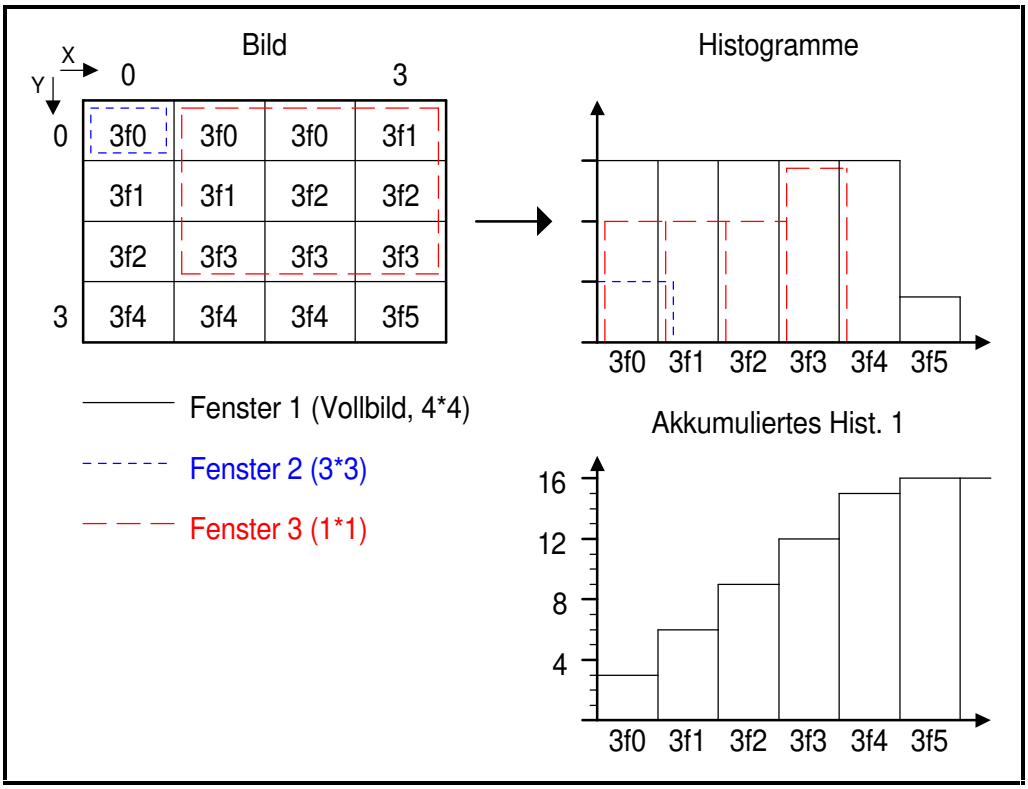


Bild
X
Y
0
3
0
3
3f0 3f0 3f0 3f1
3f1 3f1 3f2 3f2
3f2 3f3 3f3 3f3
3f4 3f4 3f4 3f5
Fenster 1 (Vollbild, 4*4)
Fenster 2 (3*3)
Fenster 3 (1*1)
Histogramme
3f0 3f1 3f2 3f3 3f4 3f5
Akkumuliertes Hist. 1
16
12
8
4
3f0 3f1 3f2 3f3 3f4 3f5

14 Anhang F: Auszüge aus Simulationslauf mit MAZ-Umgebung

Zum detaillierten Test aller Funktionen und Konfigurationen des Histogrammers wurde am MAZ eine Simulationsumgebung für den Histogrammer erstellt. Diese wurde im Rahmen einer Diplomarbeit (Diplomarbeit „Entwicklung und Konzeption einer Systemumgebung zum Testen von Komponenten aus dem Bereich der Bildverarbeitung“, Birgit Schulte, FH Wedel) überarbeitet, so daß sämtliche Einstellungen der Timings und Ein- und Ausgabeinformationen über separate Datenfiles geschehen können.
Die Simulationsumgebung setzt sich aus separaten Modulen für Takterzeugung, Simulation des Hostrechners, Simulation einer digitalen Kamera usw. zusammen. Zum Betrieb der Simulationsumgebung sind eine Datei mit den Systemtaktraten (clock.dat), ein Eingabebild (hier: testbild.in), eine Datei mit dem Timing der Hostschnittstelle (host_times.dat) und eine Steuerdatei (hostctrl.dat) notwendig. Die Steuerdatei enthält Befehle für Schreib- und Lesezugriffe auf den Histogrammer, die vom Hostrechner ausgeführt werden. Weiterhin besteht die Möglichkeit, innerhalb dieser Datei die Bildübertragung an den Histogrammer zu starten und auf das Ready-Signal des Histogrammers zu warten. Die Lesedaten sämtlicher Lesezugriffe des Hostrechners auf den Histogrammer werden in einer weiteren Datei protokolliert (host_rd.dat).

Zur Erprobung der Simulationsumgebung mit dem Netzlisten-Modell des Histogrammers wurde das Timing der Hostschnittstelle angepaßt, so daß keine Setup- und Hold-Zeitverletzungen bei der Simulation enstehen. Eine Steuerdatei und eine zugehörige Bilddatei mit nur vier Pixeln wurden erstellt, um innerhalb weniger Takte wesentliche Funktionen des Histogrammers zu testen.
Getestet werden unter anderem: Beschreiben und Lesen des Histogrammspeichers, Setzen und Löschen der Stopbits, Histogrammerstellung, Überträge und Überläufe innerhalb der ALU in einer Konfiguration und die Statistikfunktionen.

Die Steuerdateien sowie die Ausgabedateien nach Simulation mit der Netzliste des Histogrammers sind im folgenden ausgedruckt. Sie zeigen, daß alle Funktionen spezifikationsgemäß ausgeführt werden.
Die folgenden Abbildungen zeigen die Signale an den Ein- und Ausgängen des Histogrammers und einen Teil der internen Signale bei der Simulation mit den abgedruckten Dateien. Die gesamte Simulation benötigt eine Modellzeit von 7µs. Die Histogrammerstellung erfolgt innerhalb des Zeitbereiches 2,8µs bis 3,4µs. Innerhalb des Zeitbereiches 5,2µs bis 5,8µs werden Summe und Quadratsumme für Fenster 1 ausgelesen.

15 Anhang G: Erhöhung der Bild- oder Grauwertauflösung

Wenn spätere Versionen des Histogrammers eine höhere Bild- und/oder Grauwertauflösung unterstützen sollen, wird eine Reihe von Modifikationen notwendig. Die wesentlichen Änderungen sind im folgenden getrennt nach VHDL-Quellfiles aufgelistet. Die Durchführung dieser Eingriffe erfordert genaue Kenntnisse der Interna des Histogrammers.

1. Änderungen für Erhöhung der maximalen Bildgröße von (2^12 Bit)² auf (2^n Bit)²:

histadd: AkkReg auf 2*n Bit verbreitern
Add24Bit auf 2*n Bit verbreitern
Clip(2*n) einführen und bei gewünschter Konfiguration aktivieren (z.B. bei RamKonf K7050): a) in „Add24Bit"; b) in „Inkrementer"

histadr: Neue Kachelgrößen einführen: 256*256 .. (2^(n-5))²

histdecd: Kommentare anpassen

histpout: Neue Multiplexerstellungen einführen: 25 downto 16 .. (2*n-1) downto (2*n-10)

histstat: Register für Summen (Sum und Sqr) für maximale Werte vergrößern, Register ThrsSum auf 2*n Bit verbreitern

histwin: Bildpositionszähler für X, Y auf je n Bit verbreitern, entsprechend überall dort, wo auf den Bildzähler zugegriffen wird
Fensterkomparatoren und -Register auf n Bit verbreitern

2. Änderungen für Erhöhung der Grauwertauflösung von 10 Bit auf g Bit:

histadd: Summand bei Kachelmodus auf g Bit verbreitern

histadr: Adreßbits für Kachelmodus auf g/2 Bit (X) + g/2 Bit (Y) erweitern, evtl. auch geringere Kachelgrößen einführen (4*4, 2*2)

histctrl: In der Abfrage für RamAdrNull g Bit abtesten

histdecd: Kommentare anpassen

histea: Registerbreiten anpassen

histintv: Registerbreiten anpassen

memory: Auf 2*g Worte erweitern

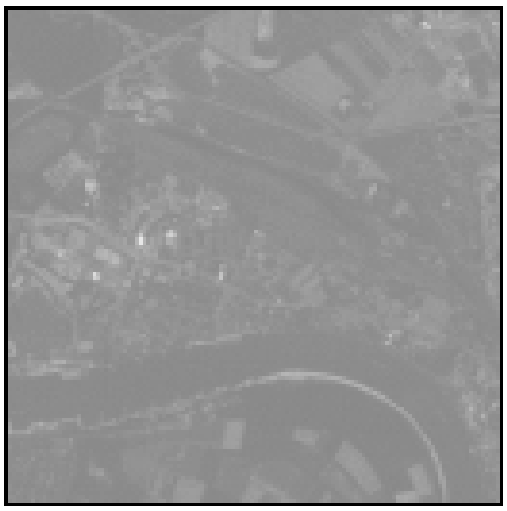
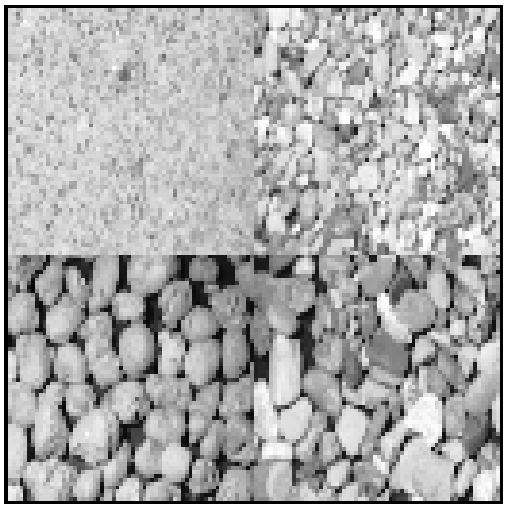
histpout: Multiplexer auf g Bit verbreitern

histstat: Register für Summen für maximale Werte verbreitern (Sum und Sqr)

histwin: X-Zähler auf g Bit verbreitern, falls g>n

16 Anhang H: Beispiele für Egalisierung, Cooccurence-Matrix

Beispiel: Egalisierung einer Luftaufnahme



Beispiel: Darstellung der Cooccurence-Matrix von Gesteinsproben als Grauwertbild: