5.2.8 Zusammenfassung und Übungsaufgaben

Der gesamte Entwurf der FIR-Filter-Schaltung begann mit der Festlegung des nachzubildenden Algorithmus. Dann folgte als erste Strukturentscheidung die Definition einer geeigneten Blockspeicher-Pipeline-Struktur. Für diese wurde ein erstes Simulationsmodell entwickelt, in dem die Werte noch durch Zahlen, die Kommunikation mit der Schaltungsumgebung durch Dateizugriffe und Textausgaben und der Ablauf imperativ modelliert wurden. Im nächsten Schritt wurden die Zahlentypen im Simulationsmodell durch Bitvektortypen ersetzt. Danach wurde die Kommunikation mit der Testumgebung auf Signale umgestellt. Zuletzt wurde die imperative Ablaufbeschreibung durch eine Automatenbeschreibung mit einer Übergangsfunktion und zwei Ausgabefunktionen nachgebildet. Für den skizzierten Entwurfsablauf bis zur korrekt synthetisierten Filterschaltung sind mehrere Wochen Entwurfszeit einzuplanen. Die meiste Zeit davon entfällt auf die Programmierung der Testhilfen, die Ausführung und Auswertung von Tests und auf die Fehlersuche.

Aufgabe 5.1

Überprüfen Sie, ob mit den Typdefinitionen in Abschnitt 5.2.5 Wertebereichsüberläufe möglich sind und wenn ja, für welche Koeffizienten- und Eingabewerte.

Aufgabe 5.2

Berechnen Sie die Simulationsausgaben des FIR-Filters für die Eingabefolge 0, 0, 0, 250, 500, 1000, -1000, -500, 300 und die Filterkoeffizienten

```
constant cROM: tMem := (to_tKoeff(0.3), to_tKoeff(0.9),
                        to_tKoeff(0.9), to_tKoeff(0.3));
```

5.3 Point-of-Interest-Berechnung

Die Zugriffszeit eines Speichers nimmt mit der Speicherkapazität zu. Algorithmen zur Verarbeitung großer Datenmengen – z.B. zur Verarbeitung von Bilddatenströmen – benötigen einen großen Hintergrundspeicher für die Gesamtinformation und kleine schnelle Blockspeicher und Register mit lokalen Datenkopien für die Versorgung der Pipeline-Rechenwerke. Das folgende Beispiel – ein fiktiver Algorithmus für die Suche interessanter Bildpunkte in einem Bilddatenstrom – ist absichtlich sehr komplex gewählt, um auch einmal einen Einblick in die Konzeptionsphase für die Entwicklung eines Funktionsmodells für eine wirklich große Schaltung mit mehreren Blockspeichern und Pipelines zu geben.

5.3.1 Die Zielfunktion

Die interessantesten Bildausschnitte für die Rekonstruktion geometrischer Informationen aus Bildfolgen sind gekrümmte Kantensegmente und Ecken. Für diese Bildobjekte werden bildweise die Positionsänderungen zum vorherigen Bild und daraus weiter deren 2D-Bewegungsbahnen bestimmt. Alle Bewegungsbahnen zusammen ergeben den optischen Fluss, der Rückschlüsse auf die Geometrie und die Bewegung der beobachteten Objekte, auf die Eigenbewegung der Kamera und auf bevorstehende Zusammenstöße der Kamera mit Objekten aus der Szene erlaubt [35].

Die Suche der interessierenden Bildobjekte soll im Beispiel mit einem Satz zweidimensionaler FIR-Filter erfolgen. Für jedes Bild der Bildfolge und für jeden Punkt des Bildes wird zuerst mit jedem Filter eines Filtersatzes ein Übereinstimmungsmaß berechnet. Dann sollen für jeden Bildpunkt aus allen Übereinstimmungsmaßen die beiden größten ausgewählt und aus diesen und den zugehörigen Filternummern je ein Klassifikator gebildet werden, der die Eckenausprägung, -orientierung und -krümmung beschreibt. Im nächsten Verarbeitungsschritt sollen die bildpunktbezogenen Klassifikatoren gemeinsam mit ihren Koordinaten so in eine Liste nach der Stärke ihrer Eckenausprägung einsortiert werden, dass Punkte, die nicht auf Eckpunkten und gekrümmten Kanten von Bildobjekten liegen, aus der Liste herausfallen. Übrig bleibt eine Liste von Datensätzen mit den Koordinaten und Klassifikatoren der interessantesten Bildpunkte, die an die nächste Verarbeitungseinheit geschickt werden. Die nächste Verarbeitungseinheit – hierfür genügt ein leistungsfähiger Rechner – sucht anhand der Klassifikatoren für alle so bestimmten Bildpunkte die korrespondierenden Punkte im vorherigen Bild und bestimmt so deren 2D-Bewegungsbahnen für die weitere Verarbeitung. Die rechenzeitintensiven Teilschritte, die eine Hardware-Lösung verlangen, sind insbesondere die Berechnung der Übereinstimmungsmaße, die Berechnung der Klassifikatoren und das Sortieren.

Bearbeitungsstand: Eine Skizze der Zielfunktion und ihre Aufteilung in mehrere in Hardware oder in Software zu realisierende Teilfunktionen.

5.3.2 Grobkonzept der Hardware-Struktur

Der Hardware-Entwurf einer sehr komplexen Schaltung ist eine Iteration, bei der nach der Aufstellung der Zielfunktion in der Regel als nächstes eine hypothetische Grobstruktur der Hardware aufgestellt wird, um die Machbarkeit zu kontrollieren.

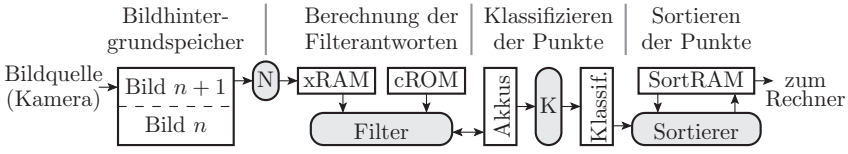
Der skizzierte Algorithmus benötigt voraussichtlich eine mehrstufige Speicherhierarchie. Die Zwischenspeicherung der Bilder verlangt einen relativ großen und entsprechend langsamen Bildspeicher, der z.B. von einer Kamera nachgefüllt wird. Für die Berechnung der Übereinstimmungsmaße muss jeder Bildpunktwert viele Male gelesen werden. Das erfordert einen schnellen und

entsprechend kleinen Blockspeicher, in den nur noch ein kleiner Bildausschnitt passt. Für den Bildausschnittsspeicher ist eine geeignete Datenzuordnung zum Hintergrundspeicher und für die Schnittstelle zwischen beiden Speichern ein geeigneter Nachladealgorithmus zu entwerfen.

Die Berechnung der Filterantworten für mehrere FIR-Filter verlangt voraussichtlich eine ähnliche Hardware-Struktur wie die für die Nachbildung des einzelnen FIR-Filters in Abschnitt 5.2. Abbildung 5.12 unterstellt, dass die Schaltung auch hier wieder aus einem Nur-Lese-Speicher für die Filterkoeffizienten, einem Blockspeicher für die Daten, einer Verarbeitungs-Pipeline und einem Akkumulatorregister für jeden Filter besteht. Die Struktur kann sich im weiteren Entwurfsablauf ändern.

Das nächste Rechenwerk berechnet aus den Akkumulatorwerten die Klassifikatoren. In Abb. 5.12 ist unterstellt, dass es sich dabei um eine Schaltung handelt, die die Berechnung in einem oder mehreren Takten ausführt und das Ergebnis in ein Register schreibt. Die Vergleiche, die für die Suche der beiden größten Übereinstimmungsmaße erforderlich sind, werden vermutlich eine sequenzielle Berechnung nahelegen.

Das nächste Rechenwerk soll die Klassifikatordatensätze sortieren. Dazu benötigt es außer einem Steuerwerk für den Sortierablauf und internen Datenregistern auch einen Blockspeicher für die sortierte Liste. Auch das ist nur eine Hypothese, die wie alle anderen Hypothesen in dieser Entwurfsphase dazu dient, eine Struktur in den weiteren Entwurfsablauf zu bringen.



N	Nachladesteuerung für den Bildausschnittsspeicher
xRAM	Bildausschnittsspeicher
cROM	Konstantenspeicher für Filtermasken
Filter	Pipeline-Rechenwerk für die Filter
K	Rechenwerk für die Klassifikation der Bildpunkte
SortRAM	Sortierspeicher für Koordinaten-Klassifikator-Tupel
Sortierer	Sortierrechenwerk

Abb. 5.12. Speicherhierarchie und Verarbeitungswerke für die Suche interessanter Bildpunkte

Eine nahe liegende Vorgehensweise für die weiteren Entwurfsschritte ist, mit den Hardware-Komponenten fortzufahren, die die meiste Rechenleistung bzw. den größten Datendurchsatz haben müssen. Das ist für die gewählte Zielfunktion die Berechnung der Übereinstimmungsmaße, die für alle Bildpunkte aller Bilder einer Bildfolge mit allen Filtermasken ausgeführt werden muss. Der nächste Schritt sei die Kontrolle, dass es für die gefundene Lösung einen

ausreichend schnellen Nachladealgorithmus für den Blockspeicher aus dem Hintergrundspeicher gibt. Dann sollen Ablaufskizzen für die Berechnung der Klassifikatoren und die Sortieroperationen folgen.

Bearbeitungsstand: Eine hypothetische Hardware-Struktur und ein Fahrplan für die nächsten Entwurfsschritte.

5.3.3 Optimierung der 2D-FIR-Filter

Abbildung 5.13 a veranschaulicht den Algorithmus für die Bildung der Übereinstimmungsmaße. Die Bildpunkte bilden ein zweidimensionales Feld. Der Bildpunkt, dessen Umgebung klassifiziert wird, hat die Spalten- und Zeilenkoordinaten i und j. Die Filterkoeffizienten der 2D-FIR-Filter bilden Masken, die nacheinander mittig über alle Bildpunkte gelegt werden. Im Beispiel sind die Filtermasken zweidimensionale Felder der Größe 5×5. Die Filterantwort ist die Summe der abgedeckten Bildpunkte multipliziert mit den Koeffizienten der Filtermaske.

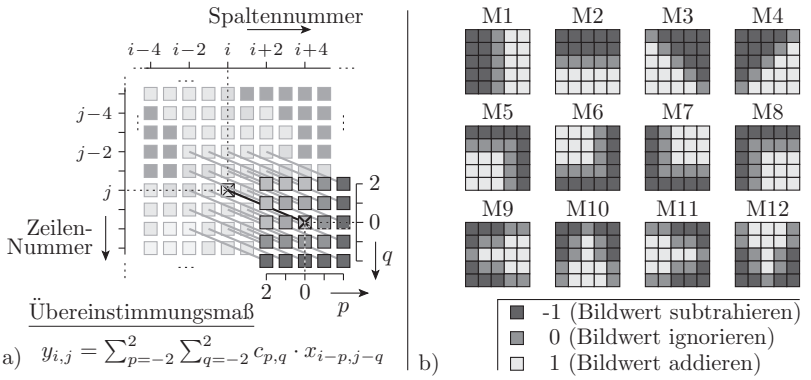


Abb. 5.13. a) Algorithmus zur Berechnung der Übereinstimmungsmaße b) Filtersatz

Vor einer Hardware-Nachbildung sind alle Möglichkeiten zur Verkürzung der Rechenzeit, die der Algorithmus bietet, auszuschöpfen. Für FIR-Filter ist eine dieser Möglichkeiten die Optimierung der Filtermasken. Für die gewählte Zielfunktion muss der Filtersatz aus Bildern von Ecken mit unterschiedlichen Orientierungen und Krümmungen bestehen. Der Rechenaufwand wird von der Anzahl der Masken, der Größe der Masken und vom Wertebereich der Filterkoeffizienten bestimmt. Abbildung 5.13 b zeigt den gewählten Filtersatz. Er besteht aus zwölf Masken mit einer Größe von 5×5 Bildpunkten und einem Koeffizientenwertebereich $c_{p,q} \in \{-1, 0, 1\}$. Die Beschränkung der Koeffizientenwerte auf »0«, »1« und »-1« erspart die Multiplikationen. Die mit »1« abgedeckten Bildpunktwerte sind zum Akkumulatorwert zu addieren, die mit

»−1« abgedeckten Bildpunktwerte sind zu subtrahieren und die mit »0« abgedeckten Werte sind zu ignorieren. Weiterhin sind die Filter so aufgebaut, dass für jeden Filter etwa dieselbe Anzahl von Bildwerten addiert und subtrahiert werden. Eine betragsmäßig größere Filterantwort erlaubt so ohne weitere Nachbearbeitung einen Rückschluss auf eine größere geometrische Ähnlichkeit mit dem Kantenverlauf in der Filtermaske, bei einer positiven Filterantwort mit dem Maskenbild selbst und bei einer negativen Filterantwort mit dem Negativ des Maskenbilds.

Der skizzierte Algorithmus ist extrem rechenzeitintensiv. Es sind etwa 10 bis 20 Millionen Bildpunkte je Sekunde zu klassifizieren. Für jeden Bildpunkt und jede Filterantwort sind 25 Indexrechnungen für Daten- und Koeffizientenadressen und 25 bedingte Additions-Subtraktions-Operationen auszuführen. Beim Entwurf der Blockspeicher-Pipeline-Struktur für die Filterberechnung ist der Engpass genau wie in Abschnitt 5.2 die Speicherbandbreite des lokalen Blockspeichers für die Daten. Eine nahe liegende Optimierung ist, einmal gelesene Bildpunktwerte parallel mit allen Maskenwerten zu verknüpfen. Dazu muss der Koeffizientenspeicher bei jedem Zugriff die Koeffizientenwerte $c_{p,q}$ für alle Masken liefern. Des Weiteren sind für jede Filtermaske ein eigenes Additions-Subtraktions-Werk und ein eigener Akkumulator erforderlich (Abb. 5.14).

Bearbeitungsstand: Ein in Hardware umsetzbarer Algorithmus zur Berechnung der Filterantworten. Durch eine geschickte Koeffizientenwahl wurden die Multiplikationen eingespart und mit einem geschickten Parallelisierungsschema die Anzahl Blockspeicherzugriffe minimiert.

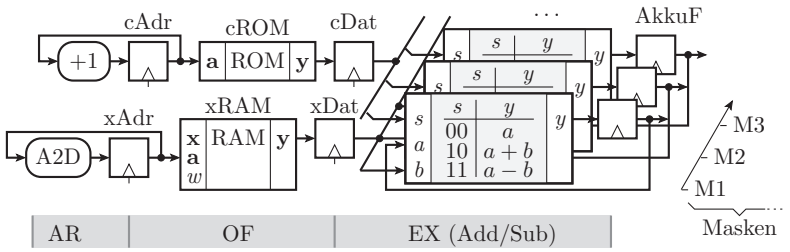
5.3.4 Entwurf der Filter-Pipeline

Nach dem Vorabschnitt besteht ein Filterberechnungsschritt in der Bestimmung der nächsten Blockspeicheradressen für die zu lesenden Koeffizienten und Daten, den Leseoperationen selbst und in Abhängigkeit vom gelesenen Koeffizientenwert in einer bedingten Addition oder Subtraktion des gelesenen Datenwertes zum Wert des jeweiligen Akkumulationsregisters. Die Blockspeicherzugiffe erhalten wie in Abschnitt 5.2.1 eine eigene Pipeline-Phase, die Adressrechnungen davor und die Akkumulationen danach gleichfalls (Abb. 5.14). Die Filter-Pipeline in Abb. 5.5 besitzt zusätzlich eine Phase für die Multiplikation, die hier durch die Beschränkung der Filterkoeffizientenwerte mit eingespart wird.

Mit der Skizze der Filter-Pipeline lassen sich die ersten VHDL-Datentypen und Operationen definieren. Die Filterkoeffizienten sind Aufzählungstypen mit drei Werten:

```
type tOperation is(add, sub, keine_Operation);
```

Der gesamte Koeffizientenvektor ist ein Feld aus Elementen von diesem Typ:



cAdr Koeffizientenadresse	xRAM Bildausschnittsspeicher	Pipelinestufen
xAdr Datenadresse	xDat Datenwert	AR Adressrechnung
A2D 2D-Adressrechnung	cDat Koeffizientenwert	OF Operanden lesen
cROM Koeffizientenspeicher	AkkuF Akku-Register 0 bis 11	EX Ausführung

Abb. 5.14. Speicherhierarchie und Verarbeitungswerke für die Berechnung der Übereinstimmungsmaße

```
subtype tFilterIdx is NATURAL range 0 to N_Mask-1;
type tKoeff is array (tFilterIdx) of tOperation;
```

Der Akkumulator muss wie die Daten einen Bitvektortyp zur Darstellung vorzeichenbehafteter Zahlen haben und soll, um Wertebereichsüberläufe auszuschließen, vier Bit breiter als ein Datenregister sein[6]:

```
constant cDatenbreite: NATURAL := ...;
subtype tDaten is tSigned(cDatenbreite - 1 downto 0);
subtype tAkku is tSigned(cDatenbreite + 3 downto 0);
```

Die Funktion zur Berechnung des Folgewertes für einen einzelnen Akkumulator ist

```
function FilterAdd(Akku: tAkku; x: tDaten; s: tOperation)
  return tAkku is
begin
 case s is
  when add => return Akku + x;
  when sub => return Akku - x;
  when others => return Akku;
 end case;
end function;
```

⇒WEB-Projekt: P5.3/POI_pack.vhdl

Alle Akkumulatoren zusammen bilden ein Feld:

```
type tAkkFeld is array (tFilterIdx) of tAkku;
```

Die auf das gesamte Akkumulatorfeld anzuwendende Methode ist eine Iteration über alle Akkumulatoren mit der auf den einzelnen Akkumulator anzuwendenden Funktion:

[6] Der sechzehnfache Wertebereich genügt, weil keine der Filtermasken mehr als sechzehn Koeffizienten ungleich null enthält.

```
procedure AllFilterAdd(signal AkkuF: inout tAkkFeld;
      x: tDaten; c: tKoeff) is
begin
 for idx in tKoeff'RANGE loop
  AkkuF(idx) <= FilterAdd(AkkuF(idx), x, c(idx));
 end loop;
end procedure;
```

⇒WEB-Projekt: P5.3/POI_pack.vhdl

Bearbeitungsstand: Skizze der Blockspeicher-Pipeline-Struktur zur Bestimmung der Übereinstimmungsmaße und erste Datentypvereinbarungen und Bearbeitungsmethoden für diese Datentypen.

5.3.5 Nachladen des Bildausschnittsspeichers

Das wichtigste Optimierungskriterium für die Größe und Organisation des Bildausschnittsspeichers ist ein beherrschbarer Datentransfer zwischen dem Bildhintergrundspeicher und dem Bildausschnittsspeicher. Ziel sei, dass im Mittel für zehn Lesezugriffe auf den Bildausschnittsspeicher nicht mehr als ein Bildwert nachgeladen werden muss.

Eine nahe liegende Organisationsform für den Bildausschnittsspeicher ist, ihn fünf Bildzeilen groß zu wählen. In Abb. 5.15 werden am Anfang von jedem Bild die ersten fünf Zeilen in den Bildausschnittsspeicher geladen. Anschließend werden für jede Filterberechnung 25 Bildwerte aus dem Bildausschnittsspeicher gelesen und das Filterfenster eine Spalte nach rechts bewegt. Das nächste Filterfenster nutzt 20 Bildpunkte des vorherigen Fensters. Von den freiwerdenden fünf Bildpunkten werden vier noch für die Filteroperationen für die nächste Zeile gebraucht. Nur der Bildwert der niederwertigsten Zeile ist durch den Bildwert für fünf Zeilen weiter zu ersetzen. Am Zeilenende springt das Maskenfenster an den Anfang der nächsten Zeile und hinterlässt fünf nachzuladende Bildpunktspeicherplätze.

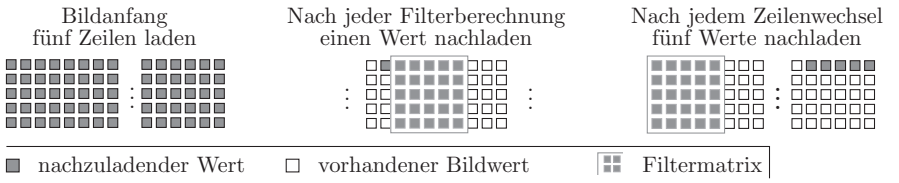


Abb. 5.15. Bildausschnittsspeicher mit Platz für fünf komplette Bildzeilen

Beim Nachladen eines Bildpunktwertes müssen alle anderen Bildpunkte im Bildausschnittsspeicher ihren Speicherplatz behalten. Denn Zeit für die Verschiebung der Datenwerte ist nicht verfügbar. Die Werte der fünften Zeile überschreiben die Werte der nullten Zeile, die Werte der sechsten Zeile die der

ersten Zeile etc. Das führt bei der Berechnung der Übereinstimmungsmaße zu einer zirkularen, von der Zeilennummer des zu klassifizierenden Bildpunktes abhängigen Verschiebung der Adressen für die Filterkoeffizienten (Abb. 5.16).

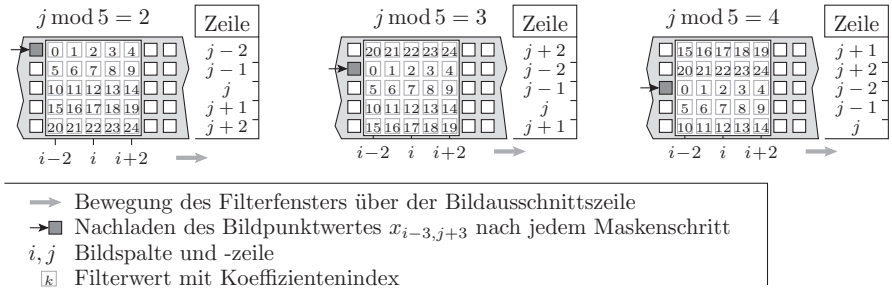


Abb. 5.16. Zirkulare Verschiebung der Koeffizientenadressen im Bildausschnittsspeicher

Ein Bildausschnittsspeicher für fünf Zeilen ist relativ groß und entsprechend langsam. Abbildung 5.17 zeigt eine alternative Lösung mit einem nur fünf Spalten mal 25 Zeilen großen Bildausschnittsspeicher. Am Bildanfang wird dieser mit den ersten 125 Bildpunktwerten der linken oberen Bildecke beschrieben (Zeile 0 bis 24, Spalte 0 bis 4). Danach wird das Filterfenster von Zeile zwei bis 22 nach unten bewegt. Nach den ersten 20 Schritten muss genau wie bei der Bewegung des Filterfensters von links nach rechts in Abb. 5.15 immer nur ein Wert nachgeladen werden. Nach dem 21. Schritt folgt ein Spaltenwechsel. Dafür sind die fünf Werte in Spalte null, Zeile 20 bis 24 nachzuladen. Der Nachladerhythmus bleibt bis zum Zeilenende gleich: zwanzigmal einen und einmal fünf Werte nachladen. Nach der Berechnung der Klassifizierungswerte für die ersten 21 Zeilen muss der Bildausschnittsspeicher komplett neu mit den ersten fünf Werten der Zeilen 20 bis 25 geladen werden. Für den nächsten und alle weiteren Zeilenblöcke verschiebt sich das Zeilenfenster um

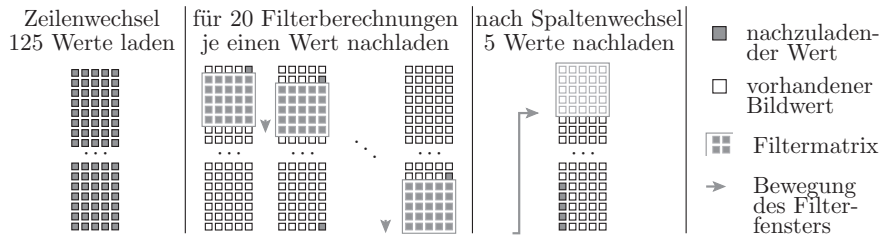


Abb. 5.17. Nachladeschema mit einem Bildausschnittsspeicher der Größe 5×25 Bildpunkte.

21 Zeilen nach unten. Die Gesamtanzahl der Nachladeoperationen beträgt für jeden Block von 21 Zeilen 125 nachzuladende Werte am Zeilenanfang und für jede Bildspalte innerhalb des Blocks zwanzigmal einen gefolgt von einmal fünf nachzuladenden Werten. Das ist nicht wesentlich mehr als bei der ersten Lösung, bei der der Bildausschnittsspeicher Platz für fünf komplette Bildzeilen bieten musste. Auch in der modifizierten Lösung behalten alle einmal in den Bildausschnittsspeicher geladenen Bildwerte bis zu ihrem Überschreiben ihre Adresse. Die Adressrechnungen für die nachzuladende Bildpunktadresse, die zugehörige Adresse im Bildausschnittsspeicher etc. sind für den FIR-Filter in Abschnitt 5.2 durch Konstantenzuweisungen und Additionen beschreibbar. Da die Filterfensterbewegung hier spaltenweise erfolgt, muss man sich die zirkulare Verschiebung der Filterkoeffizienten, die in Abb. 5.15 b für eine spaltenweise Bewegung gezeigt ist, um 90° gedreht vorstellen.

Bearbeitungsstand: Es wurde gezeigt, dass es mit einem relativ kleinen Blockspeicher möglich ist, die erforderlichen Bilddaten mit einem mittleren Verhältnis aus der Anzahl der Speicherzugriffe auf den schnellen Bildausschnittsspeicher zur Anzahl der Speicherzugriffe auf den langsamen Hintergrundspeicher von mehr als zehn zu eins nachzuladen. Damit ist gezeigt, dass das Konzept mit dem großen langsamen Hintergrundspeicher und dem kleinen schnellen Bildausschnittsspeicher prinzipiell funktioniert.

5.3.6 Suche der betragsmäßig größten Filterantworten

Aus den zwölf Filterantworten, die der bisherige Algorithmus für jeden Bildpunkt berechnet, sollen mit dem nächsten Teilalgorithmus je ein Klassifikator und ein Signifikanzmaß berechnet werden. Der Klassifikator soll, wenn es sich um einen Eckpunkt handelt, die Orientierung und Krümmung der Ecke beschreiben und aus den beiden Filternummern mit den betragsmäßig größten Filterantworten und den Vorzeichen dieser Filterantworten gebildet werden. Das Signifikanzmaß, an dem später die auffälligsten Eckpunkte ausgewählt werden, ist eine Funktion der beiden größten Beträge.

Die Suche der beiden größten Beträge und der zugehörigen Filternummern benötigt ein Register »Max1« für die betragsmäßig größte und ein Register »Max2« für die betragsmäßig zweitgrößte Filterantwort. Jedes der beiden Register muss einen Datensatz aus der Filternummer sowie aus dem Betrag und dem Vorzeichen der Filterantwort speichern:

```
type tFilterDat is record
 FNr: tFilterIdx; -- Filternummen
 betrag: tAkku;   -- Betrag des Akkuwertes
 s: STD_LOGIC;    -- Vorzeichen des Akkuwertes
end record;
signal Max1, Max2: tFilterDat;
```

Zusätzlich wird eine Konstante für einen leeren Datensatz benötigt:

```
constant cFDat_leer:tFilterDat := (FNr=>0, Betrag=>"0...0", s=>'0');
```

Im ersten Schritt wird das Signal für »Max1« mit dem Datensatz des ersten Filters und das Signal »Max2« mit der Konstanten für den leeren Datensatz beschrieben. In den nachfolgenden elf Schritten wird nacheinander für alle anderen Filterantworten der Betrag gebildet und mit dem von »Max1« verglichen. Ist er größer, übernimmt »Max1« den Datensatz dieses Filters und reicht seinen aktuellen Datensatz an das Signal »Max2« weiter. Wenn der Betrag des betrachteten Filters kleiner als der von »Max1« aber größer als der von »Max2« ist, übernimmt »Max2« den Filterdatensatz. Der skizzierte Algorithmus lässt sich als Prozedur mit dem Akkumulatorfeld als Eingabesignal und »Max1« und »Max2« als les- und veränderbare Signale beschreiben:

```
procedure SortUeMass(AkkuF: tAkkFeld; signal Max1, Max2: inout
                                                  tFilterDat) is
begin
 Max1 <= (FNr=>0, betrag=>abs(AkkuF(0)), s=>Vorzeichen(AkkuF(0)));
 Max2 <= cFDat_leer;
 wait for tP;
 for idx in 1 to AkkuF'HIGH loop
  if abs(AkkuF(idx))>Max1.betrag then
   Max1 <= (FNr=>idx, betrag=>abs(AkkuF(idx)),
            s=>Vorzeichen(AkkuF(idx)));
   Max2 <= Max1;
  elsif abs(AkkuF(idx))>=Max2.Betrag then
   Max2 <= (FNr=>idx, betrag=>abs(AkkuF(Idx)),
            s=>Vorzeichen(AkkuF(idx)));
  end if;
  wait for tP;
 end loop;
end procedure;
```

Die Prozedur gibt den Kontrollfluss immer nach zwölf Zeitschritten zurück. Während dieser zwölf Schritte dürfen sich die Akkumulatorinhalte nicht ändern. Entweder die Maximabestimmung arbeitet mit einer Kopie der Akkumulatorwerte oder die Berechnung der Filterantworten und die Suche der beiden betragsmäßig größten Filterantworten erfolgen nacheinander. In Abschnitt 5.3.8 folgt später die Entscheidung für »nacheinander«, weil sich in diesem Abschnitt herausstellen wird, dass nicht die Filterberechnung, sondern das Sortieren am längsten dauert. Blockspeicher und Pipelines werden für die Suche der beiden größten Übereinstimmungsmaße nicht benötigt.

Bearbeitungsstand: Algorithmusskizzen bis zur Bestimmung der beiden Filterantworten mit der besten Übereinstimmung für jeden Bildpunkt.

5.3.7 Klassifikation

Nach der Bestimmung der Maxima folgt die Klassifikation. Der Klassifikationsdatensatz soll aus den Punktkoordinaten, Datenobjekten für zwei Vorzeichen und zwei Filternummern sowie einem Maß für die Eckenausprägung bestehen:

```
type tKlassPkt is record
 FNr1, FNr2: tFilterIdx;
 s1,s2: STD_LOGIC;
 Betrag: tAkku; -- Mass für die Eckenausprägung
 i: INTEGER range 2 to Spaltenanzahl-2;
 j: INTEGER range 2 to Zeilenanzahl-2;
end record;
```

Die Klassifikation ist eine Funktion mit den beiden signifikantesten Filterdatensätzen als Eingabe und einem Klassifikationsdatensatz als Ergebnis. Wenn nach der Sortierung der Betrag des Filters mit der besten Übereinstimmung »Max1.Betrag« mehr als doppelt so groß wie der des Filters mit der zweitbesten Übereinstimmung »Max2.Betrag« ist, soll der Klassifikationsdatensatz nur aus den Werten von »Max1« und sonst aus den Werten von »Max1« und »Max2« zusammengesetzt werden. Auf der Hardware-Ebene verbergen sich dahinter Registerzuweisungen mit umschaltbarem Datenfluss:

```
function Klassifikation(Max1, Max2: tFilterDat)
                                    return tKlassPkt is
begin
 if (Max1.Betrag sra 1) > Max2.Betrag then
  return (FNr1=>Max1.FNr, s1=>Max1.s, FNr2=>Max1.FNr, s2=>Max1.s,
   Betrag=>Max1.Betrag + (Max1.Betrag sra 1), i=>..., j=>...);
 else
  return (FNr1=>Max1.FNr, s1=>Max1.s, FNr2=>Max2.FNr, s2=>Max2.s,
   Betrag=>Max1.Betrag + Max2.Betrag, i=>..., j=>...);
 end if;
end function;
```

(**sra** 1 – vorzeichenbehaftete Halbierung durch eine arithmetische Rechtsverschiebung um eine Stelle).

Bearbeitungsstand: Algorithmusskizzen bis zur Bestimmung der Bildpunktklassifikatoren.

5.3.8 Sortieren der klassifizierten Punkte

Der nachfolgende Sortieralgorithmus ähnelt einem Bubble-Sort, bei dem nur eine begrenzte Anzahl von Werten im Sortierspeicher gehalten wird. Die Datensätze mit geringer Signifikanz fallen unten heraus. Der Sortierspeicher sei ein Blockspeicher, dessen Elemente klassifizierte Bildpunkte sind:

```
type tPtrKlassFeld is array (NATURAL range <>) of tKlassPkt;
```

Zu Beginn der Bearbeitung eines neuen Bildes ist jedes Feldelement mit einer Konstanten für den Wert »leer« zu beschreiben:

```
constant Pkt_leer: tKlassPkt := (FNr1=>0, s1=>'0', FNr2=>0,
     s2=>'0', Betrag => "0...0", i=>0, j=>0);
```

Bei einem Hardware-Sortierer ist es wichtig, die Anzahl der Blockspeicherzugriffe zu minimieren, da diese sequenziell erfolgen. Der in Abb. 5.18 skizzierte Algorithmus verwendet dazu zwei Register, »RegL« für den aus dem Blockspeicher gelesenen Wert und »RegE« für den in den Blockspeicher einzusortierenden Wert:

```
signal RegL, RegE: tPktKlass;
```

Bei unserem speziellen Bubble-Sort wird für alle einzusortierenden Werte nacheinander jedes Element im Sortierspeicher gelesen und mit dem einzusortierenden Wert verglichen. Wenn der gelesene Wert kleiner ist, wird er in das Feldelement davor kopiert. Anderenfalls, wenn er größer ist, wird der einzusortierende Wert in das vorherige Feldelement und der gelesene Wert in das Register für den einzusortierenden Wert kopiert. Beim ersten Einsortierschritt für jeden neuen Datensatz hat das Feldelement, das gelesen wird, keinen Vorgänger, so dass der Datensatz für den Bildpunkt mit der geringeren Signifikanz aus der Liste herausfällt.

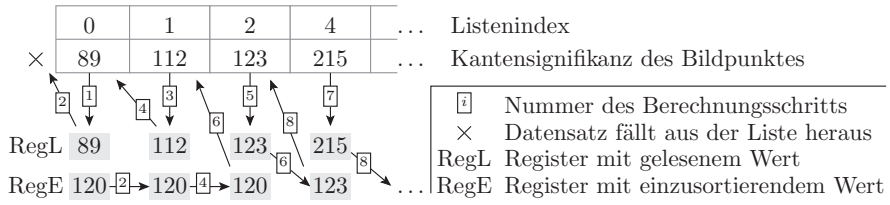


Abb. 5.18. Sortieren der klassifizierten Punkte

Jeder Sortierschritt benötigt eine Schreib- und eine Leseoperation, die nacheinander ausgeführt werden müssen. Das Sortieren dauert folglich mindestens $2 \cdot N_{\mathrm{POI}}$ Takte (N_{POI} – Anzahl der zu berechnenden »interessanten« Bildpunkte). Die Filteroperation benötigt für jeden Bildpunkt 25 Schritte. Da das Sortieren offensichtlich mehr Zeit benötigt, ist es in Abschnitt 5.3.6 nicht notwendig, während der Suche der beiden betragsmäßig größten Werte schon mit der nächsten Filteroperation zu beginnen. Wenn die Filterberechnung und die Klassifizierung nacheinander erfolgen, dauert das insgesamt etwa 40 Takte. Diese Zeit genügt, um den zuvor klassifizierten Punkt in eine Liste mit bis zu 20 Bildpunkt-Klassifikator-Tupeln einzusortieren. Die Listenlänge ist

gleichzeitig die maximale Anzahl der interessanten Bildpunkte, die der Algorithmus bestimmt. Für das gesamte Bild sind 20 interessante Punkte zu wenig. Eine Möglichkeit, ihre Anzahl zu erhöhen, ist, das Gesamtbild in Teilbilder zu unterteilen und die interessanten Bildpunkte für jedes Teilbild extra zu berechnen. Jedem Teilbild wird dazu ein Teilbereich im Sortierspeicher zugeordnet.

> *Bearbeitungsstand: Es wurde gezeigt, dass der gesamte Algorithmus in der skizzierten Weise in Hardware umsetzbar ist. Für einige Datenobjekte wurden bereits die Datentypen und die Unterprogramme für ihre Bearbeitung entwickelt.*

Das Entwurfsprojekt soll hier enden, obwohl es noch lange nicht abgeschlossen ist. In den nächsten Schritten sind auch für die restlichen Datenobjekte Typen zu vereinbaren, Bearbeitungsmethoden zu programmieren, Testbeispiele zu entwickeln, die algorithmischen Bausteine zu größeren Systemen bis hin zum Gesamtsystem zusammenzusetzen, auch dafür wieder Testrahmen und Testbeispiele zu entwickeln und vieles mehr. Der noch zu leistende Entwicklungsaufwand ist erheblich, denn wir reden hier immerhin über eine zu entwerfende Schaltung der Größe eines kleinen Graphikprozessors.

5.3.9 Zusammenfassung

Das Beispiel hat das Herangehen an die Umsetzung komplexer, rechenzeitintensiver Algorithmen in Hardware demonstriert. Gegenüber den etwas kleineren Entwurfsaufgaben wie dem Entwurf eines FIR-Filters in Abschnitt 5.2 und dem Entwurf des CORDIC-Rechenwerks in Abschnitt 3.4.6 kommen zusätzliche vorangestellte Entwurfsschritte hinzu, die den Gesamtalgorithmus auf Hardware und Software verteilen, den Hardware-Teil weiter aufspalten, die Teilalgorithmen zuerst skizzenhaft durch Hardware nachbilden, dabei die Machbarkeit untersuchen, bis dann gezielt mit der Entwicklung des Simulationsmodells für die Zielfunktion begonnen wird. Neu waren weiterhin einige Hardware-typische Optimierungsansätze, insbesondere das Konzept von Hinter- und Vordergrundspeicher. Bei Letzterem geht es darum, den Speicherengpass für Zugriffe auf einen großen und langsamen Hintergrundspeicher dadurch zu umgehen, dass mit einer Datenkopie in einem viel kleineren und entsprechend schnelleren Vordergrundspeicher gearbeitet wird. Abgesehen von den algorithmenspezifischen Optimierungen und Tricks war ansonsten eigentlich nichts grundlegend neu. Die bis hierher behandelten Entwurfstechniken erlauben, auch sehr große Schaltungen zu entwerfen.

5.4 Entwurf eines RISC-Prozessors

Ein Prozessor ist ein universeller Automat, der nacheinander Befehle aus einem Speicher liest und abarbeitet. Der Entwurf erfolgt wie bei jeder anderen