DIPLOMARBEIT

# Plattform zur Bildbearbeitung autonomer Kleinstroboter

ausgeführt zur Erlangung des akademischen Grades
eines Diplom-Ingenieurs unter der Leitung von

o. Univ. Prof. Dr. techn. Dietmar Dietrich

Univ. Ass. Dipl.-Ing. Stefan Mahlknecht

am

Institut für Computertechnik (E384)

der Technischen Universität Wien

durch

Dietmar Bruckner
9925419
Kreuzberg 97, 3922 Großschönau

Wien, am 5. Jänner 2004 .....

## Kurzfassung

Die immer größer werdende Menge an digitalen Bild- und Videodaten und der immer weiter verbreitete Einsatz von optischen Sensoren macht eine effiziente Bildbe- und verarbeitung sowie effiziente Bild- und Videokompression immer wichtiger. Die Anforderungen an Hardware, Software und Übertragungs- und Speichermedien steigen unaufhaltsam. Beispiele, wohin diese Entwicklung schon geführt hat, und wohin sie noch führen soll oder kann, gibt es in allen Bereichen des täglichen Lebens: Der Einsatz von digitalen Bildern in Photographie, Medizin oder als Sensorsignale bei Industrierobotern, die Entwicklung von immer kleineren autonomen Systemen (welche vielleicht einmal im menschlichen Blutkreislauf zur Abtötung von Viren eingesetzt werden könnten), sowie die stetig steigende Zahl an Überwachungseinrichtungen. Wichtige Teilaspekte dieser Entwicklungen liegen in der Miniaturisierung der Geräte und der Echtzeitfähigkeit von Kompressionsalgorithmen und Übertragungskanälen. In dieser Arbeit soll ein System realisiert werden, dass alle obigen Punkte verknüpft und eine möglichst kleine, leistungsfähige und stromsparende Plattform zur Bildbearbeitung darstellt. Das Hauptanwendungsgebiet dieses System können autonome Klein- und Kleinstroboter sein. Dafür ist es notwendig, dass Bauteile mit hoher Verarbeitungsleistung, kleiner Baugröße und geringem Stromverbrauch sowie effiziente Bildkompressionsalgorithmen Verwendung finden. Ziel dieser Arbeit ist es, ein System vorzustellen, dass Bilder und Videos aufnehmen und speichern, sowie in komprimierter Form drahtgebunden und/oder drahtlos übertragen kann. Weiters soll genügend Rechenleistung für eine Bildverarbeitung mit Objekterkennung bei Verwendung als intelligenter Sensor vorhanden sein.



## Abstract

The more and more growing amount of digital picture- and video-data and the widespread use of optical sensors in industrial applications make efficient image editing and image processing as well as efficient picture- and video-compression algorithms necessary. The requirements on hardware, software and transmission- and storage-media are growing irresistible. Some examples, of how far this development has gone, and how much further it can go, are present in all parts of daily life: The application of digital images in photography, medicine or as a sensor signal in industrial robots, the development of very small autonomous system (which should be small enough to be able to kill viruses in the human blood circuit sometimes) as well as the steady increase in monitoring and observation systems. Some of the foremost parts of this development are miniaturization of the hardware-components and real-time abilities of the software-algorithms and transmission lines. The aim of this project is to realize a system which merges both parts to a small, low power und high performance platform for image processing. The major field of application of the system could be autonomous micro robotics. To reach this goal, it is necessary, that all hardware-components have high performance, small packages and low power consumption and the software-algorithm should be efficient and real-time-capable. The aim of the project is to introduce a system which is capable of taking photos and videos and transmitting them compressed by wire or by air. Furthermore the system shall have enough computational power for image processing and object recognition for use as intelligent sensor in robotics.

# Danksagung

Mein Dank gilt Herrn o. Univ. Prof. Dr. techn. Dietmar Dietrich, Institutsvorstand, und Herrn DI Stefan Mahlknecht, Diplomarbeitsbetreuer, für die ständige Hilfe und gute Betreuung während der Erstellung dieser Arbeit.

Auch möchte ich den Mitgliedern des Projekts „Tinyphoon“ unter der Leitung von Herrn Dr. techn. Gregor Novak MSc. für ihre Unterstützung bei projektspezifischen Fragen danken.

Weiterer Dank gilt allen Kollegen und Freunden, die mir bei der Erstellung dieser Arbeit eine große Hilfe waren und immer ein offenes Ohr für meine Problemchen hatten.

Mein größter Dank gilt allerdings meinen Eltern Josef und Anni Bruckner, die mir meine Ausbildung ermöglicht und mich immer in meinen Entscheidungen bestärkt haben und meiner Lebensgefährtin Eva, die immer hinter mir stand und ohne die mein Leben wahrscheinlich ganz anders verlaufen wäre …

# Inhaltsverzeichnis

# Abkürzungen

| | |
|---|---|
| 2DLS | 2-Dimensional Logarithmic Search (Algorithm) |
| AC | Alternating Current |
| A/D | Analog/Digital (Wandler) |
| ASCII | American Standard Code for Information Interchange |
| baud | Bits per Second |
| BGA | Ball Grid Array |
| BM | Block Matching (Algorithm) |
| BMP | Bitmap |
| CAN | Controller Area Network |
| $C_b$ | Chrominanz-blau |
| cbp | coded block pattern |
| CD | Compact Disc |
| CPU | Central Processing Unit |
| $C_r$ | Chrominanz-rot |
| DC | Direct Current |
| DCT | Discrete Cosine Transform |
| DFD | Displaced Frame Difference |
| DPCM | Differential Pulse Code Modulation |
| DVD | Digital Versatile Disk (or Digital Video Disk) |
| DVF | Displacement Vector Field |
| dMD | differential Motion Displacement |
| DVB | Digital Video Broadcast |
| DWT | Discrete Wavelet Transform |
| Eob | End of Block |
| FET | Field Effect Transistor |
| FPGA | Field Programmable Gate Array |
| GIF | Graphics Interchange Format |
| GFSK | Gaussian Frequency Shift Keying |
| GOP | Group of Pictures |
| GPIO | General Purpose IO |
| GUI | Graphical User Interface |
| HCI | Host Command Interface |
| IDCT | Inverse Discrete Cosine Transform |
| IO | Input/Output |
| ISDN | Integrated Services Digital Network |

| | |
|---|---|
| ISM | Industry, Science, Medicine |
| ISR | Interrupt Service Routine |
| JPEG | Joint Pictures Experts Group |
| LED | Light Emitting Diode |
| MAD | Mean Absolute Difference |
| MC | Motion Compensation |
| MD | Motion Displacement |
| ME | Motion Estimation |
| MMX | Multimedia Extension |
| MPEG | Moving Pictures Experts Group |
| MSD | Mean Squared Difference |
| MV | Motion Vector |
| PAL | Programmable Array Logic |
| PC | Personal Computer |
| PHODS | Parallel Hierarchical One Dimensional Search (Algorithm) |
| PMD | Previous Motion Displacement |
| PNG | Portable Networks Graphics |
| PPI | Parallel Peripheral Interface |
| RAM | Random Access Memory |
| RGB | Rot, Grün, Blau-Werte |
| RLC | Run/Length-Code |
| ROM | Read Only Memory |
| SFR | Special Function Register |
| SMD | Surface Mount Device |
| SPI | Serial Peripheral Interface |
| SPORT | Serial Port; schnelles, serielles, synchrones Interface |
| TSS | Three Step Search (Algorithm) |
| UART | Universal Asynchronous Receiver Transmitter |
| USB | Universal Serial Bus |
| VLC | Variable Length Code |
| VLI | Variable Length Integer |
| WLAN | Wireless Local Area Network |
| XRAM | Extension RAM |
| Y | Luminanz |
| ZZ | Zic-Zac |

# Kapitel 1

# Einleitung

Bildbearbeitung und Bildverarbeitung haben in den letzten Jahren stark an Bedeutung gewonnen. Dieser Trend wird sich auch in nächster Zeit nicht ändern. Die meisten modernen Handys sind mit einer Digitalkamera ausgestattet. Hochauflösende Digitalkameras erobern den Fotomarkt im Heimbereich aber auch immer mehr im Bereich professioneller Fotografie sowie in der Medizin. Sowohl autonome als auch personalbesetzte Überwachungssysteme werden mit Digitaltechnik ausgestattet und ersetzten die analogen Systeme. Neue Möglichkeiten bei Überwachungssystemen werden angedacht und schon teilweise verwendet, die mit Analogsystemen nicht denkbar gewesen wären.

Stehende und bewegte Bilder haben den Menschen schon immer fasziniert. So haben schon sehr früh Menschen wichtige Ereignisse in Höhlenmalereien festgehalten und seitdem hat sich der künstlerische Bereich stetig weiter entwickelt. Seit es die Technik erlaubt, hat der Mensch seine Umgebung (ob natürliche oder gestellte Szenen) abgebildet, um die Bilder aufzuheben, sie mit anderen Menschen zu teilen, oder gar um sich seinen Lebensunterhalt damit zu verdienen. Im Folgenden finden sich die wichtigsten Entwicklungsschritte, die zur heutigen Videotechnik geführt haben.

In den fünfziger Jahren des neunzehnten Jahrhunderts wurde mit der Erfindung der Papierfotografie ein völlig neuer und rasant wachsender Wirtschaftssektor erschlossen. Die wohlhabenden Bürger akzeptierten die Photographie sehr rasch als eine neue Kunstform und als Gedächtnisstütze, um sich vergangene, bewegende Erlebnisse wieder ins Gedächtnis zu rufen. Um die Jahrhundertwende wurden die ersten Kinofilme vorgestellt und bald darauf gab es in jeder größeren Stadt der westlichen Welt ein Kino. Etwa um dieselbe Zeit wurden die elektromagnetischen Wellen entdeckt und über ihre wirtschaftliche bzw. militärische Nutzung nachgedacht. Noch während des ersten Weltkrieges wurde Funksprechverkehr als Anwendung vorgestellt und schon in den frühen zwanziger Jahren hatten viele wohlhabende Familien einen Radioempfänger im Wohnzimmer stehen. Um Nachrichten und Propaganda auch wirklich in jeden Haushalt in Bild und Ton übertragen zu können, unternahmen in der Zwischenkriegszeit mehrere europäische Wissenschaftler große Anstrengungen. Kurz vor dem Beginn des zweiten Weltkrieges wurde schließlich der Fernseher vorgestellt und 1935 nahm in Deutschland der erste regelmäßige Programmdienst der Welt seinen Betrieb auf.

In den fünfziger Jahren wurde in den USA der erste serienmäßig produzierte Farbfernseher vorgestellt, der sich allerdings erst zehn Jahre später durchsetzen sollte. Erst in

den sechziger Jahren wurde das Farbfernsehen zu dem am schnellsten wachsenden Zweig der Konsumgüterindustrie.

Durch die Etablierung vieler Radio- und Fernsehsendern und die Qualitätssteigerung von Audio und Video wurde der Ruf nach Aufzeichnungsmöglichkeiten laut. 1962 kam die kleine und einfach zu bedienende Audiokassette auf den Markt, nach deren Vorbild schließlich 1975 die Videokassette entwickelt wurde. Beide Systeme haben jedoch den Nachteil, dass sie nur eine begrenzte Anzahl an Abspielungen und Wiederaufnahmen verkraften und ihre Daten magnetisch abgespeichert werden, was zu Datenverlust durch Abnützung mit der Zeit führt. Den Nachteil des Verlustes von Wiedergabequalität hat die 1981 auf den Markt gebrachte CD nicht mehr. Sie hat innerhalb weniger Jahre die Schallplatte und Audiokassette fast völlig vom Markt verdrängt und ihr Pendant am Videosektor, die 1999 vorgestellte DVD, ist gerade dabei, die Videokassette zu verdrängen.

Eine sehr detaillierte Darstellung der technischen Entwicklung findet der interessierte Leser in [PRO97-1] bis [PRO97-5].

Der Preisverfall und die Leistungssteigerung bei heutigen Computersystemen und die dadurch bedingte rasante Verbreitung war schließlich der letzte Auslöser für das Anwachsen von Bild- und Videodaten. Jeder herkömmliche PC besitzt eine Soundkarte, die es ermöglicht Audiodaten aufzunehmen. Es gibt eine Fülle von Softwareprodukten, die die Bearbeitung von Audio- und Videodaten zum Kinderspiel machen und moderne Fernseh- oder Kinofilme werden nicht mehr ohne digitale Nachbearbeitung hergestellt. Komplett animierte Filme, bei deren Herstellung keine Kamera vonnöten war, wurden schon produziert und sehr erfolgreich vermarktet. Die meisten Fernseh- oder Videofilme bekommt man gegen eine geringe Gebühr leicht im Internet. Viele Computer besitzen auch schon Videokarten, die das Überspielen von Videos sowohl von der terrestrischen Antenne, als auch von Videokameras an den PC ermöglichen.

Die Bereiche Aufzeichnungs- und Speichertechnik bei Bildern und Videos haben vor ein paar Jahren mit der Einführung von Speicherchips und Minifestplatten mit einer Kapazität von mehreren hundert MByte einen wichtigen Entwicklungsschritt vollzogen. Weitere Bereiche, die noch hohes Entwicklungspotential für die Zukunft aufweisen, sind Miniaturisierung, weitere Steigerung der Qualität (Auflösung, Farbtreue, Minilinsen, …) und autonome Bildverarbeitung bzw. Objekterkennung im intelligenten Sensor und in der Medizin. Dies wird die Entwicklung neuer, bzw. effizienterer Algorithmen erfordern.

Der einzige Bereich, der in punkto Leistungssteigerung mit der Hardware zur Bildbearbeitung nicht mithalten kann, ist die Kommunikationstechnik. Dies hat einen sehr einfachen Grund: die enormen Investitionskosten, die erforderlich wären, um alle paar Jahre neue Kabel von Haushalt zu Haushalt zu verlegen, bzw. der Installationsaufwand in Gebäuden selbst. Aus diesem Grund setzt die Kommunikationstechnik immer mehr auf drahtlose Übertragungstechnik.

In dieser Diplomarbeit soll eine Plattform zur Bildbearbeitung mit zwei wesentlichen Schwerpunkten entworfen und realisiert werden. Einerseits Miniaturisierung gepaart mit hoher Systemperformance hardwareseitig und andererseits die Entwicklung eines effizienten Videokompressionsalgorithmus mit Schwerpunkt Reduzierung der zu übertragenden Daten bei Live-Übertragung softwareseitig.

Beide Gebiete haben ihre eigenen Herausforderungen und Schwierigkeiten. So wirkt sich eine hohe Systemperformance meist negativ auf den Stromverbrauch, sowie den Platzbedarf aus. Es gibt bereits Softwarelösungen zur Videokompression (aber auch Hardwarelösungen), welche allerdings nur bedingt für die Echtzeitverarbeitung geeignet sind.

Da es sich bei der oben beschriebenen Plattform zwar um ein eigenständiges System handelt, jedoch auch die Möglichkeit gegeben sein soll, sie als intelligenten Zusatzsensor in ein bestehendes System zu integrieren, beschäftigt sich ein nicht unwesentlicher Teil dieser Arbeit mit drahtgebundenen sowie drahtlosen Übertragungssystemen zur Anbindung an ein Bussystem sowie zur drahtlosen Datenübertragung.

Nachfolgend werden die einzelnen Teilaspekte nach Kapiteln unterteilt kurz vorgestellt.

Kapitel 2 des vorliegenden Dokuments beschäftigt sich detailliert mit Grundlagen der Bild- und Videokompression. Auf eine genaue Darstellung anderer Grundlagen – wie beispielsweise für Funksysteme oder Embedded Systems – wurde verzichtet, da die Darstellung dieser Grundlagen den Umfang der vorliegenden Arbeit sprengen würde und somit dieses Wissen vorausgesetzt bzw. im Kapitel 3 kurz darauf eingegangen wird. Der interessierte Leser findet Details zur drahtlosen Übertragungstechnik beispielsweise in [WEI98] und [GEI96], zu Bussystemen in [DIE94] und [LIN00] und zu Embedded Systems in [LAB99] und [BER01].

In Kapitel 3 folgt eine detaillierte Problemanalyse zur Aufgabenstellung. Es werden die Randbedingungen definiert und darauf aufbauend auf die Anforderungen an das System eingegangen. Die Vor- und Nachteile bei der Auswahl der einzelnen Kernkomponenten werden abgewogen und die aufgrund dieser Analyse ausgewählten Komponenten vorgestellt. Aus der Problemanalyse ist bereits ein allgemeines Konzept der Plattform zur digitalen Bildbearbeitung autonomer Kleinstroboter abzuleiten. Dieses wird ebenfalls in Kapitel 3 näher beschrieben.

Kapitel 4 ist der Entwicklung des Videokompressionsalgorithmus gewidmet. Es wird hier auf Gemeinsamkeiten und Unterschiede zu vorhandenen Standards wie MPEG oder H.261 eingegangen.

Die Grundfunktionen und Schnittstellen der realisierten Hardware werden in Kapitel 5 erläutert.

Die Vielfalt der Einsatzmöglichkeiten und eventuelle Erweiterungen aufgrund des modularen Aufbaus werden letztlich in Kapitel 6 herausgearbeitet.

# Kapitel 2

# Grundlagen der Bild- und Videokompression

Gerade in letzter Zeit nehmen Bilder und Videos einen immer höheren Stellenwert bei den Konsumenten ein und deren Erstellung, Bearbeitung und Übertragung wird von Jahr zu Jahr wichtiger und kritischer. Während die Bandbreite bei Datenübertragung von einem System zum Anderen kontinuierlich steigt, entwickelt sich die Verarbeitungsgeschwindigkeit moderner Systeme noch viel schneller und die Übertragungsgeschwindigkeit wird der Verarbeitungsgeschwindigkeit auch in Zukunft um mehrere Größenordnungen hinterherhinken. Es gibt nur eins, dass noch schneller steigt als die Datenverarbeitungsgeschwindigkeit moderner Prozessoren, nämlich die Menge der zu verarbeitenden Daten. Speicherplatz ist bei modernen PCs meist ausreichend vorhanden, in mobilen Geräten jedoch beschränkt und teuer. Um die immer größer werdenden Mengen an Bild- und Videodaten überhaupt auf solchen Geräten speichern, bzw. dorthin übertragen zu können, sind Bild- und Videokompression in der heutigen Zeit nicht mehr wegzudenken.

## 2.1 Bildkompression, räumliche Redundanz

Bildkompressionsalgorithmen werden grundsätzlich in verlustlose und verlustbehaftete eingeteilt. Verlustlose Verfahren wie PNG oder GIF erreichen einen Kompressionsfaktor von bis zu 5, ohne jedoch die Bildqualität zu beeinflussen. Verlustbehaftete Verfahren, deren bekanntester Vertreter JPEG ist, erreichen Kompressionsfaktoren von bis zu 20 oder 50 wobei die Bildqualität frei wählbar ist.

Ich möchte hier nur das JPEG Verfahren näher beschreiben, da die Einzelbildkompression bei Videokompressionsalgorithmen ähnlich funktioniert. Verlustlose Verfahren spielen bei Videos aufgrund ihres geringen Kompressionsfaktors keine Rolle.

Die folgenden Abschnitte behandeln die Vorgehensweise von Einzelbildkompressionsalgorithmen.

Das Einzelbild wird in Pixelblöcke der Größe 8x8 Pixel aufgeteilt, die dann weiterverarbeitet werden. Der erste Verarbeitungsschritt ist die Umwandlung in den $Yc_bc_r$ – Farbraum, anschließend werden die Chrominanzwerte gemittelt. Die verbleibenden

Luminanz- und Chrominanz-Matrizen werden dann mittels der Diskreten Kosinustransformation DCT in den Frequenzraum transformiert. Die Frequenzanteile werden quantisiert, lauflängenkodiert und entropiekodiert.

### 2.1.1 Farbraumkonversion

Digital gespeicherte Bilder verwenden grundsätzlich zwei Arten der Speicherung, Pixel- und Vektorformate. Der überwiegende Anteil an Bildern ist in einem Pixelformat gespeichert, welches das Bild in Zeilen und Spalten aufteilt und für jeden Pixel einen Farbwert speichert. Normalerweise wird eine 24-Bit Farbpalette im RGB Raum verwendet, die für die Farbanteile Rot, Grün und Blau jeweils einen 8-Bit Wert enthält. Damit lassen sich insgesamt 16.777.216 Farben unterscheiden. Es hat sich jedoch herausgestellt, dass das menschliche Auge Helligkeit besser auflösen kann als Farbe. Diesem Umstand trägt der $YC_bC_r$ – Farbraum Rechnung.

| | |
|---|---|
| Y (Luminanz): | Grundhelligkeit |
| $C_b$ (Chrominanz): | Abweichung der Farbe vom Mittelwert Grau zu Blau |
| $C_r$ (Chrominanz): | Abweichung der Farbe vom Mittelwert Grau zu Rot |

Um einen Farbwert vom RGB-Raum in den $YC_bC_r$-Raum zu konvertieren, benötigt man folgende Konvertierungsmatrix:

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

**Gl. 2.1** Farbraumkonvertierung RGB → $YC_bC_r$

Für die Rückkonversion benötigt man folgende Konvertierungsmatrix:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.0 & 0.0 & 1.402 \\ 1.0 & -0.34414 & -0.71414 \\ 1.0 & 1.772 & 0.0 \end{pmatrix} \cdot \begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix}$$

**Gl. 2.2** Farbraumkonvertierung $YC_bC_r$ → RGB

### 2.1.2 Downsampling

Da das menschliche Auge Unterschiede in der Helligkeit viel besser auflösen kann als Farbunterschiede, kann man Speicherplatz einsparen, indem man Chrominanzwerte nicht für jeden Pixel extra speichert, sondern ein kleines Gebiet von Pixeln zusammenfasst und die Chrominanzwerte mittelt. Üblicherweise ist dieses Gebiet 2x2 Pixel groß. Das entstandene Speicherformat trägt den Namen 4-1-1 Format, da vier Lumi-

nanz-Werte, ein Chrominanz-blau- und ein Chrominanz-rot-Wert gemeinsam gespeichert werden und ein Feld von vier Pixel charakterisieren.

## 2.1.3 Transformationskodierung

Die 64 Werte im 8x8 Pixelblock werden als diskretes Signal aufgefasst. Sollten die Dimensionen des Bildes nicht durch 8 teilbar sein, werden für die Berechnung am linken, bzw. am unteren Rand Pixel angefügt (Bild spiegeln und fortsetzen). Der JPEG Algorithmus verwendet die DCT als Transformation, der neuere JPEG2000 Algorithmus die diskrete Wavelettransformation DWT. Beide sind mit der bekannten Fouriertransformation verwandt und bis auf ihren diskreten Charakter verlustlos. Die Kosinustransformation liefert die vertikalen und horizontalen Frequenzanteile des Signals während die Wavelettransformation keine harmonischen Schwingungen, sondern so genannte Wavelets als Grundfunktionen hat und das Signal in seine Waveletanteile zerlegt. Während die Koeffizienten bei der DCT Amplitude und Frequenz der Kosinusanteile darstellen, werden bei der DWT drei Anteile verwendet: Amplitude, Dauer und Zeitverschiebung des Wavelets.

### 2.1.3.1 Diskrete Kosinustransformation

Der Wert mit der horizontalen und vertikalen Frequenz von 0 wird als Gleichstromkoeffizient (DC) bezeichnet, die restlichen 63 Werte als Wechselstromkoeffizienten (AC). Im DC-Koeffizient ist die meiste Information über den Pixelblock enthalten und er ist auch mit Abstand der numerisch größte Wert, da er der Grundhelligkeit bzw. Grundchrominanz im Block entspricht. Die AC-Koeffizienten repräsentieren die Farbänderungen und Kanten, die jedoch bei natürlichen Bildern eher selten auftreten. Wenn man die Koeffizienten wieder in einer 8x8 Matrix anordnet, so sieht man, dass die Koeffizienten von links oben nach rechts unten immer kleiner werden, bzw. immer näher zu Null gehen.

Die DCT ist verlustfrei, das bedeutet IDCT(DCT(x)) = x. In der Praxis gelingt dies jedoch nicht ganz. Heutige Rechner können bedingt durch ihren Aufbau Funktionen wie beispielsweise den Kosinus nur mit einer bestimmten Genauigkeit berechnen. Dadurch ergeben sich Abweichungen der errechneten von den mathematisch korrekten Werten. Diese Abweichungen finden sich im Bild als Informationsverlust wieder. Um eine gewisse Bildqualität garantieren zu können, hat die JPEG Gruppe nicht die Implementierung der DCT und IDCT standardisiert, sondern deren maximale Abweichung von den mathematisch korrekten Werten ([ITU91]).

$$F(u,v) = \frac{1}{4} C(u) C(v) \left[ \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos\left( \frac{(2x+1)u\pi}{16} \right) \cos\left( \frac{(2y+1)v\pi}{16} \right) \right]$$

**Gl. 2.3** Diskrete Kosinustransformation

$$C(u), C(v) = \left\{ \begin{array}{c} \frac{1}{\sqrt{2}}, \; für\, u, v = 0 \\ 1, sonst \end{array} \right\}$$

**Gl. 2.4** Gewichtungskoeffizienten der DCT

$$f(x,y) = \frac{1}{4}\left[ \sum_{u=0}^{7} \sum_{v=0}^{7} C(u)C(v)F(u,v)\cos\left( \frac{(2x+1)u\pi}{16} \right)\cos\left( \frac{(2y+1)v\pi}{16} \right) \right]$$

**Gl. 2.5** Inverse Diskrete Kosinustransformation

$$C(u), C(v) = \left\{ \begin{array}{c} \frac{1}{\sqrt{2}}, \; für\, u, v = 0 \\ 1, sonst \end{array} \right\}$$

**Gl. 2.6** Gewichtungskoeffizienten der IDCT

### 2.1.3.2 Diskrete Wavelettransformation

Wavelet bedeutet soviel wie „kleine Welle“. Die Wavelets bilden eine Basis für eine Transformation, d.h. ein Signal lässt sich als Linearkombination von Wavelets darstellen. Abbildung 2.1 zeigt beispielhaft die Zerlegung der Funktion F(x), wobei als Basis ein sehr einfaches Wavelet, das Haar-Wavelet H(x) dient.

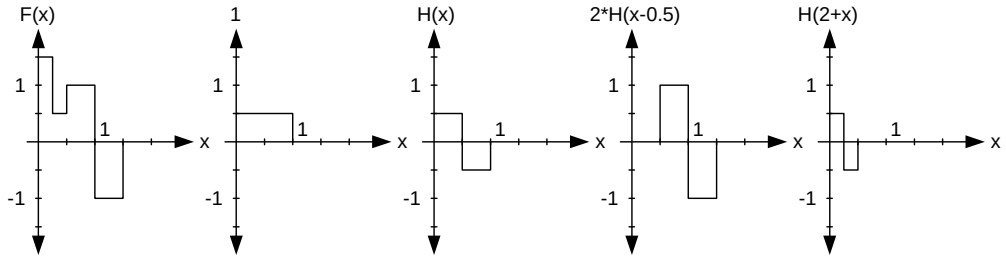


**Abbildung 2.1** Darstellung der Funktion F(x) als Linearkombination des Haar-Wavelets H(x)

F(x) = 0.5 + H(x) + 2H(x-0.5) + H(2x)

Die Wavelettransformation lässt sich als digitale Filterung interpretieren. In der digitalen Signalverarbeitung wird eine Filterung erreicht, indem man die Signalfolge mit den Filterkoeffizienten faltet. Gl. 2.7 zeigt die allgemeine Formel einer diskreten Faltungssumme.

$$y(n) = \sum_{k=0}^{\infty} x(n-k)h(k)$$

**Gl. 2.7** Allgemeine Diskrete Faltungssumme

Man kann zeigen, dass sich die Dekomposition eines Signals bei der DWT auf eine Faltung des Signals mit den Wavelet-Koeffizienten reduzieren lässt. Der Anwender muss dabei die Gestalt der Wavelets nicht kennen. Gl. 2.8 und Gl. 2.9 geben an, wie sich aus den Scaling- und Wavelet-Koeffizientensätzen die Skalierungs- und Waveletfunktionen errechnen lassen.

$$c_{j-1}(m) = \sum_{m} h(m-2k)c_j(k)$$

**Gl. 2.8** Scaling Koeffizienten bei der Dekomposition

$$d_{j-1}(m) = \sum_{m} g(m-2k)d_j(k)$$

**Gl. 2.9** Wavelet Koeffizienten bei der Dekomposition

Die Gleichungen zeigen, wie die Wavelet- und Scaling-Koeffizienten der verschiedenen Betrachtungsebenen durch Faltung der Koeffizienten der j-ten Stufe mit den Filterkoeffizienten und einem anschließenden Downsampling von 2 (d.h. nur jeder zweite Wert wird berücksichtigt) zu den Koeffizienten der (j-1)-ten Stufe – der nächst gröberen Stufe – führen. Die Filterkoeffizienten für h(n) entsprechen einem Tiefpass und g(n) einem Hochpass. Beim Haar-Wavelet wird der Tiefpass aus der Folge h = [1/sqrt(2) 1/sqrt(2)] und der Hochpass aus der Folge g = [1/sqrt(2) -1/sqrt(2)] erzeugt.

Führt man diesen Prozess der Filterung mit anschließender Dezimierung über mehreren Iterationen, erhält man die in Abbildung 2.2 gezeigte Dekomposition.

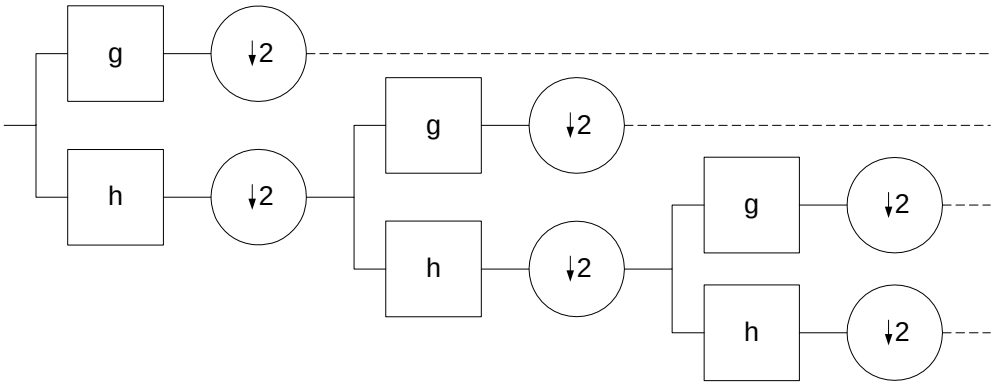


**Abbildung 2.2** Dekomposition mittels einer Filterbank

Für die Rekonstruktion muss die j-te Auflösungsstufe der Scaling- und Wavelet-Funktionen zurück zur nächst detailreicheren Auflösung – der (j+1)-ten Stufe – gebracht werden. Gl. 2.10 zeigt die Rekonstruktionsformel.

$$c_{j+1}(k) = \sum_m g(k-2m)d_j(m) + \sum_m h(k-2m)c_j(m)$$

**Gl. 2.10** Synthese der Originalwerte

Bei der Rekonstruktion werden demnach die beiden Teilsignale niedrigerer Auflösung einem Upsampling von 2 unterzogen, d. h. an jeder 2. Stelle wird eine Null eingefügt und anschließend wieder mit den Rekonstruktionsfiltern, die nicht identisch mit den Dekompositionsfiltern sein müssen, es aber meistens sind, gefiltert.

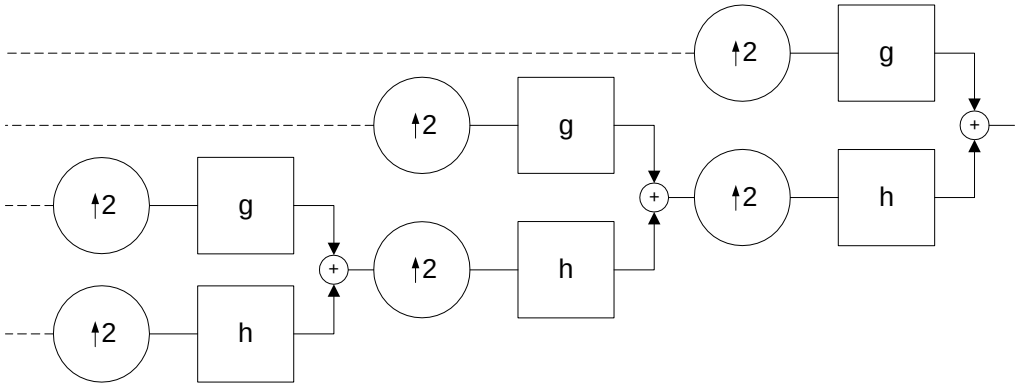


**Abbildung 2.3** Synthese des Ursprungssignals mittels einer Filterbank

Die Filterbank-Implementierung gewährleistet eine schnelle numerische Realisierung der DWT. Dadurch lassen sich ähnlich wie bei der FFT schnelle Algorithmen für die Berechnungen der Koeffizienten implementieren.

Bei der 2D-DWT wird – genau wie bei der 2D-DCT – die eindimensionale Transformation zuerst bei jeder Zeile und anschließend bei jeder Spalte angewendet.

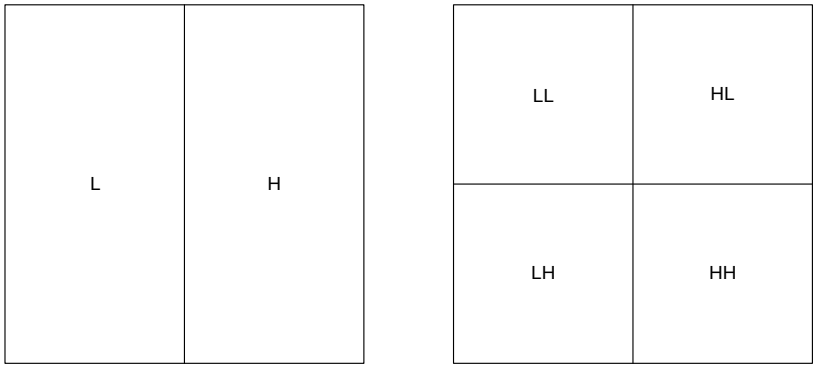


**Abbildung 2.4** Dekompositionsschritte der DWT bei 2-dimensionale Datensätzen

Abbildung 2.4 zeigt schematisch die Dekomposition, wobei L für tiefpassgefilterte (Lowpass) und H für hochpassgefilterte Bereiche (Highpass) steht. Dementsprechend bedeutet HL, dass zuerst die Zeilen hochpassgefiltert und dann die Spalten tiefpassgefiltert wurden. Das rechte Bild in Abbildung 2.4 zeigt die Bereiche nach der ersten Iteration der Dekomposition. Wie im eindimensionalen Fall werden auch hier die weiteren Iterationen nur im tiefpassgefilterten Bereich durchgeführt, also in LL. In der Literatur ist es üblich, den gefilterten Bereichen die Iterationsnummer hintenanzustellen. Abbildung 2.5 zeigt dies für den Fall der abgeschlossenen dritten Iteration.

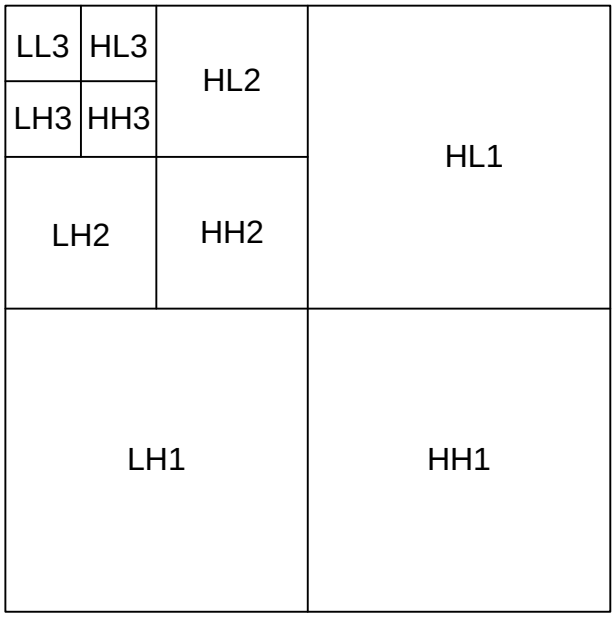


**Abbildung 2.5** Dekomposition von 2D-Daten nach der 3. Iteration

Die Wavelettransformation kann verlustlos oder verlustbehaftet ausgeführt werden. Tabelle 2.1 bis Tabelle 2.4 zeigen gebräuchliche Filterkoeffizienten für beide Varianten. Die unterschiedliche Dimension des Hochpass- und des Tiefpassfilters bewirkt, dass beim Downsampling keine Information verloren geht. Die Erklärung liegt darin, dass durch die unterschiedliche Dimension die Signale eine Zeitverschiebung zueinander erfahren und somit beim Downsampling immer verschiedene Werte verwendet werden.

| Analyse Filterkoeffizienten | | |
|---|---|---|
| | Tiefpass-Filter $h_L(i)$ | Hochpass-Filter $h_H(i)$ |
| 0 | 0.6029490182363579 | 1.115087052456994 |
| ± 1 | 0.2668641184428723 | -0.59127177631142470 |
| ± 2 | -0.07822326652898785 | -0.05754352622849957 |
| ± 3 | -0.01686411844287495 | 0.09127176311424948 |
| ± 4 | 0.02674875741080976 | |

**Tabelle 2.1** Daubechies 9/7 Analyse Filterkoeffizienten (verlustbehaftet)

| Synthese Filterkoeffizienten | | |
|---|---|---|
| | Tiefpass-Filter $g_L(i)$ | Hochpass -Filter $g_H(i)$ |
| 0 | 1.115087052456994 | 0.6029490182363579 |
| ± 1 | -0.59127177631142470 | 0.2668641184428723 |
| ± 2 | -0.05754352622849957 | -0.07822326652898785 |
| ± 3 | 0.09127176311424948 | -0.01686411844287495 |
| ± 4 | | 0.02674875741080976 |

**Tabelle 2.2** Daubechies 9/7 Synthese Filterkoeffizienten (verlustbehaftet)

| Analyse Filterkoeffizienten | | |
|---|---|---|
| | Tiefpass-Filter $h_L(i)$ | Hochpass -Filter $h_H(i)$ |
| 0 | 6/8 | 1 |
| ± 1 | 2/8 | -1/2 |
| ± 2 | -1/8 | |

**Tabelle 2.3** Daubechies 5/3 Analyse Filterkoeffizienten (verlustlos)

| Synthese Filterkoeffizienten | | |
|---|---|---|
| | Tiefpass-Filter $g_L(i)$ | Hochpass -Filter $g_H(i)$ |
| 0 | 1 | 6/8 |
| ± 1 | -1/2 | 2/8 |
| ± 2 | | -1/8 |

**Tabelle 2.4** Daubechies 5/3 Synthese Filterkoeffizienten (verlustlos)

### 2.1.4 Quantisierung

Die Quantisierung ist der eigentliche verlustbehaftete Teil der Bildkompression. Dabei werden die 64 Koeffizienten der DCT bzw. DWT, wie in Gl. 2.11 gezeigt, durch jeweils eigene Quantisierungsfaktoren dividiert und das Ergebnis auf die nächste ganze Zahl gerundet. Da die höherfrequenten Bildanteile vom Auge nicht mehr so gut wahrgenommen werden, können die AC Koeffizienten gröber quantisiert werden, als jene niederer Frequenzen.

$$F_Q(u,v) = round\left(\frac{F(u,v)}{Q(u,v)}\right)$$

**Gl. 2.11** Quantisierung

Die Bildqualität lässt sich über einen Qualitätsfaktor q, der von 1 bis 100 gewählt werden kann, einstellen. Die tatsächliche Quantisierungsmatrix ergibt sich je nach q folgendermaßen:

$$Q(u,v) = \begin{cases} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} & q = 100 \\ round\left( Q(u,v)_{orig} \left( \frac{100-q}{50} \right) \right) & 50 \le q < 100 \\ round\left( Q(u,v)_{orig} \frac{50}{q} \right) & q < 50 \end{cases}$$

**Gl. 2.12** Berechung der Quantisierungsmatrix

Q(u,v) ist die tatsächlich verwendete Quantisierungsmatrix und $Q(u,v)_{orig}$ die Matrix aus Abbildung 2.6 oder Abbildung 2.7. Gl. 2.13 zeigt die Umkehrung der Quantisierung, es handelt sich dabei um eine einfache Multiplikation des Wertes mit dem jeweiligen Eintrag in der Quantisierungsmatrix.

$$F'(u,v) = F_Q(u,v) \cdot Q(u,v)$$

**Gl. 2.13** Umkehrung der Quantisierung

Hierbei wird deutlich, dass der Originalwert nicht rekonstruiert werden kann.

Die Quantisierung bildet jedoch die Grundlage für die effizienten Kodierungsmethoden der folgenden beiden Abschnitte. Die Quantisierungsmatritzen sind im Prinzip frei wählbar, das JPEG Komitee schlägt jedoch die beiden Standard-Quantisierungsmatritzen in Abbildung 2.6 und Abbildung 2.7 vor.

$$\begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 59 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 34 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 67 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix}$$

**Abbildung 2.6** Standard-Quantisierungsmatrix Luminanz

$$\begin{pmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{pmatrix}$$

**Abbildung 2.7** Standard-Quantisierungsmatrix Chrominanz

### 2.1.5 Zic-Zac-Scan

Vor der Kodierung werden die Werte durch einen so genannten Zic-Zac-Scan geschickt umgeordnet. Sieht man sich die quantisierten Koeffizienten in der Matrix an, so bemerkt man, dass im rechten unteren Teil nur Nullen stehen, während die einzigen von Null verschiedenen Werte in der linken oberen Ecke der Matrix stehen, wobei der DC-Koeffizient mit Abstand den numerisch größten Wert darstellt. Das Zic-Zac-Scan Verfahren versucht jetzt, wie in Abbildung 2.8 gezeigt, die Koeffizienten nach ihrem numerischen Wert zu ordnen.

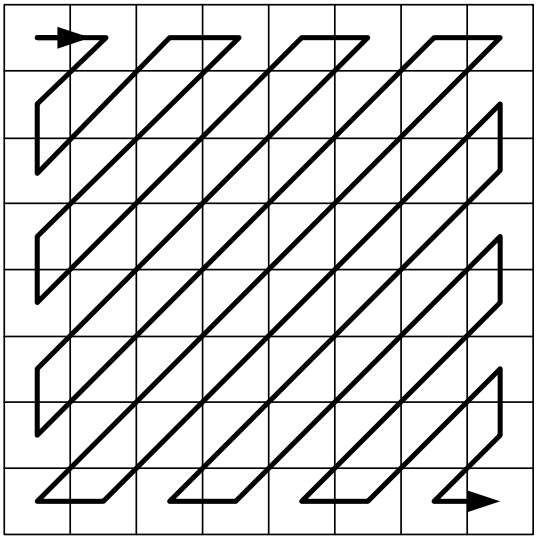

**Abbildung 2.8** Zic-Zac-Scan Wertereihung

### 2.1.6 Lauflängenkodierung

Die umgeordneten quantisierten Koeffizienten werden nun lauflängenkodiert. Die Lauflängenkodierung ordnet jedem AC Wert zwei Symbole wie folgt zu.

| Symbol 1 | Symbol 2 |
|---|---|
| (Lauflänge, Größe) | (Wert) |

Die Lauflänge ist die Anzahl der vor dem Wert vorkommenden Nullen. Bei den Lauflängencodes für die AC Koeffizienten gibt es zwei Spezialfälle von Symbol 1:

- (0,0) **E**nd **o**f **B**lock bedeutet, dass ab jetzt bis zum Ende des Blocks nur mehr Nullen kommen
- (15,0) Dieses Symbol bedeutet, dass vor der Null 15 Nullen kommen. Es wurde eingeführt, um die maximale Lauflänge (und damit die Länge der Codierungstabellen) auf 15 begrenzen zu können. Sollte es beim Codieren dreimal hintereinander auftreten, kann man die drei Symbole durch ein EoB ersetzen, da man davon ausgehen kann, dass nach 48 Nullen auch weiterhin nur mehr Nullen folgen.

Der DC Wert wird DPCM moduliert (siehe unten) und anschließend ebenfalls kodiert, wobei Symbol 1 nur die Größe enthält, da vor dem DC Wert keine Nullen vorkommen können.

| Symbol 1 | Symbol 2 |
|---|---|
| (Größe) | (Wert) |

Der Zusammenhang zwischen Größe und Amplitude wird in Tabelle 2.5 erläutert.

| Größe | Amplitude |
|---|---|
| 0 | 0 |
| 1 | -1\|1 |
| 2 | -3,-2\|2,3 |
| 3 | -7..-4\|4..7 |
| 4 | -15..-8\|8..15 |
| 5 | -31..-16\|16..31 |
| 6 | -63..-32\|32..63 |
| 7 | -127..-64\|64..127 |
| 8 | -255..-128\|128..255 |
| 9 | -511..-256\|256..511 |
| 10 | -1023..-512\|512..1023 |
| … | |
| 15 | -32767..-16384\|16384..32676 |

**Tabelle 2.5** Zusammenhang Größe ↔ Amplitude beim VLI

Beim DC Wert kann davon ausgegangen werden, dass er sich von Block zu Block nicht wesentlich ändert, deshalb wird immer nur die Differenz zum vorhergehenden Wert übertragen. Dieses Verfahren bezeichnet man als Differential Puls Code Modulation DPCM. Symbol 1 des DPCM Wertes lässt sich aus Tabelle 2.5 ablesen, Symbol 2 entspricht dem tatsächlichen Wert. Diese Art der Darstellung inklusive Größe (Anzahl der zur Darstellung benötigten Bits) ist notwendig für die anschließende Kodie-

rung mit variabler Länge. Symbol 1 wird VLC (Variable Length Code) kodiert, Symbol 2 VLI (Variable Length Integer). Der VLC ist ein Huffmancode, der VLI ist streng genommen kein Huffmancode.

In Anhang A befinden sich die Standardcodierungstabellen, wie sie bei MPEG verwendet werden. Auf die Darstellung der Standardcodierungstabellen, die JPEG verwendet, wurde verzichtet, da sie im Projekt keine Verwendung finden.

Anmerkung: Die DCT kann Koeffizienten erzeugen, deren Wertebereich 3 Bit größer als der Ursprüngliche ist, daher hat die Tabelle für Symbol 1 12 Einträge.

### 2.1.7 Entropiekodierung

Um Kodiertabellen aufzustellen, gibt es mehrere Möglichkeiten,

- Arithmetische Kodierung und
- Huffmankodierung

sind die zwei in JPEG zur Verfügung stehenden.

Bei der arithmetischen Kodierung werden die zu kodierenden Zeichen reellen Zahlenintervallen in [0,1] zugeordnet. Je länger die zu kodierende Zeichenfolge ist, desto enger werden die Intervalle. Bei der Berechnung der Zahlenintervalle werden die Auftrittswahrscheinlichkeiten der Zeichen berücksichtigt, die dynamisch angepasst werden können. Das Kompressionsergebnis ist gut, da Zeichenreihen und nicht einzelne Zeichen kodiert werden.

Im Beispiel in Abbildung 2.9 besteht das Alphabet der Eingangsfolge aus den Zeichen a, b, c und d mit den Auftrittswahrscheinlichkeiten p(a) = 0,5; p(b) = 0,25; p(c) = p(d) = 0,125. Es ist dargestellt, wie sich die Intervalle entwickeln, wenn die Zeichenfolge **abacd** kodiert werden soll. Das Intervall für die Zeichenreihe abacd ist beispielsweise [311/1024, 312/1024 = 39/128). Die Folge kann durch jede beliebige Zahl innerhalb des Intervalls 0,0100111 und 0,0100110111 dargestellt werden. Zur Dekodierung müssen die Auftrittswahrscheinlichkeiten und die Länge der kodierten Folgen bekannt sein.

Die Idee der Huffmankodierung lehnt sich an die Idee des Morsealphabets aus Tabelle 2.7 an: Symbolen mit einer hohen Auftrittswahrscheinlichkeit werden kurze Bitfolgen zugeordnet und umgekehrt.

Die Codetabelle selbst wird unter Zuhilfenahme des so genannten Huffmanbaumes ermittelt. Das folgende Beispiel zeigt die Kompression einer Textdatei. Diese Datei besteht aus den in Tabelle 2.6 angeführten Symbolen mit den angegebenen Häufigkeiten.

| Symbol | A | B | C | D | E |
|---|---|---|---|---|---|
| Häufigkeit | 15 | 7 | 6 | 6 | 5 |

**Tabelle 2.6** Symbole und Häufigkeiten zum Beispiel Huffmancode

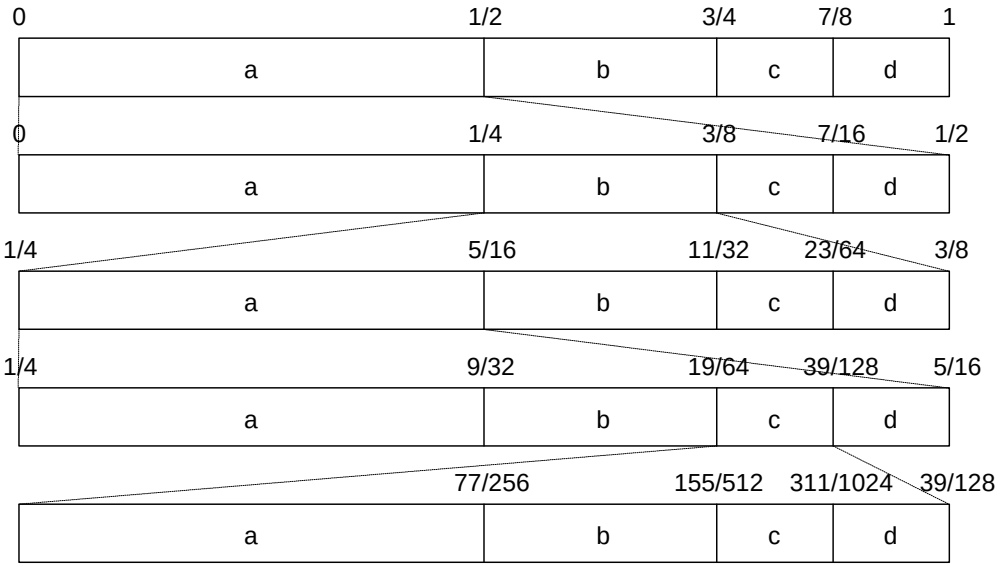


**Abbildung 2.9** Entwicklung der Intervalle zum Beispiel arithmetische Kodierung

Der Huffmanbaum wird nun nach folgenden Regeln aufgebaut:

- Die beiden freien Knoten mit der geringsten Häufigkeit werden ermittelt
- Für diese beiden Knoten wird ein Elternknoten generiert, dessen Häufigkeit sich als Summe der Häufigkeiten der Kindknoten ergibt. Die Kindknoten werden aus der Liste der freien Knoten entfernt und der Elternknoten eingefügt.
- Einem der beiden Kindknoten wird eine 0 zugewiesen, dem anderen eine 1
- Die drei Schritte werden wiederholt, bis nur noch ein Knoten frei bleibt. Dieser wird als Wurzel des Baumes bezeichnet.

Um die Einträge der Codetabelle zu ermitteln, geht man von der Wurzel zum jeweiligen Blatt und setzt die Bits hintereinander. Bei diesem Beispiel mit dem Huffmanbaum aus Abbildung 2.10 ergibt sich die Codetabelle Tabelle 2.8.

| Buchstabe | Morse-zeichen | Buchstabe | Morse-zeichen | Buchstabe | Morse-zeichen | Buchstabe | Morse-zeichen |
|---|---|---|---|---|---|---|---|
| A | •– | H | •••• | O | ––– | V | •••– |
| B | –••• | I | •• | P | •––• | W | •–– |
| C | –•–• | J | •––– | Q | ––•– | X | –••– |
| D | –•• | K | –•– | R | •–• | Y | –•–– |
| E | • | L | •–•• | S | ••• | Z | ––•• |
| F | ••–• | M | –– | T | – | | |
| G | ––• | N | –• | U | ••– | | |

**Tabelle 2.7** Morsecodetabelle

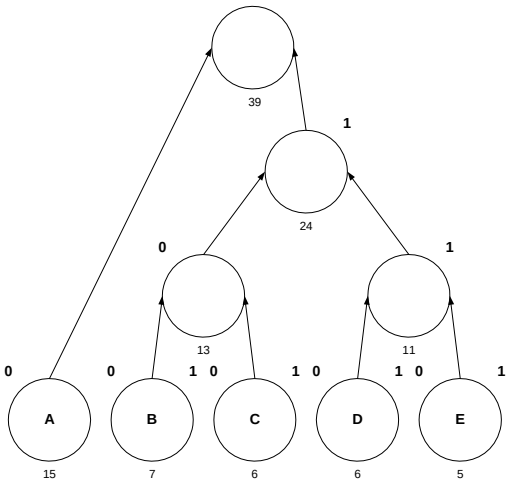


**Abbildung 2.10** Huffmanbaum zum Beispiel Huffmancode

| Symbol | Huffman Code |
|---|---|
| A | 0 |
| B | 100 |
| C | 101 |
| D | 110 |
| E | 111 |

**Tabelle 2.8** Codetabelle zum Beispiel Huffmancode

Geht man davon aus, dass die Datei als ASCII Textdatei gespeichert war, so benötigt man statt 312 Bit nur mehr 87 Bit zum Abspeichern, man muss jedoch die Huffmantabelle mitspeichern. Wie schon bei der arithmetischen Kodierung verbessert sich auch bei der Huffmankodierung das Kompressionsergebnis, wenn man Codewörter für längere Zeichenfolgen einführt.

### 2.1.8 Beispiel zur Kodierung eines Blocks

Im Folgenden werden alle Schritte der Komprimierung und Dekomprimierung an einem 8x8 Pixelblock gezeigt, wobei davon ausgegangen wird, dass die Farbtransformation abgeschlossen ist. Abbildung 2.11 zeigt die Luminanz- (oder Chrominanz-) werte, Abbildung 2.12 die Koeffizienten nach der DCT. Die Koeffizienten werden dann durch die an gleicher Position stehenden Quantisierungskoeffizienten aus Abbildung 2.6 dividiert und auf die nächste ganze Zahl gerundet, was die Koeffizienten in Abbildung 2.13 ergibt. Die Werte nach der inversen Quantisierung und IDCT zeigen Abbildung 2.14 und Abbildung 2.15. Man beachte, dass sich die rekonstruierten Werte von den Ursprünglichen nicht wesentlich unterscheiden, obwohl 64 Byte durch die in Abbildung 2.16 gezeigten 32 Bit dargestellt wurden.

$$\begin{pmatrix} 139 & 144 & 149 & 153 & 155 & 155 & 155 & 155 \\ 144 & 151 & 153 & 156 & 159 & 156 & 156 & 156 \\ 150 & 155 & 160 & 163 & 158 & 156 & 156 & 156 \\ 159 & 161 & 162 & 160 & 160 & 159 & 159 & 159 \\ 159 & 160 & 161 & 162 & 162 & 155 & 155 & 155 \\ 161 & 161 & 161 & 161 & 160 & 157 & 157 & 157 \\ 162 & 162 & 161 & 163 & 162 & 157 & 157 & 157 \\ 162 & 162 & 161 & 161 & 163 & 158 & 158 & 158 \end{pmatrix}$$

**Abbildung 2.11** Beispielwerte eines Luminanz (Chrominanz) blocks

$$\begin{pmatrix} 1235{,}6 & -1{,}0 & -12{,}1 & -5{,}2 & 2{,}1 & -1{,}7 & -2{,}7 & 1{,}3 \\ -22{,}6 & -17{,}5 & -6{,}2 & -3{,}2 & -2{,}9 & -0{,}1 & 0{,}4 & -1{,}2 \\ -10{,}9 & -9{,}3 & -1{,}6 & 1{,}5 & 0{,}2 & -0{,}9 & -0{,}6 & -0{,}1 \\ -7{,}1 & -1{,}9 & 0{,}2 & 1{,}5 & 0{,}9 & -0{,}1 & 0{,}0 & 0{,}3 \\ -0{,}6 & -0{,}8 & 1{,}5 & 1{,}6 & -0{,}1 & -0{,}7 & 0{,}6 & 1{,}3 \\ 1{,}8 & -0{,}2 & 1{,}6 & -0{,}3 & -0{,}8 & 1{,}5 & 1{,}0 & -1{,}0 \\ -1{,}3 & -0{,}4 & -0{,}3 & -1{,}5 & -0{,}5 & 1{,}7 & 1{,}1 & -0{,}8 \\ -2{,}3 & 1{,}6 & -3{,}8 & -1{,}8 & 1{,}9 & 1{,}2 & -0{,}6 & -0{,}4 \end{pmatrix}$$

**Abbildung 2.12** Koeffizienten nach der DCT

$$\begin{pmatrix} 77 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

**Abbildung 2.13** Quantisierte und gerundete Koeffizienten

$$\begin{pmatrix} 1232 & 0 & -10 & 0 & 0 & 0 & 0 & 0 \\ -24 & -12 & 0 & 0 & 0 & 0 & 0 & 0 \\ -14 & -13 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

**Abbildung 2.14** Koeffizienten vor der IDCT

$$\begin{pmatrix} 144 & 146 & 149 & 152 & 154 & 156 & 156 & 156 \\ 148 & 150 & 152 & 154 & 156 & 156 & 156 & 156 \\ 155 & 156 & 157 & 158 & 158 & 157 & 156 & 155 \\ 160 & 161 & 161 & 162 & 161 & 159 & 157 & 155 \\ 163 & 163 & 164 & 163 & 162 & 160 & 158 & 156 \\ 163 & 164 & 164 & 164 & 162 & 160 & 158 & 157 \\ 160 & 161 & 162 & 162 & 162 & 161 & 159 & 157 \\ 158 & 159 & 161 & 161 & 162 & 161 & 159 & 157 \end{pmatrix}$$

**Abbildung 2.15** Rekonstruierte Pixelwerte

| Lauflänge/Kategorie | -/4 | 1/2 | 0/1 | 0/1 | 0/1 | 2/1 | EoB |
|---|---|---|---|---|---|---|---|
| VLD Code | 1110 1111 | 11011 01 | 00 0 | 00 0 | 00 0 | 11100 0 | 10 |

**Abbildung 2.16** Bitstrom zu obigem Beispiel

## 2.2 Bewegte Bilder, zeitliche Redundanz

Die ersten Videokompressionsverfahren nahmen auf die in den Bildern enthaltene zeitliche Redundanz noch keine Rücksicht. MJPEG (Motion JPEG) beispielsweise überträgt JPEG komprimierte Einzelbilder. Die benötigte Datenrate bei MJPEG ist aber deutlich höher als bei modernen Verfahren wie MPEG oder H.263.

Moderne Videokompressionsverfahren vergleichen aufeinander folgende Bilder, und suchen nach Bereichen, die sich nicht wesentlich geändert, dafür aber "bewegt" bzw. verschoben haben. Diese Suche wird Bewegungschätzung oder Motion Estimation genannt. Wird eine Übereinstimmung gefunden, so braucht nur der Versatz des Bereichs, der Motion Vector, übertragen zu werden. Der Dekoder kann mit Hilfe der

Motion Vectors das nächste Bild vorhersagen, er erhält ein predicted Frame. Jetzt braucht nur mehr die Differenz zwischen dem predicted Frame und dem tatsächlichen Frame übertragen zu werden. Wird die Prediction nur in eine Richtung durchgeführt, nennt man das Bild P-Frame (predicted). Bezieht sich das Differenzbild auf ein vorhergehendes und ein nachfolgendes Frame, spricht man vom B-Frame (bidirectional). P-Frames brauchen eine wesentlich geringere Datenrate als I-Frames (~10 – 30%), B-Frames ungefähr nochmals die Hälfte. I-Frames nennt man die ohne Motion Vectors, also unabhängig (independent) kodierten Bilder. Theoretisch könnte man in einem Video mit einem einzigen I-Frame am Start auskommen, praktisch geht das jedoch aus zwei Gründen nicht:

- Die Implementierung der IDCT und DCT ist nicht vorgeschrieben, es kann daher zu kleinen Fehlern kommen, die sich jedoch mit der Zeit auswirken würden
- Beim schnellen Vor- und Rücklauf müsste man alle Differenzen aufrechnen, um das aktuelle Bild zu erhalten

In den folgenden Unterkapiteln werden zuerst die Bewegungsschätzung und -kompensation, danach einige Unterschiede bei der Behandlung von Differenzbildern gegenüber unabhängigen Bildern und abschließend die wichtigsten Standards im Bereich Videokompression vorgestellt.

### 2.2.1 Bewegungsschätzung und –kompensation

Die Bewegungskompensation (Motion Compensation MC) ist ein Prozess, der Änderungen beweglicher Bildinhalte von einem Bild zum nächsten kompensiert. Die Bewegungsschätzung (Motion Estimation ME) ist jener Prozess, welcher diese Änderungen durch geeignete Algorithmen feststellt und Motion Vectors MV berechnet. Die MC ist sowohl im Dekoder als auch im Enkoder vorhanden, während die ME ist nur im Enkoder vorhanden und im Allgemeinen sehr rechenintensiv ist. Abbildung 2.17 zeigt den Aufbau eines generischen Hybrid-Enkoders mit Bewegungsschätzung, wie er bei allen gängigen Videokompressionsverfahren verwendet wird.

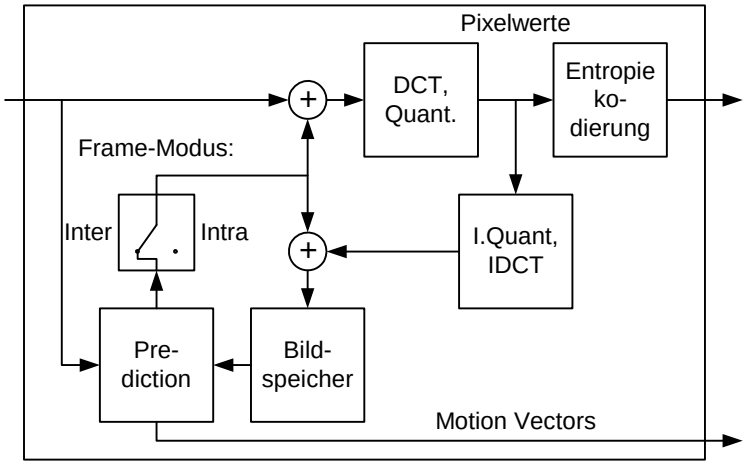


**Abbildung 2.17** Aufbau eines generischen Hybrid-Enkoders mit ME

Der Enkoder arbeitet folgendermaßen:

- Er berechnet die Differenz zwischen dem aktuellen Bild und dem geschätzten Bild. Dieses Differenzbild wird transformiert, quantisiert, entropiekodiert und an den Dekoder gesendet.
- Das Differenzbild wird nach der Quantisierung dequantisiert, invers transformiert und dann zum geschätzten Bild addiert. Dieses Bild entspricht dem des Dekoders (inkl. aller Verluste).
- Die ME errechnet aus dem rekonstruierten letzten Bild und dem aktuellen Bild die Motion Vectors.
- Die MC errechnet das geschätzte Bild aus dem rekonstruierten letzten Bild und den Motion Vectors.

Wie die Funktionsbeschreibung schon impliziert, ist der Dekoder im Enkoder vollständig enthalten. Der Dekoder ist viel einfacher und billiger zu implementieren als der Enkoder, für beide gibt es allerdings Implementierungen in Hard- und Software. Digitales Fernsehen (DVB) beispielsweise verwendet MPEG-2 zur Kompression und in jedem digitalen Satellitenreceiver ist ein eigener Chip zur Dekodierung der Daten vorhanden.

Es gibt sehr viele verschiedene Ansätze, um die Motion Vectors möglichst effizient zu berechnen. In den folgenden Abschnitten werden beispielhaft Full Search, Three Step Search, 2-D Logarithmic Search und PHODS und die Kostenfunktionen MAD und MSD behandelt. Eine erschöpfende Diskussion der verschiedenen Algorithmen und Kostenfunktionen findet sich in [MPG85] und [KUH98]. In [MPF96] werden verschiedenste Methoden der Suchalgorithmen vorgestellt und mir Referenzen versehen.

Die grundsätzliche Vorgangsweise ist bei allen Algorithmen die gleiche:

- Das Bild wird in Makroblöcke aufgeteilt
- Ein Makroblock wird ein Stück verschoben und an dieser Stelle mittels einer Kostenfunktion mit den Pixeln im anderen Frame verglichen.
- Es werden mehrere solcher Vergleiche an verschiedenen Stellen durchgeführt, der Best Match ist Ausgangspunkt für eine mögliche weitere Verfeinerung der Suche.
- Die Differenz vom Ausgangspunkt zum letzten Best Match ist der Motion Vector

Die Effizienz der Suche hängt von

- der Größe des Suchbereichs
- der Kostenfunktion und
- dem Suchalgorithmus

ab.

#### 2.2.1.1 Full Search (Block Matching BM)

Beim BM handelt es sich um den einfachsten vorstellbaren Suchalgorithmus: Der Makroblock wird an jeder Stelle im Suchfenster mit dem zweiten Frame verglichen. Der Vorteil dieser Methode ist, dass im Suchfenster **sicher** das absolute Minimum gefunden wird, der Nachteil liegt auf der Hand: es sind sehr viele Vergleiche notwendig und daher nur kleine Suchfenster möglich.

Um die Suche zu beschleunigen, also nicht jeden Vergleich durchführen zu müssen, werden folgende Annahmen getroffen:

- Es gibt im Suchbereich genau ein Minimum der Kostenfunktion
- Je näher man diesem Minimum kommt, desto kleiner wird der Wert der Kostenfunktion

Diese Annahmen können auch so interpretiert werden, dass man für die Kostenfunktion eine „Schüsselform“ erwartet. Nicht berücksichtigt werden allerdings lokale Minima, in denen der Suchalgorithmus hängen bleiben kann.

#### 2.2.1.2 Three Step Search TSS

Beim TSS Algorithmus wird der Makroblock zunächst an seiner Ursprungsposition verglichen. Dann werden in jedem Schritt acht weitere Vergleiche durchgeführt, wobei die Schrittlänge jedes Mal halbiert wird. Die Vergleiche werden an den Eckpunkten und den Kantenmittelpunkten eines Quadrates um den Suchmittelpunkt herum durchgeführt, dessen Kanten zweimal so lang wie die Schrittlänge sind. Abbildung 2.18 zeigt einen möglichen Durchlauf des TSS Algorithmus bei einem Suchfenster der Größe 12x12.

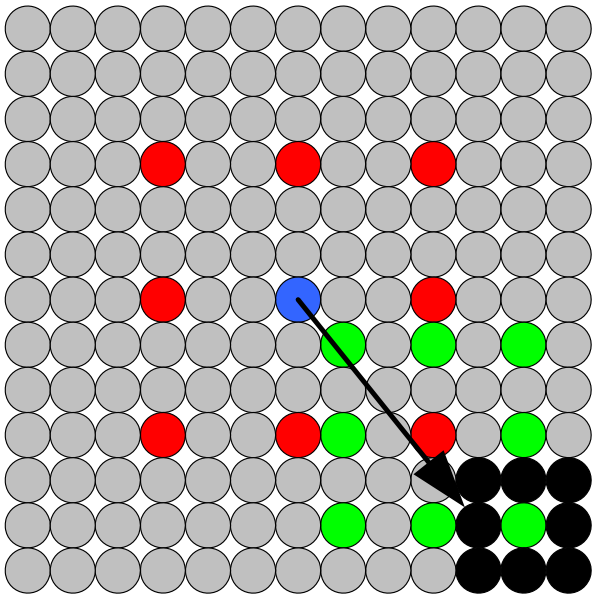

**Abbildung 2.18** möglicher Durchlauf des TSS bei einem Suchfenster von 12x12

Bei den im MPEG Algorithmus verwendeten Suchfenstern der Größe 48x48 Pixel benötigt TSS 29 mal weniger Rechenoperationen als BM.

#### 2.2.1.3 2D-Logarithmic Search Algorithm

Ähnelt dem TSS Algorithmus, teilt jedoch die einzelnen Schritte noch in zwei Phasen auf. Zusätzlich kann ein Schwellwert eingestellt werden, bei dem die Suche abbricht. Damit lassen sich Geschwindigkeit und Qualität variieren.

In der ersten Phase werden die Vergleiche an den Kantenmittelpunkten und im Zentrum durchgeführt. Nur wenn das Minimum nicht im Zentrum liegt, werden in der zweiten Phase an den benachbarten Eckpunkten Vergleiche durchgeführt. Das Minimum der sechs (mit dem Zentrum sieben) Vergleiche ist Zentrum des nächsten Schrittes, bei dem die Schrittlänge wie beim TSS halbiert wird. Abbildung 2.19 zeigt einen möglichen Durchlauf des 2D-Logarithmic Search bei einem Suchfenster der Größe 12x12. Man sieht, dass man gegenüber dem TSS noch sechs Vergleiche eingespart hat, allerdings kann es durch die geringere Anzahl an Vergleichen auch zu einem anderen Motion Vector führen wenn die Schüsselform nicht gegeben ist.

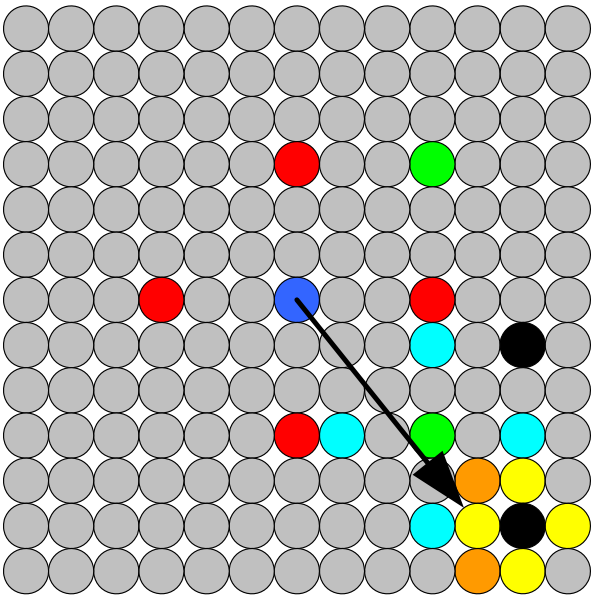

**Abbildung 2.19** möglicher Durchlauf des 2dLS bei einem Suchfenster von 12x12

#### 2.2.1.4 Parallel Hierarchical One-Dimensional Search Algorithm PHODS

PHODS ist der hier vorgestellte Algorithmus mit den wenigsten erforderlichen Vergleichen, allerdings erwartet er eine sehr ausgeprägte Schüsselform der Werte der Kostenfunktion.

- Beim ersten Schritt werden wieder die 4 Werte in den Kantenmittelpunkten des Quadrats, nicht jedoch der im Zentrum berechnet.
- In x- und y-Richtung definiert jetzt der kleinere der beiden Werte je eine Achse. Auf diesen Achsen wird jetzt unabhängig voneinander fortgefahren.
- Die Schrittweite wird mit jedem Schritt halbiert
- Es werden die Werte unterhalb/oberhalb bzw. links/rechts vom Achsenzentrum verglichen.
- Das Minimum auf jeder Achse ist Zentrum des nächsten Schrittes.

Anmerkung: Befindet sich das Minimum nicht auf einer Achse, wird es selbst nie berechnet.

Abbildung 2.20 zeigt einen möglichen Durchlauf des PHODS bei einem Suchfenster der Größe 12x12. Man sieht, dass man gegenüber dem TSS noch einmal sechs Vergleiche eingespart hat. PHODS ist sehr schnell, die gefundenen Motion Vectors entsprechen aber nicht immer den optimalen. Für den Performancegewinn während des Enkodiervorgangs bezahlt man mit zusätzlicher Bitrate beim Übertragen bzw. Speichern der Differenzbilder.

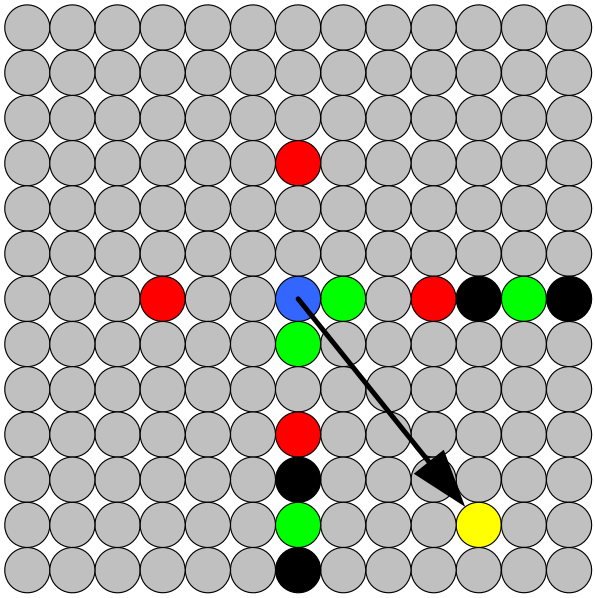

**Abbildung 2.20** möglicher Durchlauf des PHODS bei einem Suchfenster von 12x12

### 2.2.1.5 Mean Absolute Difference MAD

Die MAD (Mittlere absolute Abweichung) ist sehr einfach zu berechnen und kann auch in Hardware implementiert werden. Wie Gl. 2.14 zeigt, werden alle Pixeldifferenzen addiert und gemittelt.

$$MAD(x,y) = \frac{1}{mn} \sum_{i=-\frac{n}{2}}^{\frac{n}{2}} \sum_{j=-\frac{m}{2}}^{\frac{m}{2}} \left| F(i,j) - G(i+x, j+y) \right|$$

**Gl. 2.14** Mean Absolute Difference MAD

### 2.2.1.6 Mean Squared Difference MSD

Die MSD (Mittlere quadratische Abweichung) ist ein eher intuitives Abstandsmaß. Ihre Berechnung (Gl. 2.15) dauert aufgrund des Quadrierens länger, dafür liefert sie ein qualitativ besseres Ergebnis als die MAD.

$$MSD(x,y)=\frac{1}{mn}\sum_{i=-\frac{n}{2}}^{\frac{n}{2}}\sum_{j=-\frac{m}{2}}^{\frac{m}{2}}\left(F(i,j)-G(i+x,j+y)\right)^2$$

**Gl. 2.15** Mean Squared Difference MSD

## 2.2.2 Besonderheiten der Differenzbilder

Differenzbilder werden in einigen wesentlichen Punkten anders behandelt als unabhängig kodierte Bilder. So wird Ihnen eine andere Quantisierungsmatrix zugewiesen, es gibt die Möglichkeit Blöcke und ganze Makroblöcke in denen nur Nullwerte stehen, zu überspringen und die Kodierung der Motion Vectors muss ebenfalls erläutert werden.

### 2.2.2.1 Quantisierung nach MPEG

Anders als beim JPEG Algorithmus werden bei MPEG zwei verschiedene Abbildungsfunktionen – mit schmalem Nullband für I-Frames und mit breitem Nullband für P- und B-Frames – zur Quantisierung verwendet. Gl. 2.16 und Gl. 2.17 zeigen Quantisierung und Rekonstruktion für I-Frames, Gl. 2.18 und Gl. 2.19 für die Differenzbilder. σ(x) ist die Signumfunktion mit dem Wertebereich ±1 und 0.

$$F_Q(u,v)=\frac{16\cdot F(u,v)+\sigma(F(u,v))\cdot Q_I(u,v)\cdot q}{2\cdot Q_I(u,v)\cdot q}$$

**Gl. 2.16** Quantisierung von I-Frames nach MPEG (mit schmalem Nullband)

$$F(u,v)=\frac{2\cdot F_Q(u,v)\cdot Q_I(u,v)\cdot q}{16}$$

**Gl. 2.17** Inverse Quantisierung von I-Frames nach MPEG

$$F_Q(u,v)=\frac{16\cdot F(u,v)}{2\cdot Q_{BP}(u,v)\cdot q}$$

**Gl. 2.18** Quantisierung von P- und B-Frames nach MPEG (mit breitem Nullband)

$$F(u,v)=\frac{\left(2\cdot F_Q(u,v)+\sigma(F_Q(u,v))\right)\cdot Q_{BP}(u,v)\cdot q}{16}$$

**Gl. 2.19** Inverse Quantisierung von P- und B-Frames nach MPEG

Über den Qualitätsfaktor q kann wie bei der Einzelbildkompression die Bildqualität gesteuert werden, wobei die zulässigen Werte für q aus Tabelle 6.8 herausgelesen werden können.

Beim MPEG Standard werden, um nicht zu viele Tabellen zu erhalten, Luminanz- und Chrominanzwerten beim Quantisieren nicht unterschieden. Es gibt allerdings trotzdem zwei verschiedene Matrizen, die in Abbildung 2.21 gezeigte gilt für Blöcke in I-Frames, die für Blöcke in P- und B-Frames enthält überall den Wert 16. Auffällig ist hier, dass die Werte bei der Quantisierung der Differenzbilder nicht mit der Frequenz steigen. Da die Differenzwerte meist sowieso sehr klein sind ist dies nicht nötig, es gibt auch so nach der Quantisierung kaum höherfrequente Koeffizienten.

$$\begin{pmatrix} 8 & 16 & 19 & 22 & 26 & 27 & 29 & 34 \\ 16 & 16 & 22 & 24 & 27 & 29 & 34 & 37 \\ 19 & 22 & 26 & 27 & 29 & 34 & 34 & 38 \\ 22 & 22 & 26 & 27 & 29 & 34 & 37 & 40 \\ 22 & 26 & 27 & 29 & 32 & 35 & 40 & 48 \\ 26 & 27 & 29 & 32 & 35 & 40 & 48 & 58 \\ 26 & 27 & 29 & 34 & 38 & 46 & 56 & 69 \\ 27 & 29 & 35 & 38 & 46 & 56 & 69 & 83 \end{pmatrix}$$

**Abbildung 2.21** MPEG Standard-Quantisierungsmatrix für Intra-Kodierung $Q_I$

Die Werte in der Quantisierungsmatrix entsprechen ungefähr dem Frequenzempfinden des menschlichen Auges bei einem Abstand zum Bildschirm von sechs Mal der Bildschirmbreite und einem Bild von 320x240 Pel[1] Auflösung.

#### 2.2.2.2 Skipping Blocks and Makroblocks

Bei Differenzbildern kann es häufig vorkommen, dass in einem Block oder Makroblock nur mehr Werte gleich Null vorkommen. Diesem Umstand trägt man durch Einführung des coded_block_pattern und des macroblock_adress_increment Rechnung. Beide Werte werden als VLC übertragen. Das coded_block_pattern gibt für alle sechs Blöcke im Makroblock an, ob der Block im Bitstrom vorkommt oder nicht. Die 64 möglichen Kombinationen und VLC's sind … zu entnehmen. Da in I-Frames normalerweise keine leeren Blöcke vorkommen, wird im Bitstrom über ein Bit angegeben, ob coded_block_pattern übertragen wird.

Jeder Makroblock enthält einen Eintrag macroblock_adress_increment, der angibt, ob ein vorhergehender Makroblock übertragen wurde. Sollten alle sechs Blöcke eines Makroblocks Nullwerte enthalten, so wird das … zu entnehmende macroblock_adress_increment einfach um Eins erhöht. Ein ganzer Makroblock kann somit in einem (oder noch weniger) Bit übertragen werden. Sollten mehr als 33 Makroblö-

[1] Pel steht für picture element und wird im MPEG Standard verwendet. Die in Europa gebräuchliche Bezeichnung Pixel für picture element kommt im MPEG Standard nicht vor.

cke übersprungen werden, so wird das macroblock_escape Zeichen, das für 34 steht, sooft als nötig eingefügt.

#### 2.2.2.3 Darstellung der Motion Vectors

Jeder der bis zu vier Motion Vectors MV eines Makroblock wird in MPEG-1 durch drei Werte dargestellt: Principal Part, Residual Part und Scaling Factor. Außerdem muss angegeben werden, ob die Werte für halbe Pel oder ganze Pel gelten.

Gl. 2.20 bis Gl. 2.27 zeigen die Berechnungen der einzelnen Werte.

$$dMD_p = motion\_code \times f$$

**Gl. 2.20** Principal Part eines Motion Vectors

*motion_code* ist definiert als ganze Zahl mit Wertebereich [-16,16] und *f* ist ein Skalierungsfaktor, der sich mit

$$f = 2^{r\_size}$$

**Gl. 2.21** Skalierungsfaktor eines Motion Vectors

ergibt. Die VLC's für *motion_code* finden sich in Anhang A. f wird für Vorwärts- und Rückwärts-MV getrennt angegeben, der Wert gilt allerdings für ein ganzes Bild. Übertragen wird aber nicht *f*, sondern *f_code*, dass sich aus *r_size* mit

$$r\_size = f\_code - 1$$

**Gl. 2.22** Zusammenhang zwischen Exponent *r_size* und übertragenem Code

ergibt. Das Residuum *r* ist per Definition eine positive ganze Zahl.

$$r = \left|dMD_p\right| - \left|dMD\right|$$

**Gl. 2.23** Residual Part *r* eines Motion Vectors

zeigt, dass $|dMD_p|$ größer oder gleich $|dMD|$ sein muss. *dMD* ist der eigentliche, DPCM kodierte Motion Vector, der die Differenz zwischen aktuellem Versatz und Versatz des letzten Makroblocks angibt.

$$dMD = MD - PMD$$

**Gl. 2.24** differential Motion Displacement

Es gibt eine Reihe von Regeln, wann das Previous Motion Displacement Null gesetzt werden muss, auf diese wird jedoch hier verzichtet.

Vom Residuum r wird das Einser-Komplement *motion_r* übertragen. (f-1) ergibt eine *r_size* lange Bitkette aus Einsern.

$$motion\_r = (f-1) - r$$

**Gl. 2.25** Einer Komplement des Residual Parts eines Motion Vectors

Der Wert des Skalierungsfaktors *f* muss so gewählt werden, dass

$$-(16\times f) \le |dMD|_{\max} < (16\times f)$$

**Gl. 2.26** Randbedingung für den Skalierungsfaktor

im ganzen Bild erfüllt wird. Der kleinste Wert, der dieser Bedingung genügt, wird gewählt.

Hat man ein *f* gewählt, lässt sich aus

$$motion\_code = \frac{dMD + \sigma(dMD)\cdot(f-1)}{f}$$

**Gl. 2.27** Übertragungswert des Principal Part eines Motion Vectors

der Wert für *motion_code*, und damit letztlich auch *motion_r* ermitteln.

Der Decoder muss nun *r* aus der Umkehrung von Gl. 2.25 ermitteln und *dMD* aus

$$dMD = motion\_code \cdot f - \sigma(motion\_code)\cdot r$$

**Gl. 2.28** Rekonstruktion des differential Motion Displacement im Decoder

zu *PDM* addieren, um den aktuellen Motion Vector zu bekommen. Bei der Addition muss man mit der Modulooperation aus Gl. 2.29 noch den Wertebereich des ermittelten MV sicherstellen.

$$MD = [(PMD + dMD + 16\cdot f)\%(32\cdot f)] - 16\cdot f$$

**Gl. 2.29** Rekonstruktion des Motion Vectors

# 2.3 Standards im Bereich Videokompression

Da sich die Übertragung der Daten vom Signalprozessor zum PC sehr stark an existierende Standards anlehnt, soll hier näher auf diese eingegangen werden.

## 2.3.1 MPEG

An diese Stelle möchte ich ausdrücklich darauf hinweisen, dass die untenstehenden Flussdiagramme und die in Anhang A befindlichen Tabellen nur einen sehr kleinen Teil des MPEG Standards darstellen. Sie sind jedoch für den Einsteiger eine sehr gute Hilfe um die Methoden der Videokompression zu verstehen, außerdem erleichtern sie das Einlesen und Einarbeiten in bestehende Softwareprojekte. Für interessierte Leser gibt es eine Reihe von Werken, die die einzelnen Standards gut verständlich beschreiben. Ein repräsentatives Beispiele hierfür ist [MPF96], genaue Informationen findet man auf den Webseiten der Standardisierungsgremien [W3MP] und [W3JP]/[W3JP2].

Der MPEG Standard definiert 3 verschiedene Datenströme, den Videdatenstrom, den Audiodatenstrom und den zur Synchronisation notwendigen Systemdatenstrom. Abbildung 2.22 zeigt die weitere Aufteilung des Videodatenstroms in seine 6 Schichten.

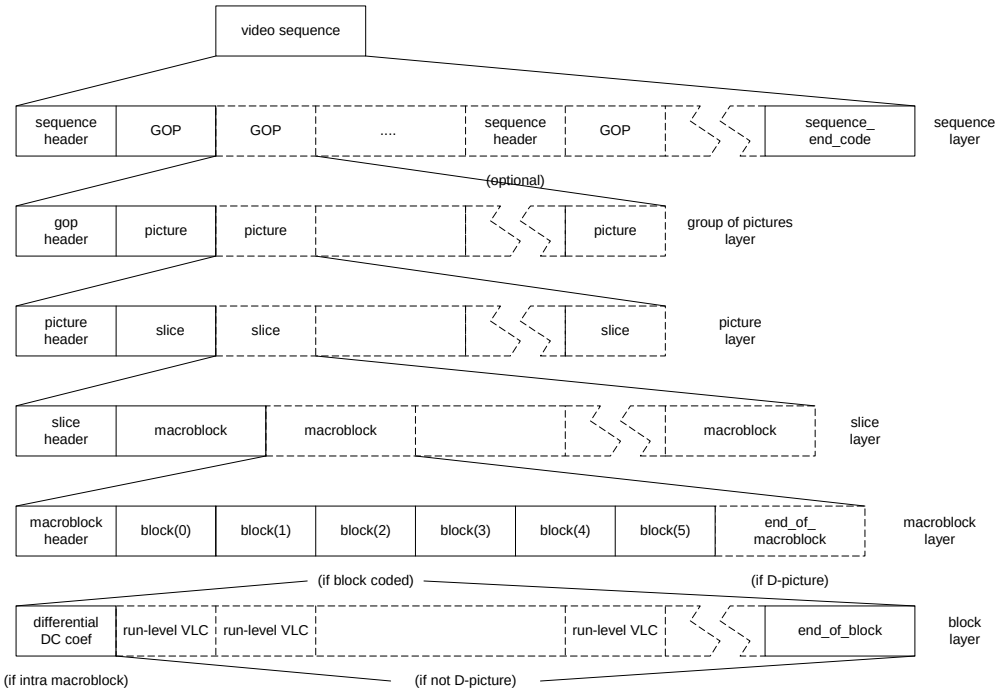


**Abbildung 2.22** MPEG Video Stream Layers

Eine Videosequenz besteht aus zumindest einer Group of Pictures GOP, wobei die im Sequence_header stehenden Informationen wie beispielsweise Breite und Höhe von GOP zu GOP geändert werden können. Abbildung 2.23 zeigt den Aufbau der Video_Sequence() Funktion, wobei next_start_code() für das Lesen (im Decoder) bzw.

das Schreiben (im Enkoder) eines Start_codes (Sequences, GOP's, Pictures und Slices haben definierte 32 Bit Start_codes) steht.

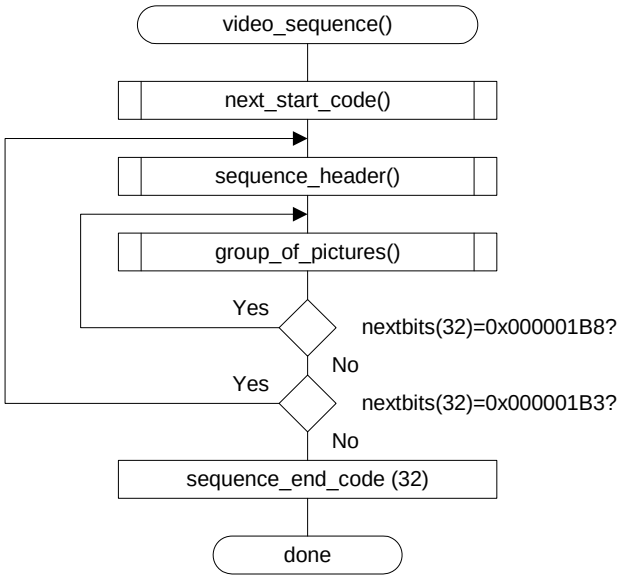


**Abbildung 2.23** Flussdiagramm einer MPEG Video Sequence

Abbildung 2.24 zeigt den Aufbau des Sequence_headers, die Werte werden in Tabelle 2.9 erläutert.

| Eintrag im Sequence_header | Bedeutung |
|---|---|
| horizontal_size | Breite |
| vertical_size | Höhe |
| pel_aspect_ratio | Breite/Höheverhältnis der pels |
| picture_rate | Bildrate (definierte Werte) |
| bit_rate | Bitrate in Vielfachen von 400 kBit/s |
| vbv_buffer_size | Untergrenze für die Datenbuffergröße des Decoders in Vielfachen von 2 kByte |
| constraint_parameters_flag | Wird gesetzt, wenn allen anderen Parameter und die Motion Vectors gewisse Obergrenzen nicht überschreiten. Limitiert den Dekodierauswand. |
| extension_and_user_data | In MPEG-1 nicht definiert, in MPEG-2 zur Übertragung von Zusatzinformation wie Profile-Wahl oder Diplay-Extensions zur Definition von Videoformat oder Farbdarstellung, … |

**Tabelle 2.9** Einträge im MPEG Sequence_header

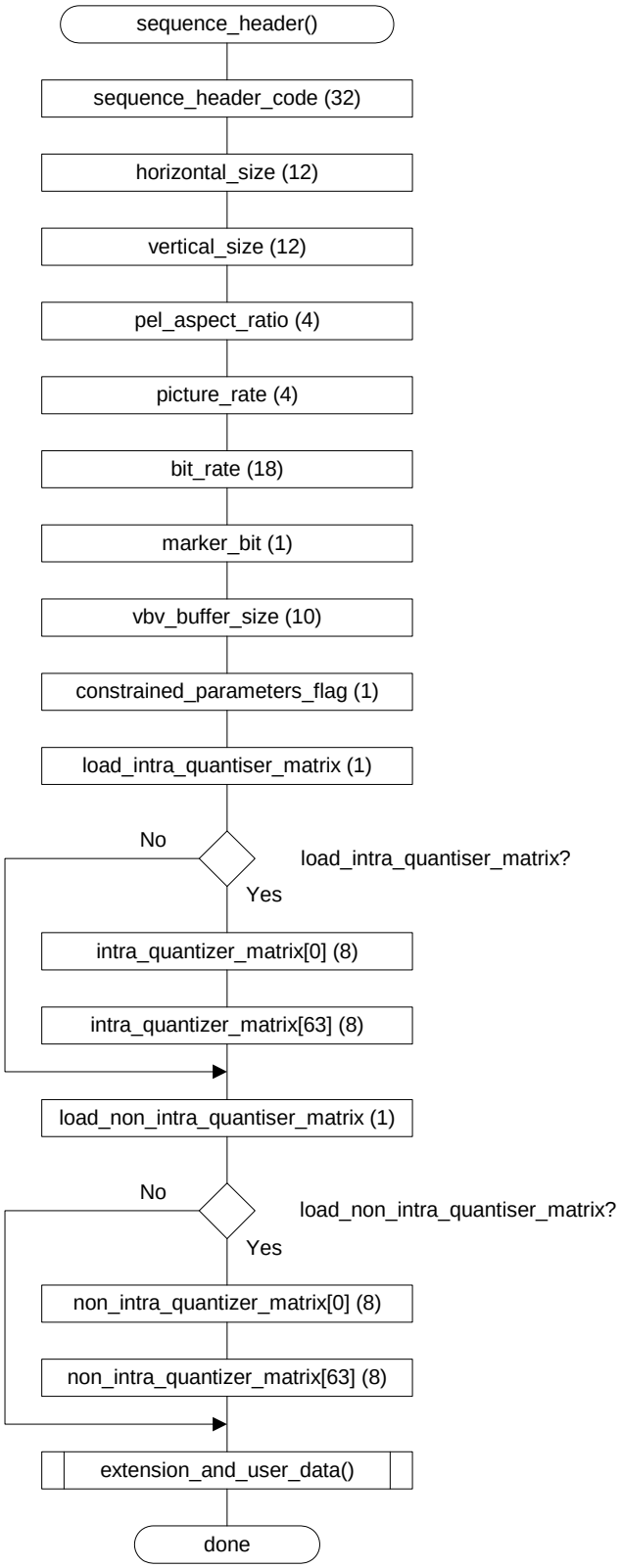


**Abbildung 2.24** Flussdiagramm des MPEG Sequence Headers

Die in Abbildung 2.25 dargestellte GOP() Funktion beinhaltet einen Zeitstempel, der für das erste der in der GOP enthaltenen Bilder gilt. Die einzelnen Einträge werden in Tabelle 2.10 erläutert.

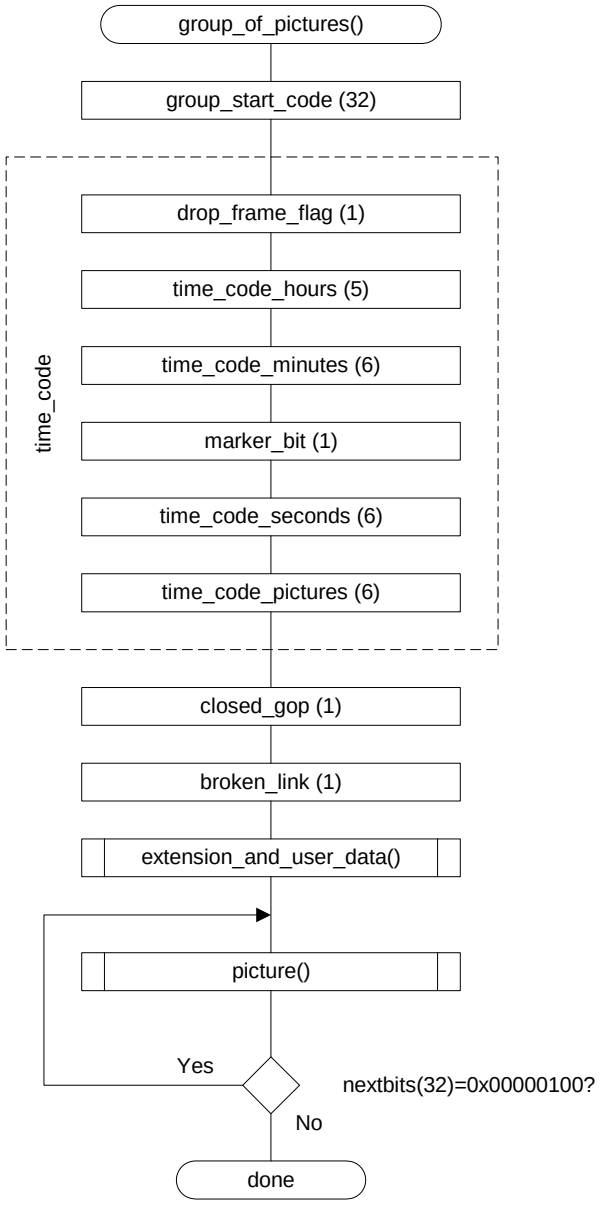


**Abbildung 2.25** Flussdiagramm der MPEG Group_of_Pictures() Funktion

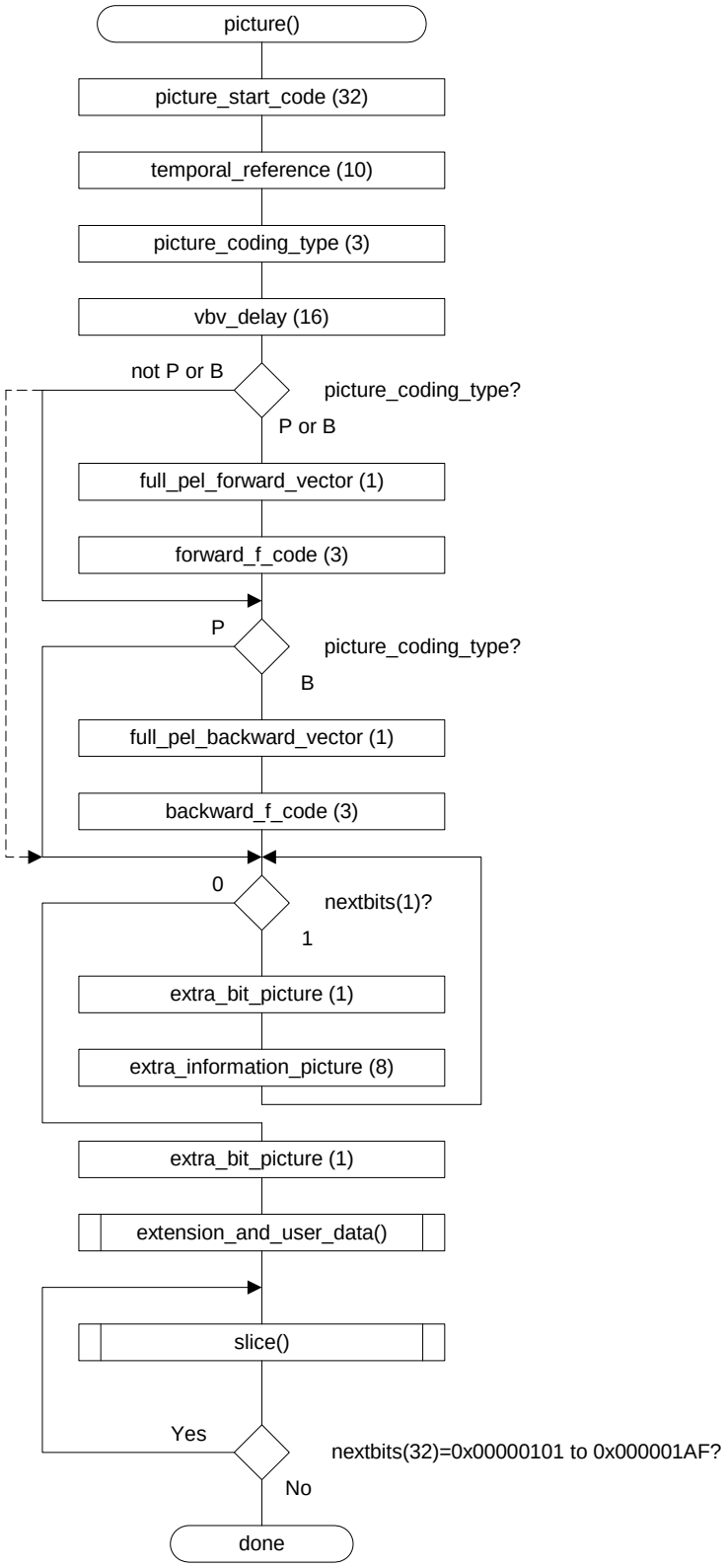


**Abbildung 2.26** Flussdiagramm der MPEG Picture() Funktion

Im Picture-Layer aus Abbildung 2.26 werden bildspezifische Angaben bekanntgegeben. Solche Angaben sind Typ des Bildes (I, P, B, D), Skalierung und Auflösung der Motion Vectors und Bildnummer innerhalb der GOP (temporal_reference), da die Reihenfolge der Übertragung bei B-Frames nicht mit der Reihenfolge der Anzeige übereinstimmt. vbv_delay gibt an, wie viele Bits gelesen werden müssen, bevor das Bild decodiert werden kann.

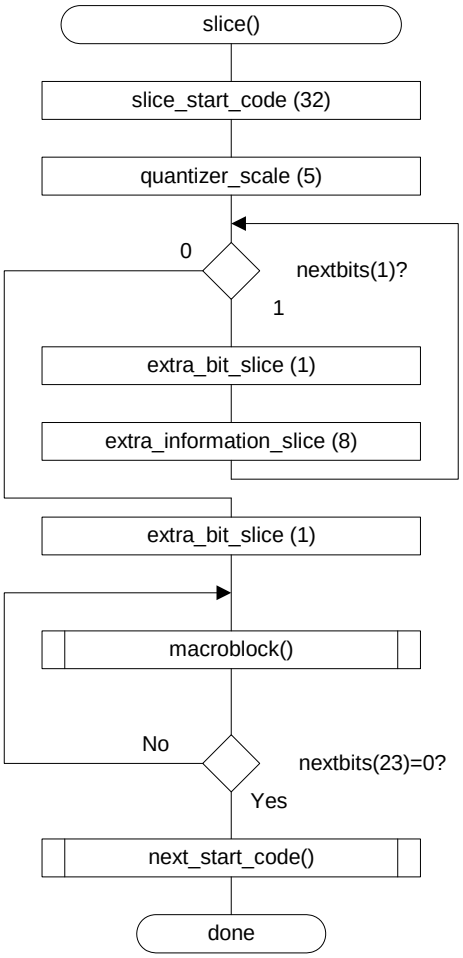


**Abbildung 2.27** Flussdiagramm der MPEG slice() Funktion

Der Slice-Layer in Abbildung 2.27 mag zunächst unnötig erscheinen, da er außer einem zusätzlich Overhead durch den Start_code keine Information enthält und Slices nicht über eine Zeile Makroblocks hinausgehen dürfen (jede Zeile Makroblocks muss mindestens ein Slice enthalten). Diese Betrachtung ist jedoch nicht ganz richtig, da der wichtige Quantisierungsfaktor von Slice zu Slice verändert werden kann und dem Enkoder somit das mächtige Werkzeug zum Steuern von Bitrate und Bildqualität bereitstellt.

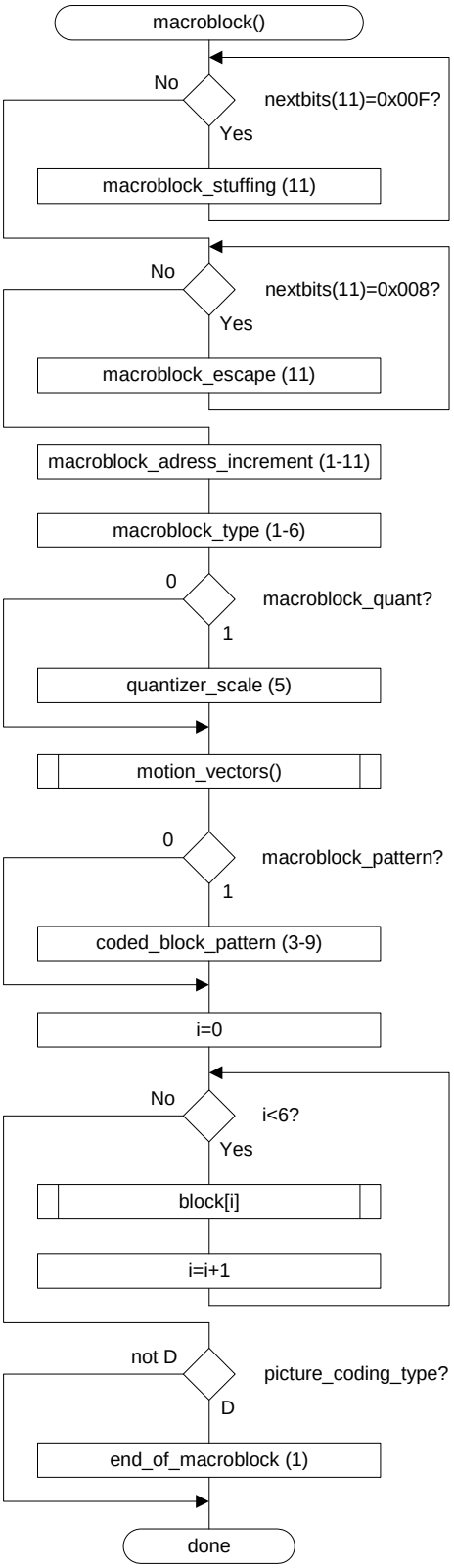


**Abbildung 2.28** Flussdiagramm der MPEG Macroblock() Funktion

Der Header im Makroblock-Layer in Abbildung 2.28 enthält Informationen über die Position des Makroblocks relativ zum letzten kodierten Makroblock (siehe Abschnitt 2.2.2.2), die Motion Vectors, die tatsächlich kodierten Blöcke innerhalb des Makroblocks, den Typ des Makroblocks und einen Quantisierungsfaktor. Da die meisten dieser Angaben optional über Bits gesetzt werden können, kann man Makroblocks individuell sehr gut komprimieren.

Auf die Darstellung der motion_vectors() Funktion wurde verzichtet, da man sie sich leicht vorstellen kann: Über Bits wird gesetzt, ob ein MV, bzw. ein Residuum vorhanden ist, und der Code wird dann übertragen/gelesen.

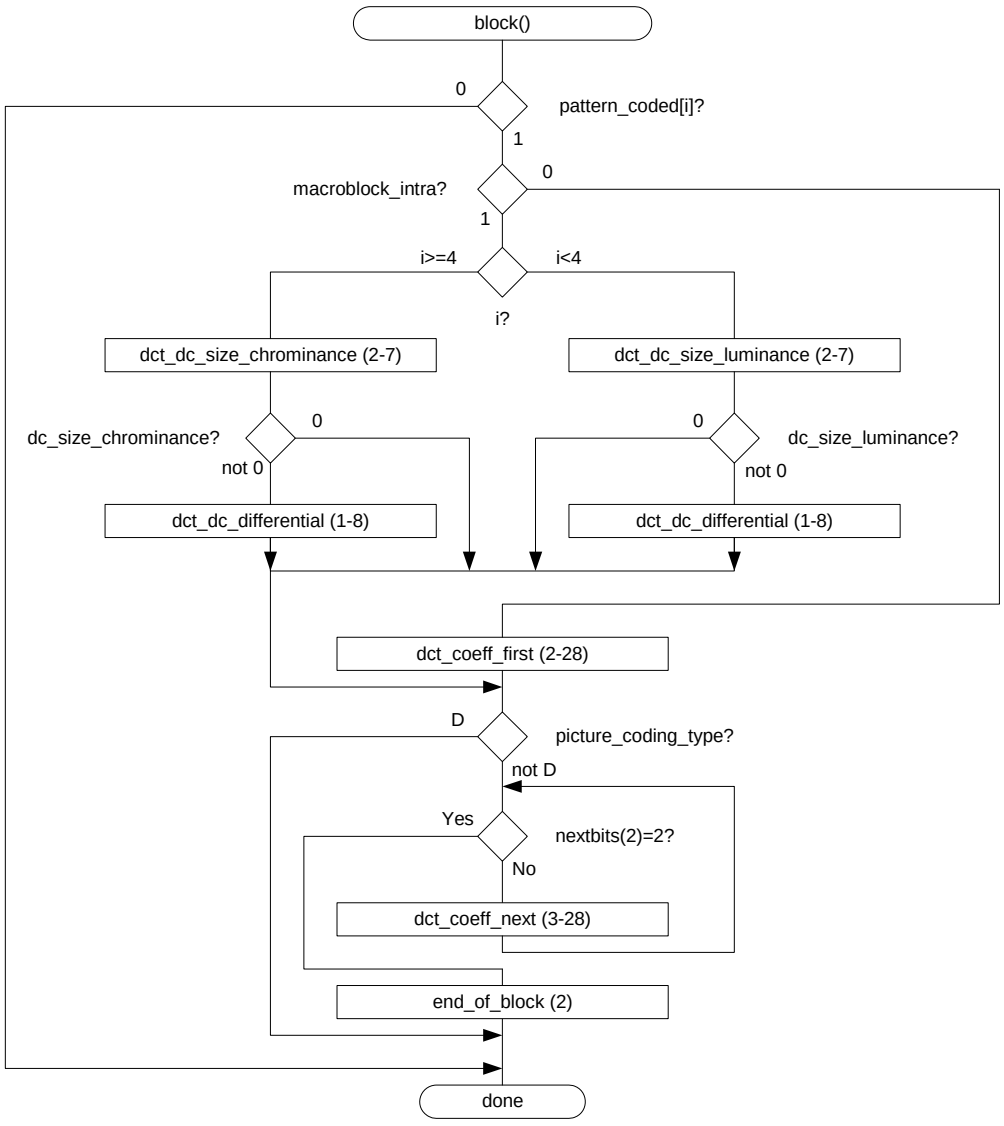


**Abbildung 2.29** Flussdiagramm der MPEG Block() Funktion

Der Block-Layer ist die unterste Schicht einer MPEG Videosequenz. Er enthält die Daten eines 8x8 DCT Blocks. Die Kodierung erfolgt unterschiedlich, je nachdem ob

es sich um einen Luminanz- oder Chrominanzblock handelt und ob es sich um ein I-Frame oder ein Differenz-Frame handelt. Außerdem können Blocks aus Differenzbildern, die nur Nullwerte enthalten, einfach übersprungen werden. Abbildung 2.29 zeigt das Flussdiagramm der MPEG block() Funktion.

| Eintrag im GOP_header | Bedeutung |
|---|---|
| drop_frame_flag | Ist bei einer Bildrate von 29.97 Hz gesetzt, um durch Weglassen von Bilder eines 30 Hz Datenstroms echte 29.97 Hz zu erreichen, sonst immer 0. |
| time_code_hours, minutes, seconds | Geben die vergangene Zeit von Beginn der Sequence zum ersten Bild der GOP an. |
| time_code_pictures | Anzahl der Bilder in der nächsten Sekunde. |
| closed_gop | Die Bilder in der GOP sind unabhängig von anderen GOP's, d.h. im ersten und letzten Bild referenzieren keine Motion Vectors Bilder außerhalb der GOP. |
| broken link | Meldet Fehler im folgenden Bitstrom, sollte also immer 0 sein. |

**Tabelle 2.10** Einträge im MPEG Group of Pictures_header

### 2.3.2 H.261 und H.263

Die ersten Videokompressionsstandards wurden lange vor dem MPEG Standard verabschiedet. So stammt der H.261 Standard (besser bekannt als p x 64) aus dem Jahre 1984. Er wurde für Videotelephonie entwickelt, die verwendeten Übertragungskanäle waren ISDN Kanäle mit p * 64 kBit/s Übertragungsrate. Da der Fokus bei der Entwicklung des Standard auf Echtzeitanwendungen lag, war eine Unsymmetrie in der Komplexität von Enkoder und Dekoder nicht duldbar. Dennoch unterscheidet sich H.261 nicht wesentlich von MPEG-1 (viele Personen saßen in beiden Standardisierungsgremien):

Das Bild wird in Makroblöcke und Blöcke gleicher Größe wie bei MPEG aufgeteilt, die 8x8 DCT wird angewandt und die Koeffizienten werden mit demselben Zig-Zag-Scan ausgelesen. Die inter-frame Quantisierung erfolgt mit schmalem Nullband, die intra-frame Quantisierung mit Breitem. Die zeitliche Redundanz der Bilder wird mit Motion Compensation und Estimation ausgenutzt und Motion Vectors werden differentiell übertragen. Es kann der Quantisierungsfaktor verändert werden, Blöcke können übersprungen und mit denselben coded_block_pattern wie in MPEG-1 dargestellt werden. Viele der VLC Tabellen sind gleich oder fast gleich den in MPEG verwendeten.

Es gibt dennoch einige nicht unwesentliche Unterschiede:

Die Quantisierung erfolgt nicht über eine Matrix, sondern über einen einzigen Wert, der sich nur alle 11 Makroblöcke ändern lässt. Bei intra-kodierten AC Koeffizienten wird Quantisierung mit breitem Nullband verwendet. Es existieren nur vier Layer, die Motion Vectors gibt es nur mit full Pel Auflösung und einem Wertebereich von ±15. Der wesentliche Unterschied ist aber, dass, um Verzögerungszeiten zu minimieren, zur Prediction nur das vorhergehende Bild verwendet wird. Deshalb gibt es auch keine B-Frames in H.261.

11 Jahre später wurde ein neuer Standard für digitale Videotelephonie verabschiedet, der auf die beiden älteren Standards H.261 und H.262 (MPEG-2) referenziert: H.263. Dieser Standard verlangt, dass die Videodaten mit der Vorgehensweise von H.261 komprimiert werden, allerdings unterstützt er zusätzlich 4 Modi. Der Unrestricted Mode erlaubt größere Motion Vectors, die auch auf Punkte außerhalb des Bildes zeigen können, wobei das Bild bis dorthin gespiegelt wird. Im Arithmetic Coding Mode werden die Huffmankodierung mit ihren VLC Tabellen durch die effizientere arithmetische Kodierung ersetzt. Dieser Mode erlaubt gleiche Bildqualität bei kleinerem Datenvolumen. Im Advanced Prediction Mode bekommt jeder Luminanzblock einen eigenen Motion Vector und im PB-Frames Mode werden zwei Bilder als Eines kodiert, indem ein P-Frame das letzte I-Frame referenziert und ein B-Frame zwischen den beiden generiert wird. Die Makroblöcke des B-Frame werden immer gleich nach den an gleicher Position Stehenden im P-Frame übertragen.

Anders als bei MPEG kann das Bild bei H.261 und H.263 nur definierte Größen (Formate) annehmen. Bei H.263 wurden 3 zusätzliche Formate eingeführt, Höhe und Breite des Bildes oder Seiten/Höhenverhältnis der Pels muss nicht übertragen werden. In allen nicht arithmetischen Modi wurden die Codetabellen geändert und das EoB-Zeichen durch ein Prefix an alle anderen Kombinationen ersetzt, welches besagt, dass es sich bei Diesem um den letzten Koeffizienten im Block handelt. Der Quantisierungsfaktor kann bei H.263 für jeden Makroblock geändert werden. Die differentielle Übertragung des Motion Vectors hängt bei H.263 von allen drei benachbarten Makroblöcken ab.

Genauere Informationen über H.261 und H.263 finden sich als Einführung auf [W3261] und [W3263] und ausführlich in den Standards [ITU93] und [ITU95].

# Kapitel 3

# Problemanalyse

Bereits in der Einleitung dieses Dokuments wurde die Aufgabenstellung dieser Arbeit erläutert. Im folgenden Kapitel soll noch einmal detailliert die Aufgabenstellung dargestellt werden. Anschließend wird die Aufgabenstellung in Teilprobleme zerlegt und im dritten Unterpunkt eine Übersicht über vorhandene Hard- und Softwarekomponenten gegeben. Abschließend wird ein Überblick über das entstandene System gegeben.

## 3.1 Anforderungen

Noch einmal zusammengefasst ist das Ziel dieser Diplomarbeit die Entwicklung einer Plattform zur Bildbearbeitung mit folgenden Eigenschaften und Anforderungen:

- Kleine Baugröße (die Größe der Platine ist aufgrund der ebenfalls geforderten Verwendung am Fussballroboter „Tinyphoon“ auf 75 x 75 mm² limitiert)
- Hohe Rechenleistung, um die Voraussetzungen für die Abarbeitung eines Videokompressionsalgorithmus bzw. eines Objekterkennungsalgorithmus zu ermöglichen
- Implementierung einer Schnittstelle zum am Fussballroboter verwendeten CAN-Bus
- Implementierung eines Funkmoduls zur drahtlosen Videodatenübertragung
- Implementierung der Software-Schnittstellen für alle verwendeten Komponenten
- Implementierung eines Videokompressionsalgorithmus, der es erlaubt, Videodaten live an einen PC zu übertragen
- Komponenten zur Aufnahme von Audio- und Videodaten
- Wahl möglichst kostengünstiger Komponenten und Herstellungsverfahren
- Flexibilität durch Wahl von Komponenten, die durch pinkompatible Alternativen anderer Hersteller ersetzt werden können
- Flexibilität durch Vorsehen von optional bestückbaren Zusatzausstattungen
- Möglichst geringe Stromaufnahme um einen Betrieb als drahtlose Überwachungseinheit, bzw. als Modul eines batteriebetriebenen Kleinroboters zu gewährleisten

- Mitintegration aller notwendigen Bauteile für Spannungs- und Taktversorgung, sowie eine geeignete Resetierschaltung
- Um die Herstellungskosten der Platinen so niedrig wie möglich zu halten, sollen nicht mehr als vier Layer und 500 Vias verwendet werden
- Weiters müssen alle elektrischen Bauteile als SMD-Bauform (und nach Möglichkeit bei großen Chips als BGA-Bauform) ausgeführt sein um möglichst viel Platz für Routing zu sparen

## 3.2 Teilprobleme

Da die Aufgabenstellung nun klar formuliert und abgegrenzt ist, wird sie nun hinsichtlich ihrer Realisierungsmöglichkeiten untersucht. Für die einzelnen Teilprobleme müssen verschiedene Lösungsansätze gefunden und verglichen werden.

Die gesamte Aufgabe lässt sich grob in 5 Teilprobleme zerlegen:

1. Entwicklung einer Systemarchitektur, parallel dazu Recherche, welche Module in Hardware bzw. Software implementierbar und am Markt erhältlich sind. Nach der Schaltungsentwicklung folgt Routing der Platine.
2. Implementierung aller Interfaces zu den Komponenten für den Signalprozessor. Die Hardware der Plattform ist damit fertig und kann in anderen Projekten verwendet werden.
3. Implementierung eines effizienten Bildkompressionsalgorithmus am PC
4. Erweiterung des Algorithmus zur Kompression von Videodaten
5. Portierung des Algorithmus auf den Signalprozessor und Geschwindigkeitsoptimierung

Das folgende Kapitel widmet sich der Auswahl der Kernkomponenten. Die Beschreibung der Interfaces befindet sich in Kapitel 5, die Implementierung des Algorithmus in Kapitel 4. Die Schaltpläne sind in Anhang B und die Bestückungspläne in Kapitel 5.2 angeführt.

## 3.3 Recherche

Um den Anforderungen der Aufgabenstellung gerecht zu werden, müssen alle Komponenten des Systems dem neuesten Stand der Technik entsprechen. Obwohl die Verarbeitungsleistung moderner Chips stetig steigt, deren Baugröße trotzdem sinkt und die Stromaufnahme laufend geringer wird, war es sehr schwierig, geeignete Komponenten für das System zu finden. Es kamen für einzelne Komponenten überhaupt nur die schließlich Verwendeten infrage, da es an konkurrenzfähigen Alternativen mangelte.

Die folgenden Unterkapitel beschreiben die Auswahl der Kernkomponenten im Detail.

## 3.3.1 Hardware

### 3.3.1.1 Signalprozessor

Die Überlegung, den Videokompressionsalgorithmus eventuell in einem FPGA auszuführen, sind aus mehreren Gründen nicht zielführend:

- Die Kostenersparnis bei einer Massenproduktion gegenüber einem Signalprozessor invertiert sich bei einer Klein- oder Mittelserie zu erheblichen Mehrkosten
- Die Implementierung von anwendungsspezifischen Objekterkennungsalgorithmen wird unmöglich
- Ein FPGA bringt in punkto Stromverbrauch nur minimale Vorteil und in punkto Baugröße nur einen Nachteil
- Die Implementierung des Videokompressionsalgorithmus ist in beiden Fällen gleich aufwendig

Es gibt eine Fülle von verschiedenen Signalprozessoren am Markt, die – jede Familie für sich – für verschiedene Anwendungen optimal mit on-Chip-Pheripherie ausgestattet sind. Um dem Anwender die Möglichkeit zu bieten, verschiedene Kameras zu verwenden, muss es zwischen Signalprozessor und Kamera eine standardisierte Schnittstelle geben. Diese Schnittstelle ist als ITU-656 Standard bekannt, genaue Details findet man in [ITU98]. Es kommen demnach nur Signalprozessoren mit einem Videointerface nach o.g. Standard in Frage.

Die meisten Hersteller von Signalprozessoren haben Familien mit Videointerface im Programm, allerdings unterscheiden sich die Hauptanwendungen doch erheblich. Tabelle 3.1 gibt beispielhaft einige Eckdaten solcher Prozessoren wieder.

| **Hersteller**<br>Chipbezeichnung | **Analog Devices**<br>ADSP-BF533 Blackfin | **Motorola**<br>MC9328MXL i.MX | **Texas Instruments**<br>TSM320DM642 |
|---|---|---|---|
| Leistungsaufnahme | 200 mW | 1100 mW | 2000 mW |
| Taktfrequenz | 600 MHz | 200 MHz | 600 MHz |
| Pins | 160 | 256 | 548 |
| Chipgröße | 9x9 mm² BGA | 14x14 mm² BGA | 27x27 mm² BGA |
| Preis (06/03) | 10 USD | 10 USD | 85 USD |
| Architektur | 16 Bit fixpoint | 32 Bit fixpoint | 32 Bit fixpoint |
| Videointerfaces | 1 | 1 | 3 |
| on Chip Memory | 144 kByte | 144 kByte | 288 kByte |

**Tabelle 3.1** Vergleich dreier Signalprozessoren mit Videointerface nach ITU-656

Der Chip von Analog Devices hat die bei weitem günstigste Stromaufnahme und Baugröße und zielt auf Anwendungen mit hoher Rechenleistung in batteriebetriebenen

Geräten wie Handys ab. Die Konkurrenzprodukte von Motorola und TI besitzen beide einen Ethernet-Controller on Chip, letzterer sogar einen PCI-Controller. Sie sind eher für Anwendungen mit fixer Stromversorgung und Anbindung an ein Netzwerk gedacht (TI weist ausdrücklich darauf hin, dass der Prozessor (trotz seiner vielen on-Chip-Pheripherals) ohne Kühlung betrieben werden kann!).

Es wurde deshalb der Blackfin Prozessor von Analog Devices gewählt. Ich möchte hier am Rande erwähnen, dass die Firma Analog Devices uns schon einige Monate vor Markteinführung Samples des Blackfin zur Verfügung gestellt hat, ohne die dieses Projekt nicht in der Form realisierbar gewesen wäre.

Ein Signalprozessor ist im Gegensatz zu einem Mikrokontroller ohne externen Speicher fast nie einsatzbereit. Außerdem benötigt er eine externe Spannungsregulierung für die Core-Spannung und wie (fast) jeder Mikrokontroller braucht er weiters eine externe Taktversorgung und eine Resetbeschaltung.

Es soll hier noch kurz auf die Auswahl des Speichers eingegangen werden.

Bei Speicherbausteinen – egal ob RAM oder ROM – gibt es nach außen nicht viele Freiheiten für den Hersteller, die verschiedenen Pins zu wählen. Es gibt deshalb einige Familien von Speicherbausteinen verschiedener Hersteller, die pinkompatibel sind. Es wurde großer Wert darauf gelegt, solche Bausteine zu verwenden. Im Zuge der Hardwarerealisierung ist mir dieses Prinzip schon zugute gekommen, da ein Hersteller Lieferprobleme hatte. Aufgrund der Kompatibilität konnte ich den Baustein eines anderen Herstellers verwenden.

Beim RAM habe ich mich an die Referenzimplementierung von Analog Devices gehalten, welche einen 32 MB RAM Baustein mit 133 MHz Taktfrequenz vorsieht. Beim ROM wurde ein einfacher, aber schneller 2 MB ROM Standardflash gewählt. In der Referenzimplementierung enthält der Flash-Chip noch einige frei programmierbare Gatter in Form eines kleinen PAL, diese sind jedoch bei diesem Projekt nicht in Verwendung.

#### 3.3.1.2 Kamera

Als Kamera wurden nur solche Kameras in Betracht gezogen, wie sie in mobilen Geräten wie Handys oder PDA's Verwendung finden. Es gibt auch hier eine Reihe von Herstellern, die Minikameras mit unterschiedlichen Spezifikationen anbieten. Bei der Auswahl der Kamera waren folgende Punkte von Bedeutung:

- ITU-656 Standard-Interface
- Hohe Bildrate mit einstellbarer Auflösung
- Kleine Bauform, wie sie in Handys oder Handheld-PC's verwendet wird
- Geringe Stromaufnahme

Tabelle 3.2 zeigt eine Gegenüberstellung am Markt erhältlicher Kameras.

| **Hersteller** | **Omnivision** | **Dialog Semi.** | **Conexant** | **Sharp** |
|---|---|---|---|---|
| Chipbezeichnung | OV 7648 FB | DA 3510 | MK100-D160 | LZ0P390M |
| Abmessungen [mm] | 10 x 9 x 7,3 | 10 x 10 x 8 | 12 x 12 x 9 | 11 x 11 x 5 |
| max. Bildrate | 60 | 30 | 30 | 15 |
| Auflösung | bis VGA | bis VGA | VGA fix | bis CIF |
| Stromaufnahme | 40 mW | 200 mW | k.A. | 45 mW |

**Tabelle 3.2** Vergleich von vier Minikameras

Wie aus oben stehender Tabelle ersichtlich ist, hat das Modell der Firma Sharp die kleinste Bildrate und die kleinste Auflösung. Das Modell von Conexant hat eine fix eingestellte Auflösung, über den Stromverbrauch wurden keine Angaben gefunden. Die anderen beiden Modelle unterscheiden sich nicht in den einstellbaren Bildformaten und fast nicht in der Bauform, allerdings ist die Kamera von Omnivision der von Dialog in punkto Bildrate und Stromverbrauch doch deutlich überlegen, weshalb sie auch in diesem Projekt Verwendung findet.

#### 3.3.1.3 Funkmodul

Schon seit längerer Zeit versucht man in allen Bereichen der Datenübertragung die drahtgebundenen Systeme durch drahtlose Systemen zu ersetzen. Die Entwicklungen auf diesem Gebiet sind noch lange nicht abgeschlossen, wie allein die Fülle von Modulationsarten zeigt, die schon implementiert und noch angedacht sind.

Für ein mobiles Gerät, welches mit anderen Geräten kommunizieren soll, ist es unvermeidbar, mit einem Funksystem ausgestattet zu sein, jedoch will die Wahl des Funkmoduls gut überlegt sein.

Bei der Auswahl des Funkmoduls gab es primär vier Argumente, die sehr viele Kandidaten ausscheiden ließen:

- Senden in einem lizenzfreien Frequenzband
- Hohe Datenrate (~ 1 Mbaud)
- Geringer Stromverbrauch
- Kleine Bauform

Die ersten beiden Forderungen lassen sich nur im lizenzfreien 2,4 GHz ISM Band vereinen, in dem Technologien wie Bluetooth und WLAN ebenfalls beheimatet sind.

Die Frage, ob man Bluetooth oder einen protokolllosen Transceiver verwenden sollte, wurde entscheidend durch die fertigen Bluetooth-Profile beeinflusst. Es gibt beinahe 20 verschiedene Profile, allerdings keines zum Datenstreaming mit hoher Bitrate und hohem Datenvolumen und konsequenterweise auch keines für die Übertragung von Videodaten. Das Audiostreaming Profile eignet sich nicht für die höheren Datenraten einer Videoübertragung. Ein weiterer Punkt, der in diesem Fall gegen Bluetooth anzuführen ist, ist der hohe Protokolloverhead, der auch bei der Verwendung von HCI-

Kommandos auftritt. WLAN schied aufgrund der nichtvorhandenen Mini-Module aus, es bleibt also nur eine Lösung: die propritäre Lösung.

Nachdem nun die Entscheidungen hinsichtlich Übertragungsfrequenz und Protokoll getroffen sind, können am Markt erhältliche Funkmodule verglichen werden.

| **Hersteller** | **Nordic** | **RF-Waves Ltd.** | **OPC network** |
|---|---|---|---|
| Chipbezeichnung | nrf2401 | RFW 102 M | OPC 1505 FM |
| Stromaufnahme | | | |
| senden | 13 mA | 21 mA | 65 mA |
| empfangen | 18 mA | 38 mA | 59 mA |
| Stand By | 12 µA | 2,6 µA | 1,5 mA |
| Interface | SPI | IO, Clk | SPI |
| Abmessungen | 5 x 5 | 4 x 4 | 46 x 41 |
| Sendeleistung | 0 dBm | 2 dBm | 6 dBm |
| Empfangssensitivität | - 90 dBm | - 80 dBm | - 100 dBm |

**Tabelle 3.3** Vergleich dreier Funkmodule

Aus der Tabelle geht hervor, dass das Modul der Firma Nordic den geringsten Stromverbrauch der drei verglichenen Module hat. Die Reichweite ist größer als beim Konkurrent RFW, allerdings weit geringer als beim Modul von OPC, das jedoch aufgrund seiner Abmessungen und seines Stromverbrauchs für den Einsatz in einem möglichst kleinen batteriebetriebenen Gerät nicht in Frage kommt. Aufgrund des geringen Stromverbrauchs und der akzeptablen Baugröße habe ich mich für das Modul der Firma Nordic entschieden.

#### 3.3.1.4 Audioperipherie

In den ersten Schaltungsentwürfen wurde noch kein Audio Codec vorgesehen, da ein akustischer Sensor beim Roboterfussball keinen Vorteil bringt. Weil aber ausreichend Platz auf der Platine vorhanden ist, von Analog Devices eine Referenzimplementierung in Kombination mit einem Blackfin Prozessor existiert und ein akustischer Sensor bei anderen Anwendungen wie Überwachungseinheiten Vorteile bringen kann, habe ich mich entschieden, einen Audio Codec zu verwenden. Es wurde der am EZ-LITE Kit von Analog Devices – dem Entwicklerboard für den Blackfin – verwendete Baustein gewählt. Die Datenblätter des EZ-LITE Kit und des Audio Codec AD1885 findet man auf der Homepage von Analog Devices [W3AD].

Der AD1885 wird über ein SPORT angesprochen und liefert Audiodaten mit variabler Abtastrate im AC´97 Format (wie die meisten handelsüblichen Soundkarten). Details zum AC´97 (Audio Codec ´97) Format finden sich auf der Homepage von Intel [W3INT], dem Entwickler.

#### 3.3.1.5 Spannungsversorgung

Die für oben beschriebene Komponenten notwendige Spannungsversorgung beinhaltet

- 3,3 V bei ca. 500 mA
- 2,5 V bei ca. 80 mA (nur die Kameras)
- 5,0 V bei ca. 150 mA (nur der Audioteil)
- ~1,0 V (programmierbar) Core-Spannung des Signalprozessors

Um einen möglichst hohen Wirkungsgrad der Spannungsversorgung zu gewährleisten, wurden nur DC/DC Switches verwendet, die einem Wirkungsgrad von nur wenig unter 100% erreichen, was für ein batteriebetriebenes System essentiell ist. Für die Erzeugung von 2,5 und 5,0 V aus 3,3 V gibt es spezielle Bausteine mit Referenzbeschaltung, welche auch verwendet wurden. Es sind dies MAX8885 (5,0V) und MAX683 (2,5 V) von Maxim. Für die Erzeugung von 3,3 V über einen großen Eingangsspannungsbereich (~ 5 – 13 V) gibt es ebenfalls einen optimierten Chip, den Analog Devices als ADP3088 vertreibt. Die Core-Spannung des Signalprozessors wird nicht über einen Spannungsregler, sondern einen FET reguliert, den Analog Devices in der Referenzimplementierung als NDS8434A von Fairchild Semiconductors spezifiziert.

## 3.3.2 Software

### 3.3.2.1 Beispielcodes

Die Suche nach Code Beispielen für die „intelligenten“ Komponenten gestaltete sich aufgrund der Neuheit der meisten Komponenten als sehr schwierig. So veröffentlichen manche Hersteller Application Notes mit Beispielcode teilweise erst Monate nachdem ein neues Produkt am Markt ist, bzw. es werden unvollständige, unkommentierte oder schlicht falsche Beispielcodes veröffentlicht. Eine zweite potentielle Quelle für Beispielcodes sind Bildungseinrichtungen wie technische Mittelschulen und Universitäten. Auf deren Websites finden sich oft Beispiele, die den Studenten beim Vorbereiten einer Laborübung helfen sollen.

#### a) Beispiele der Hersteller

Für das Funkmodul gibt es vom Hersteller keine, für den Mikrokontroller – der ja auf der 8051 Architektur basiert – gibt es sehr viele Beispiele, allerdings nur zwei, in denen der CAN-Controller angesprochen wird. Von diesen Zweien war eines fehlerhaft, das zweite sehr schlecht lesbar, weil unkommentiert und mit nichts sagenden Variablennamen programmiert.

Für den Blackfin gibt es eine ganze Reihe von Code-Beispielen, da es Vorgängermodelle gibt, die weitgehend codekompatibel sind. Wie oben schon erwähnt, gibt es auch ein Beispiel zum Ansprechen des Audio Codec AD1885.

Die jeweiligen Homepages sind bei den Internetreferenzen zu finden.

#### b) Beispiele von Bildungseinrichtungen

Auf einer solchen Seite fand ich ein Beispiel für das Funkmodul. Dieses Beispiel ist für einen Risk-Prozessor geschrieben, also leider für den Blackfin so nicht brauchbar, die Struktur lässt sich aber übernehmen.

Leider wurde die betreffende Seite schon wieder gelöscht, sie stammt von der TU-Chemnitz.

Zusammenfassend lässt sich nur sagen, dass manche Hersteller sehr wenig Support mit Code-Beispielen bieten. Dieser Mangel an Support stellt meines Erachtens ein großes Manko dar, da die selbstständige Entwicklung der Interfaces viel Zeit kostet und somit Zusatzkosten verursacht, die in die Kaufentscheidung beim Kunden sicher miteinfließen.

#### 3.3.2.2 Open Source Projekte

Verschiedene Communities, Studentengruppen oder Einzelpersonen arbeiten weltweit an der Implementierung von Standards in lauffähige Programme. Ich habe mich am Beginn der Programmiertätigkeit sehr intensiv mit vorhandenen Codes und Codecs auseinandergesetzt.

So gibt es beispielsweise fertige En-/Dekoder für JPEG, MPEG, MPEG-2 und MPEG-4. Auch für Teile des Algorithmus gibt es fertige Implementierungen, wie die DCT, die DWT und einige Motion Estimation Algorithmen. Die jeweiligen Homepages sind bei den Internetferenzen zu finden.

Das grundlegende Problem mit fertigen Softwareprojekten ist das Einlesen. Ohne sehr genaue Kenntnisse über Videokompression und genaue Kenntnisse über den jeweiligen Standard kann man sich in solche Projekte nicht oder nur bedingt einarbeiten.

Ein weiteres Problem speziell mit MPEG ist, dass sich der MPEG Standard nicht sehr gut für echtzeitfähige Datenübertragung eignet, und deshalb nur Teile davon für dieses Projekt brauchbar sind. D.h. bei Verwendung eines fertigen Codecs müsste man nicht nur den Dateneinleseteil und die Metaangaben über das verwendete Format, die Bildgröße, Bitrate, usw. überarbeiten, sondern überall die Codeteile für B-Frames und D-Frames finden und entfernen. Die Unterstützungen für moderne Prozessoren wie MMX und dergleichen müssten ebenfalls entfernt bzw. abgeändert werden.

Eine detailliertere Argumentation befindet sich im Kapitel 5.

Ich habe mich dagegen entschieden, einen fertigen Codec zu verwenden, da ich der Meinung bin, der Zeitaufwand entspricht in etwa dem, den notwendigen Code selber zu entwickeln, wenn es nicht sogar länger dauern würde.

Die von mir verwendeten Codes von externen Quellen sind die DCT (für PC und Blackfin optimiert) und der TSS Algorithmus in Assembler für den Blackfin.

Mein Kollege Roland Oberhammer beschäftigt sich in seiner Diplomarbeit mit Bilderkennung und Objekterkennung am Signalprozessor. Er hat unter anderem das Interface für die Kameras geschrieben, welches von mir übernommen wurde.

### 3.3.3 Systemarchitektur

In den vorigen Kapiteln wurde die Aufgabenstellung erläutert, Probleme diskutiert, die gefundenen Lösungen präsentiert sowie die verwendete Hardware vorgestellt und die Konzeption der Software beschrieben. An dieser Stelle ist es nun an der Zeit, einen Gesamtüberblick über das System zu geben und das Zusammenspiel der einzelnen Komponenten der Plattform zur Bildbearbeitung autonomer Kleinstroboter darzustellen. Die Architektur des Projekts wird als Software- und Hardwarearchitektur getrennt vorgestellt. Diese Trennung in Hardware und Software fördert die Übersichtlichkeit und erleichtert die Beschreibung und Darstellung des Systemverhaltens.

**Abbildung 3.1** Hardwarearchitektur

Beginnen möchte ich mit der Darstellung der Hardwarearchitektur in Abbildung 3.1. Das Herz der Hardware bildet der Blackfin ADSP-BF533 Signalprozessor der Fa. Analog Devices, der über 32 MB RAM und 2 MB ROM externen Speicher verfügt.

Der Systemtakt von 20 MHz wird von einem PLL-synchronisierten Clockverteiler IDT-2305 der CPU, den Kameras, dem Funkmodul und dem Coprozessor zur Verfügung gestellt. Der Audio Codec benötigt eine eigene Taktversorgung, da eine Taktrate von 24.576 MHz vorgeschrieben ist.

Der Mikrokontroller, oben als Coprozessor bezeichnet, verfügt über einen on-Chip CAN-Controller und wird über die UART mit dem Signalprozessor verbunden, außerdem werden zwei der vier auf der Platine befindlichen LED's durch ihn geschaltet. Er basiert auf der 8051 Architektur, verfügt zusätzlich über einen X2 Mode, der die Befehlsausführungszeiten gegenüber einem normalen 8051 halbiert. Optional bestückbar sind eine Referenzspannungsquelle für den Analog/Digital Wandler des Mikrokontrollers sowie zwei Helligkeitssensoren, die auch als Light2Voltage- oder Lux2Volt-Sensoren bezeichnet werden. Um den Mikrokontroller wirklich als Coprozessor verwenden zu können, sind drei vollständige Ports verfügbar (diese können als GPIO's oder als 16 Bit Daten/Adressbus für externen Speicher und 8 A/D-Wandler-Eingänge verwendet werden) und das Protokoll zw. Mikrokontroller und Signalprozessor ist sehr einfach und erweiterbar gestaltet.

Der Audio Codec AD1885 von Analog Devices kann zwei Lautsprecher treiben und Signale von zwei Mikrophonen verarbeiten, wobei er die benötigte Referenzspannung für die Mikrophone zur Verfügung stellt. Die zusätzlich verfügbaren Funktionen wie Schnittstelle für CD-ROM, Line-In, Line-Out und weitere analoge IO's wurden nicht verwendet. Der AD1885 ist über ein schnelles synchrones Interface, das SPORT mit der CPU verbunden. Das zweite SPORT des Blackfin ist verfügbar.

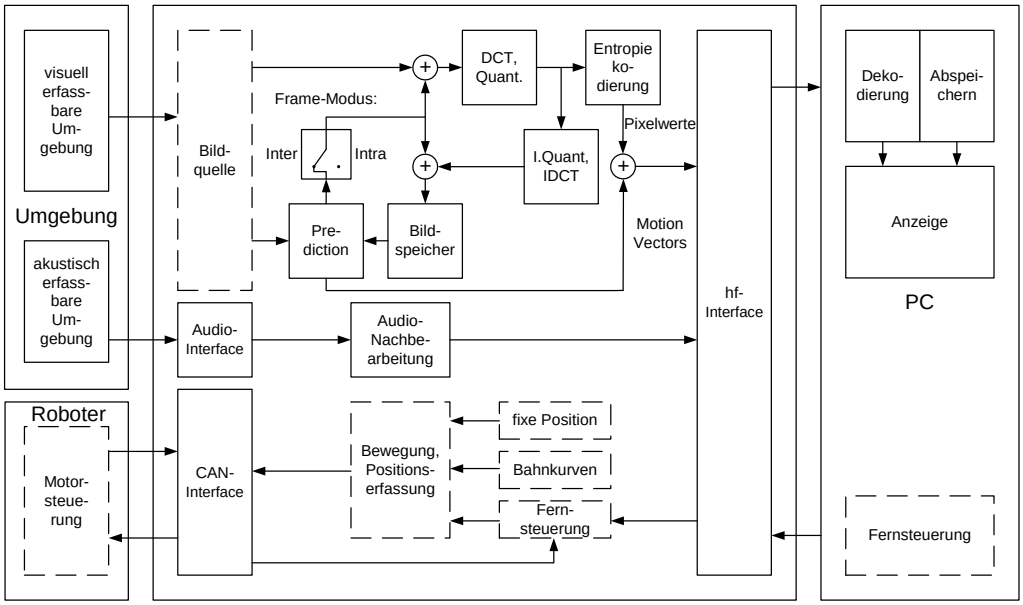


**Abbildung 3.2** Softwarearchitektur

Die beiden Kameras vom Typ OV 7648 FB von Omnivision sind über das PPI, ein paralleles Interface, welches Videoquellen nach ITU-656 Standard unterstützt, angeschlossen. Bilddaten können immer nur von jeweils einer Kamera eingelesen werden, nicht aber gleichzeitig von beiden. Es kann allerdings sehr schnell zwischen den beiden Kameras gewechselt werden.

Das letzte Modul der Hardware ist das nrf2401 Funkmodul von Nordic. Es ist über ein serielles Interface, das SPI mit dem Signalprozessor verbunden. Im Unterschied zu fertigen Standards wie Bluetooth und WLAN ist kein Protokollstack implementiert, ein Protokoll zur Kommunikation mit anderen Modulen muss selbst entwickelt werden.

Die Protokolle, die zur Kommunikation mit anderen Boards am Fussballroboter über CAN und zur Übertragung von Videodaten über Funk entwickelt wurden, sind in den Unterkapitel 6.3.1 und 6.3.2 beschrieben. Unterkapitel 6.3.3 gibt eine Beschreibung des Interfaces zum Audio Codec.

Die Datenblätter der verwendeten Komponenten können den Homepages der Hersteller entnommen werden, welche sich im Internetreferenzen-Verzeichnis finden.

Abbildung 3.2 skizziert das Softwarekonzept, wobei der mittlere Teil Module bezeichnet, die auf der Platine, also vom Signalprozessor ausgeführt werden. Der rechte Teil stellt Module dar, die auf einem PC oder Laptop ausgeführt werden können und links sind die von der Plattform erfassbare Umgebung und der Roboter Tinyphoon dargestellt. Strichliert umrahmte Module sind Module, die von externen Quellen stammen.

Im oberen Bereich der Abbildung ist nochmals das Konzept eines generischen Hybridenkoders dargestellt, wie er in Kapitel 2.2 beschrieben wurde. Die Bilddaten werden dem hf-Interface übergeben, welches sie per Funk an einen Empfänger (normalerweise einem PC oder Laptop, kann aber auch eine gleichartige Plattform mit Display statt

Kamera sein) überträgt. Laut Aufgabenstellung werden die Bilddaten vom PC dekodiert und angezeigt, optional kann man auch Daten zur wiederholten Anzeige abspeichern.

Im Mittelteil der Abbildung ist dargestellt, dass Audiodaten über ein Interface vom Audio Codec eingelesen werden können. Diese können dann noch nachbearbeitet und über das hf-Interface (oder über CAN) übertragen werden.

Der untere Teil der Abbildung zeigt Module, die für die Verwendung der Plattform beim Projekt Tinyphoon entwickelt wurden. Der Roboter lässt sich entweder fernsteuern, fährt vorprogrammierte Bahnkurven ab, oder bleibt wie eine Videokamera einfach ruhig stehen.

Die Daten der Fernsteuerung können über das eigene hf-Modul oder über ein anderes Funk-Modul übertragen werden, welches sich noch am Roboter befindet und die Befehle dann über CAN weiterleitet. Die Routinen, welche die Befehle an die Motorsteuerung errechnen sind sog. Positionsfunktionen und können dabei vom Roboterfussball direkt übernommen werden.

# Kapitel 4

# Implementierung des Video-kompressions-Algorithmus

Es wurde schon mehrfach darauf hingewiesen, dass der programmierte Video-kompressionsalgorithmus sich sehr stark an Standards wie MPEG und H.261 anlehnt, diese ab nicht vollständig implementiert. Das hat folgende Gründe:

- Standards müssen sehr flexibel sein, haben daher einen Overhead an Daten, die zu spezifizieren sind. Fertige Codecs können beispielsweise verschiedene Eingangs-Datenformate lesen, unterstützen verschiedene Bitraten, Bildraten und -größen.
- Die fertigen Codecs sind für PC's mit Intel Pentium kompatiblem Prozessor optimiert. Die Spezialbefehle für die on-Chip-Peripherals des Pentium müssten dem Blackfin angepasst werden.
- Betriebssystemaufrufe müssten angepasst oder umgeschrieben werden.
- Der MPEG Standard ist nicht für Echtzeit-Datenübertragung geeignet, da bei der Berechnung bzw. Dekodierung von B-Frames sehr viel Zeit verloren geht. Alle Verweise, Hinweise, Aufrufe, Variablen, usw., die etwas mit B-Frames zu tun haben, müssten entfernt werden.
- Die im MPEG Standard definierten D-Frames, die in jedem Block nur den Gleichanteil enthalten und für den schnellen Vorlauf verwendet werden, müssten ebenfalls entfernt werden.
- Synchronisierungskonstrukte wie Timestamps oder der komplette System-Datastream können weggelassen werden, ebenso die Bildnummer innerhalb einer Bildserie, da die Übertragungsreihenfolge mit der Anzeigereihenfolge übereinstimmt.
- Man muss sich sehr lange in eine fertige Software einlesen und
- Open Source Projekte sind oft nicht oder nicht ausreichend kommentiert. (Ausnahmen bestätigen die Regel!)
- Dazu kommt oft noch, dass ein fremder Programmierstil verwirrend erscheinen mag oder gar nicht durchschaut wird

- Eine mögliche vorhandene GUI müsste entfernt werden und das in ihr durchgeführte Eventmanagement in C umgeschrieben oder an den Blackfin angepasst werden.
- Manche Erweiterungen von MPEG-2 und -4 gegenüber MPEG sind zu rechenintensive, um in Echtzeit durchgeführt zu werden. Dazu gehören Advanced Prediction, Half Pel Correction und New Chromina-Sampling oder die in MPEG-4 eingeführte Alpha-Plane.
- Bei jeder neuen Version der Standards MPEG und H.261 werden mehr Metadaten in die Datei übernommen. Diese müssen hier aber, weil bekannt, nicht mitgespeichert werden.

Die folgenden Kapitel beschreiben die Entwicklung des Algorithmus, grob gegliedert in Kompression von Einzelbildern, Erweiterung zur Kompression von Videosequenzen, beides am PC, und abschließender Portierung auf den Signalprozessor mit Geschwindigkeitsoptimierung.

## 4.1 Bildkompressionsalgorithmus

Bei der Entwicklung des Bildkompressionsalgorithmus habe ich mich an den JPEG-Algorithmus gehalten. JPEG offeriert verschiedene Methoden der Kompression, das am meisten verwendete Verfahren ist das sog. Baseline-JPEG, das in Abbildung 4.1 dargestellt ist.

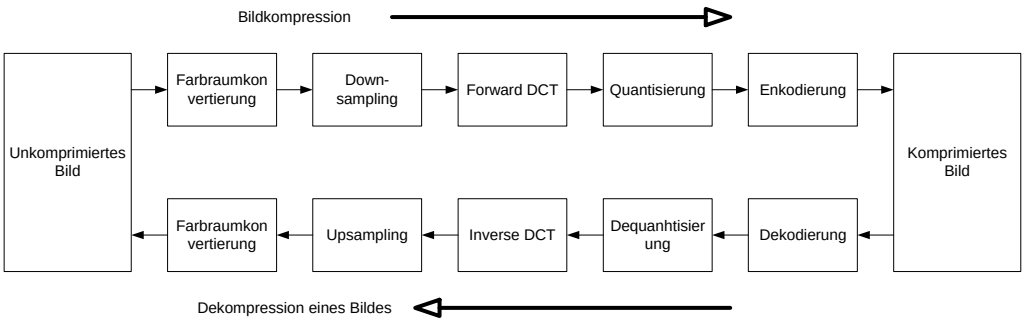


**Abbildung 4.1** Blockdiagramm des Baseline-JPEG

Die Implementierung des Baseline-JPEG mit den dazugehörigen Funktionsaufrufen zeigt Abbildung 4.2.

Der C-Code ist in vier Source-Files und vier Header-Files gegliedert. Die Struktur der beiden Hauptmodule, pack und unpack, wird am Einfachsten durch einen Auszug der dazugehörigen Header-Files beschrieben, anschließend erfolgt eine Kurzbeschreibung aller im Source-Code vorkommenden Methoden.

### 4.1.1 Modul pack

```
class pack
```

```
{
public:
  void compress(void);
  void readData(void);
private:
  void Q(double**, bool, short*);
  void ZZ(short*);
  void RLC(short*, bool, bool);
  void pushBit(short*, bool);
  int Huff( bool, bool, short, short, short, unsigned long*);
};
```

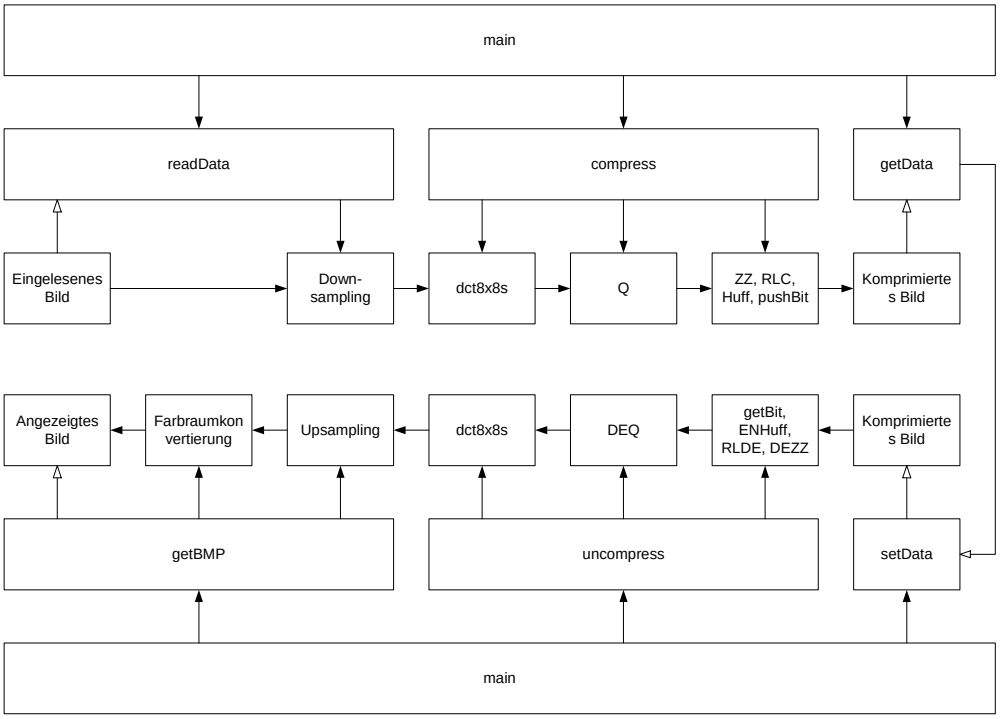


**Abbildung 4.2** Funktionsaufrufe bei der Implementation des Baseline-JPEG

Die Funktionen werden in folgender Reihenfolge aufgerufen:

1. readData
2. compress
    a. dct8x8s
    b. Q
    c. ZZ
    d. RLC
    e. Huff
    f. pushBit

Die von mir verwendeten Kameras liefern ein Bild im YUV 4-2-2 Format, d.h. es werden für zwei nebeneinander liegende Pixel folgende vier Bytes übertragen:

$$UY_1VY_2$$

$Y_i$ sind dabei die beiden Luminanzwerte, U und V die Chrominanzen, die schon gemittelt sind.

Eine Farbraumkonversion ist deswegen nicht mehr erforderlich.

Die Rohdaten der Kamera werden zeilenweise geliefert, d.h. ein weiteres Downsampling kann erst erfolgen, wenn immer zwei aufeinander folgende Zeilen eingelesen wurden. Das Einlesen der Daten und das Downsampling erfolgt in der Funktion `readData`. Die anschließend aufgerufene Funktion `compress` arbeitet schließlich das Bild Block für Block ab und führt die Kosinustransformation, Quantisierung, Zic-Zac-Scan, Lauflängenkodierung und Entropiekodierung nach den JPEG Standardkodiertabellen aus.

Der Code der Kosinustransformation stammt von der Homepage von Takuya Ooura, [W3COS], auf der man den "am schnellsten ausführenden Code weltweit" für Kosinus-, Sinus- und Fouriertransformationen ein- oder mehrdimensional, real oder komplex findet.

### 4.1.2 Modul unpack

```
class unpack
{
public:
  void uncompress(void);
  void setData(unsigned long);
  unsigned char* getBMP(void);
private:
  void getBits(short*, bool);
  int ENHuff( bool,  bool,  short*,  short*,  short*,  unsigned
              long);
  void RLDE(short*, bool, bool, short*);
  void DEZZ(short*);
  void DEQ(short*, double*, bool);
};
```

Die Funktionen werden in folgender Reihenfolge aufgerufen:

3. setData
4. uncompress
    a. getBits
    b. ENHuff
    c. RLDE
    d. DEZZ
    e. DEQ
    f. dct8x8s

Die Funktion `setData` legt einen Zeiger auf das im Speicher stehende komprimierte Bild. Die anschließend aufgerufene Funktion `uncompress` arbeitet das Bild Block für Block ab und führt dabei die Umkehrung der Huffman- und Lauflängenkodierung und des Zic-Zac-Scan, die inverse Quantisierung und schließlich die inverse Kosinustransformation aus.

### 4.1.3 Ergebnisse

Die folgenden Abbildungen zeigen das Ergebnis, einmal als Bitmap mit einer Bildgröße von 226 kByte (die Rohdaten der Kamera haben 153,6 kByte), danach das komprimierte Bild, welches wieder in ein Bitmap umgewandelt wurde. Das komprimierte Bild benötigt genau 66.220 Bit (8,7 kByte) bei einem Kompressionsfaktor von 50. Das dritte Bild zeigt das Kompressionsergebnis mit dem Standard-JPEG Verfahren, dieses Bild hat 19,6 kByte. Die meisten Programme haben einen Kompressionsfaktor von 80 bis 90 voreingestellt.

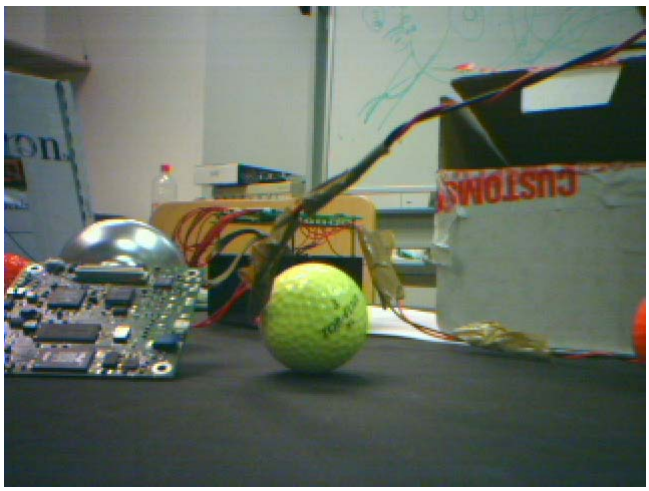

**Abbildung 4.3** Unkomprimiertes Testbild (153,6 kByte)

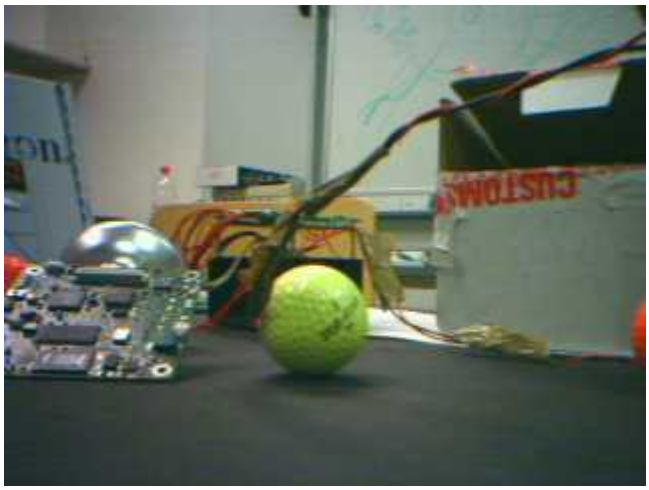

**Abbildung 4.4** Kompressionsergebnis mit q = 50 (8,7 kByte)

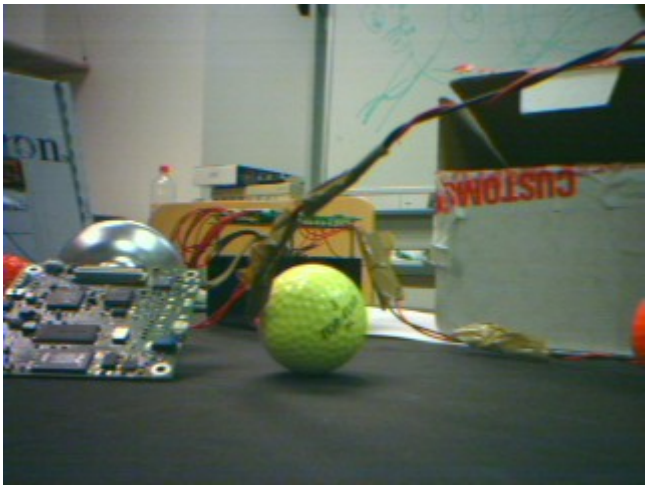

**Abbildung 4.5** Kompressionsergebnis eines Standardprogramms (q = 80; 19,6 kByte)

Die Abbildungen entsprechen ungefähr der angezeigten Bildgröße, da die Bilder eine Auflösung von 320x240 Pixel haben. Bei doppelter Vergrößerung (640x480) des mittleren Bildes kann man rund um den Kabelstrang rechts oberhalb der Bildmitte ganz kleine, so genannte Artefakte erkennen, die aber bei bewegten Bildern nicht mehr wahrgenommen werden können.

Wenn man von einer gewünschten Bildrate von 15 Bildern bei der dem Funkmodul möglichen Bandbreite von 300 kBit/s ausgeht, dürfte jedes Bild im Durchschnitt 20 kBit groß sein, das Bild hat aber 66 kBit. Dieses Rechenbeispiel zeigt deutlich, dass nur mit Ausnutzung der räumlichen Redundanz noch kein optimales Kompressionsergebnis erzielbar ist.

Die restlichen beiden Module **dct** und **main** beinhalten die Kosinustransformation sowie ein Hauptprogramm zur Ausführung der beiden oben vorgestellten Module. Die Header-Files **dct** und **quant** beinhalten Konstanten für die Kosinustransformation sowie die Standard-Quantisierungsmatrizen des Baseline-JPEG.

### 4.1.4 Funktionsbeschreibung

Die Kurzbeschreibung der Funktionen erfolgt für Module getrennt und innerhalb der Module in der Reihenfolge des Aufrufens.

*(i) Modul Main*

Um das Kompressionsergebnis auch anzeigen zu können, muss ein komprimiertes Bild dekomprimiert und anschließend in ein anzeigbares Format gebracht werden. Das einfachste nur denkbare Dateiformat für Bilder ist das Bitmap-Format. Es besteht aus einigen Bytes Header gefolgt von jeweils drei Byte RGB-Daten pro Pixel.

Um den Bitmap-Header richtig in eine Datei schreiben zu können, muss man bei modernen Programmiersprachen den Compiler darauf hinweisen, dass die folgenden Daten Byte-aligned sind, d.h. der Compiler darf einzelne Bytes nicht mit Zeros stuffen, um ganze Wörter in die Datei zu schreiben. Der C-Code für den Bitmap-Header, der aus dem eigentlichen Header und dem Infoheader besteht, sieht folgendermaßen aus:

```
#pragma pack(push, 1)

typedef struct {
   unsigned short type;        // Magic identifier
   unsigned int size;          // File size in bytes
   unsigned short reserved1, reserved2;
   unsigned int offset;        // Offset to image data, bytes
} HEADER;
typedef struct {
   unsigned int size;              // Header size in bytes
   int width, height;              // Width and height of image
   unsigned short planes;          // Number of colour planes
   unsigned short bits;            // Bits per pixel
   unsigned int compression;       // Compression type
   unsigned int imagesize;         // Image size in bytes
   int xresolution, yresolution;   // Pixels per meter
   unsigned int ncolours;          // Number of colours
   unsigned int importantcolours; // Important colours
} INFOHEADER;

#pragma pack(pop)
```

Die genaue Bedeutung und die zulässigen Werte der einzelnen Angaben können z.B. auf der Website der … nachgelesen werden.

Im Hauptprogramm `int main()` wird zuerst die tatsächlich verwendete Quantisierungsmatrix berechnet, anschließend werden die Werte des Bitmap-Headers gesetzt. Dann werden Kompression und Dekompression aufgerufen und schließlich das Ergebnis in eine Datei geschrieben.

### *(ii) Modul dct*

Wie schon erwähnt, stammt die DCT von einer externen Quelle.

Dieses Unterprogramm berechnet die 8x8-Punkt diskrete Kosinustransformation für ein 8x8-double-Array von Eingangswerten und liefert das Ergebnis an Stelle der Originalwerte. Der Parameter `int isgn` gibt an, ob es sich um die Forward DCT oder Backward DCT handelt.

### *(iii) Modul pack*

```
pack::pack(int width, int height, int format, char
*filename1, unsigned char *Y_quant, unsigned char
*C_quant)
```

Konstruktor, der die Werte Höhe, Breite, Bildformat, Quelldatei und die beiden Quantisierungsmatritzen übernimmt.

```
void pack::readData()
```

Ließt die Quelldaten immer für zwei Pixel auf einmal ein. Die eingelesenen Bytes haben die Reihenfolge $UY_1VY_2$, wobei Y für Luminanz, U für Chrominanz-blau und V für Chrominanz-rot steht. Bei jeder zweiten Zeile werden die Chrominanzen mit denen der vorigen Zeile gemittelt und nur ein Wert für vier Pixel gespeichert.

```
void pack::ReadHex(char* _Hex, unsigned char *dest)
```

Wird von `readData()` aufgerufen, da die Werte in der Datei zwar HEX-Zeichen sind, aber in ASCII-Zeichen gespeichert wurden, um sie wie normalen Text lesen zu können. So würde beispielsweise das Zeichen ´00´ als zweimal 48 interpretiert werden.

```
void pack::compress(void)
```

Die Funktion arbeitet das ganze Bild in einer doppelten Schleife ab. Die innere Schleife arbeitet eine 8 Pixel hohe Blockzeile Block für Block ab, die äußere Schleife springt vertikal immer um 8 Pixel weiter. So ist im Schleifeninneren ein Pixelblock zu behandeln.

Vor dem Aufruf der inneren Schleife müssen die Anfangsadressen der jeweiligen Blockzeile im Speicher ermittelt werden.

In der inneren Schleifen müssen nun nur mehr die richtigen Werte aus dem Array der Pixelwerte in die 8x8 Matrix geschrieben und anschließend die einzelnen Stufen des JPEG Algorithmus ausgeführt werden.

```
void pack::Q(double **data, bool Y, short *dest)
```

Dividiert die in `data` stehenden DCT-Koeffizienten durch ihren zugehörigen Wert in der Quantisierungsmatrix und schreibt das Ergebnis in das Array `dest` mit Länge 64. Welche Standard-Quantisierungsmatrix verwendet wird, lässt sich über `bool Y` angeben.

```
void pack::ZZ(short *data)
```

Ordnet die quantisierten Koeffizienten entsprechend dem Zic-Zac-Scan um.

```
void pack::RLC(short *data, bool Y, bool cb)
```

Führt für die von `ZZ` kommenden Werte die Lauflängenkodierung (Run/Length Coding) durch. Da die DC-Anteile der Koeffizienten im DPCM-Verfahren übertragen werden, muss die Funktion nicht nur wissen, ob es sich um Luminanzen oder Chrominanzen handelt, sondern auch, um welche Chrominanzen.

Es gilt drei Spezialfälle zu beachten:

- Kommen 16 aufeinander folgende Nullen, so werden diese durch das Symbol (15/0) dargestellt.
- Tritt (15/0) dreimal hintereinander auf (48 Nullen), so kann man diese drei Symbole durch das EoB-Symbol (0/0) ersetzen. Anm.: Man geht dann davon

aus, dass nach 48 Nullen kein von Null abweichender Wert mehr kommt, kontrolliert diese Annahme jedoch gar nicht erst.

- Am Ende eines Blocks mus EoB eingefügt werden.

```
void pack::pushBits(short *tmp1, bool Y)
```

Zerlegt die in `tmp1` stehenden Werte in gültige Lauflängensymbole und übergibt diese an `int Huff()`. Dann werden die oberen `h` Werte von `out_stream` an den Ausgangsbitstrom angehängt. `h` ist der Rückgabewert von `Huff`, `out_stream` der an `Huff` übergebene Wert.

```
int pack::Huff(bool DC, bool Y, short s11, short s12,
short s2, unsigned long *result)
```

Diese Funktion ist die bei weitem Längste im Modul pack. Sie sucht aus den JPEG-VLC-Tabellen den Huffman-Code für die in `s11`, `s12` und `s2` stehenden Symbole. Über die beiden Flags `DC` und `Y` wird angegeben, ob es sich um einen DC-Anteil und/oder Luminanzen handelt. Die gefundenen Bits werden in `result` gespeichert, der Rückgabewert der Funktion ist die Anzahl der Bits des gefundenen Wertes.

```
bool pack::getData(unsigned long *d)
```

Setzt den Zeiger d auf das erste Byte des komprimierten Bildes.

*(iv) Modul unpack*

```
unpack::unpack(int width, int height, int format, char
*filename2, unsigned char *Y_quant, unsigned char
*C_quant)
```

Konstruktor, der die Werte Höhe, Breite, Bildformat, Zieldatei und die beiden Quantisierungsmatritzen übernimmt. Die Zieldatei ist dazu gedacht, die dekomprimierten Werte im YUV-Format mit den von der Kamera gelieferten bei verlustloser Kompression zu vergleichen

```
void unpack::setData(unsigned long data)
```

Übernimmt den Zeiger auf das komprimierte Bild.

```
void unpack::uncompress(void)
```

Arbeitet ein Bild in zwei Schleifen ab und führt in der inneren Schleife die Stufen der Dekompression eines Pixelblocks aus.

```
void unpack::getBits(short *tmp1, bool Y)
```

Führt die Umkehrung von `pack::pushBits()` aus. Die Funktion liest 32 Bit des Datenstroms und übergibt sie `int ENHuff()`.

```
int unpack::ENHuff(bool DC, bool Y, short *s11, short
*s12, short *s2, unsigned long data)
```

Führt die Umkehrung von `pack::Huff()` aus. Es ist dies die längste Funktion im Modul, sie liest aus `data` ein gültiges VLC-Zeichen heraus und liefert die RLC-Symbole und die Anzahl der zur Darstellung nötigen Bits.

`void unpack::RLDE(short *data, bool Y, bool cb, short *dest)`

Führt die Umkehrung von `pack::RLC()` aus. Die Funktion wandelt die Lauflängensymbole in `data` in 64 Werte in `dest` um.

`void unpack::DEZZ(short *data)`

Führt die Umkehrung von `pack::ZZ()` aus, d.h. die Koeffizienten der DCT werden wiederhergestellt.

`void unpack::DEQ(short *s, double *d, bool Y)`

Führt die Umkehrung von `pack::Q()` aus. Die Funktion multipliziert die quantisierten Koeffizienten mit dem jeweiligen Eintrag in der Quantisierungsmatrix, das Ergebnis wird als double-Wert in `d` geliefert.

`unsigned char* unpack::getBMP(void)`

Nach der Ausführung von `void uncompress()` liegen die Bilddaten im YUV-4-2-2 Format vor. Diese Funktion führt ein Upsampling der Chrominanzwerte durch und errechnet schließlich die RGB-Werte der einzelnen Pixel. Die zulässigen Wertebereiche werden kontrolliert und anschließend ein Zeiger auf ein Array zurückgegeben, in dem die RGB-Werte für die Abspeicherung als BMP gespeichert sind.

## 4.2 Erweiterung zum Videokompressionsalgorithmus

Um einen Einzelbild- in einen Videokompressionsalgorithmus umzuschreiben, ist es notwendig, von der vorher vorhandenen Symmetrie Abstand zu nehmen. Es ist ja, wie schon in Abbildung 2.17 und Abbildung 3.2 klar erkennbar, der Decoder komplett im Enkoder enthalten. Eine weitere wesentliche Änderung stellt die Einführung einer Makroblock-Struktur da, die vorher nicht notwendig war. Wo ein Bild vorher aus drei Arrays mit Luminanz- und Chrominanzwerten bestand, herrscht jetzt eine Struktur, die aus GOP, Picture, Makroblock, Block und Wert besteht, vor. Diese Struktur ist notwendig, um die neuen Verfahren überhaupt sinnvoll durchführen zu können. Außerdem müssen noch ein Motion Vector und ein Picturetyp eingeführt werden, um allen Notwendigkeiten der Videokompression zu genügen. Die Struktur ist in folgendem C-Code wiedergegeben:

```
struct block
{
  short pixels[8][8];
};
struct Motionvector
{
  char x, y;
};
struct macroblock
{
  block Y[4];
```

```
  block Cb, Cr;
  Motionvector mv;
};
struct searchwindow
{
  macroblock mb[3][3];
};
enum PictureType
{
  I = 1,
  P
};
struct picture
{
  macroblock **mb;
  PictureType pt;
};
struct GOP
{
  picture p[15];
};
```

Der Algorithmus ist wie schon bei der Vorgängerversion in vier C-Files und vier H-Files gegliedert, die Namensgebung hat sich ebenfalls nicht geändert. Das Header-File **dct** wurde um die oben abgebildete Struktur erweitert, **quant** um die MPEG-Standard-Quantisierungsmatritzen und einige VLC's, die ebenfalls dem MPEG-Standard entnommen sind.

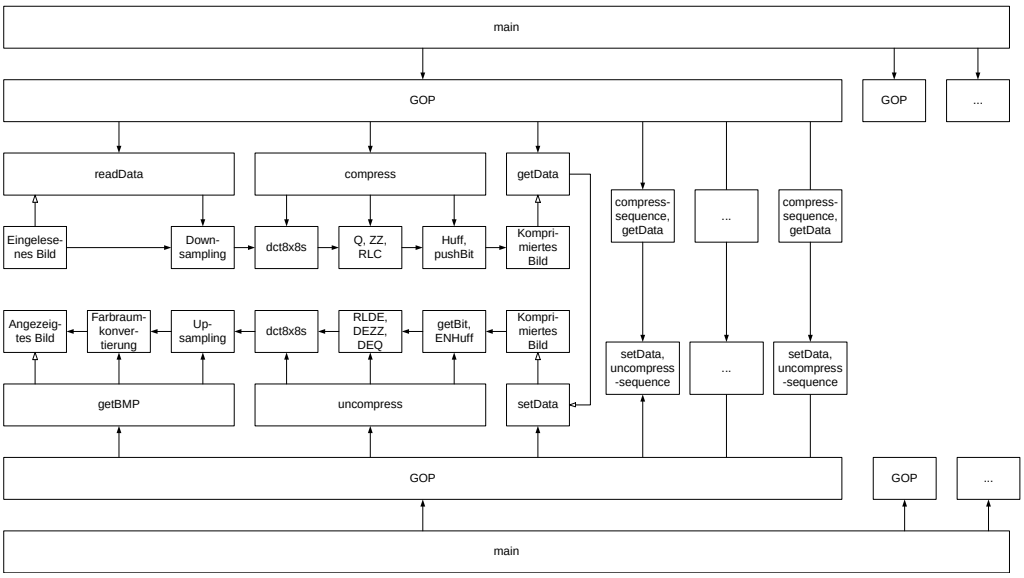


**Abbildung 4.6** Funktionsaufrufe bei der Implementation des Videokompressionsalgorithmus

Das Source-File **dct** bleibt unverändert, **main** wurde um die Routine GOP() erweitert, die die Kompression einer Gruppe aufeinander folgender Bilder während einer Sekunde übernimmt. Die restlichen beiden Module wurden jedoch, wie unten beschrieben, sehr stark verändert.

Abbildung 4.6 zeigt die Funktionsaufrufe während der Ausführung des Videokompressionsalgorithmus.

### 4.2.1 Modul pack

```
class pack
{
public:
  void compress(picture, bool);
  picture readData(char*, unsigned char);
  void compress_Sequence(unsigned char);
  void createPredicted(void);
private:
  void defHuff(short, short, short, unsigned long*);
  void Q(double**, short*, bool);
  void DEQ(short*, double*, bool);
  void ZZ(short*);
  void RLC(short*, bool, bool);
  void pushBit(short*, bool);
  int Huff( bool, bool, short, short, short, unsigned long*);
  int calDiff(macroblock, block*, block, block);
  void MotionCompensation(picture);
  void MotionEstimation(picture);
  void sendDVF(picture);
  void sendMV(Motionvector);
  void findME_TSS(macroblock*, searchwindow);
  picture calPicDiff(picture, picture);
};
```

Die Methode `compress_Sequence` bildet die Statemachine, die für die Kompression eines Bildes unter Zuhilfenahme des Vorhergehenden zuständig ist.

```
void pack::compress_Sequence(unsigned char pic_i)
{
  picture DFD;
  MotionEstimation(gop.p[pic_i]);
// calculate DVF of new picture (displacement vector field)
  sendDVF(gop.p[pic_i]);
// send DVF of new picture
  MotionCompensation(gop.p[pic_i]);
// build old picture + MV of new one (predict frame)
  DFD = calPicDiff(predicted, gop.p[pic_i]);
// calculate DFD (displaced frame difference)
  compress(DFD, PIC_DIFF);
```

```
// send DFD, which is quantized different than a whole picture
}
```

Die Struktur eines Hybrid-Enkoders wurde in Kapitel 2.2 schon vorgestellt, hier soll sie noch einmal genau erläutert werden.

1. Die ME errechnet aus dem geschätzten (predicted[1]) Bild die Motion Vectors, die unter dem Begriff Displacement Vector Field DVF zusammengefasst werden.
2. Die MC beaufschlagt die Makroblöcke im geschätzten Bild mit ihrem Versatz, um so zum selben Bild wie der Dekoder zu kommen.
3. Die Differenz des gerade Ermittelten und des aktuellen Bildes werden gebildet. Diese heißt displaced frame Difference DFD.
4. Die DFD wird komprimiert und versandt, wobei sie gleichzeitig zum letzten geschätzten und verschobenen Bild addiert wird, um wieder ein geschätztes Bild für den nächsten Durchlauf bereitzuhalten.

Die Methode `compress` hat sich stark verändert, da sie jetzt die Makroblock-Struktur unterstützen muss. Außerdem müssen Macroblock-Increment und coded-block-pattern ermittelt werden und die transformierten und quantisierten Werte müssen wiederhergestellt werden, um ein predicted Frame erhalten zu können.

### 4.2.2 Modul unpack

```
class unpack
{
public:
  void uncompress(bool, unsigned char);
  void uncompress_Sequence(unsigned char);
  void setData(unsigned long, unsigned char);
  unsigned char* getBMP(picture);
  void createPredicted(void);
private:
  void getBits(short*, bool);
  int ENHuff( bool, bool, short*, short*, short*, unsigned
              long);
  void RLDE(short*, bool, bool, short*);
  void DEZZ(short*);
  void DEQ(short*, double*, bool);
  void MotionCompensation(picture);
  void readDVF(picture);
  Motionvector getMV(void);
  unsigned char getMbi(void);
  unsigned char getCbp(void);
};
```

[1] die Namensgebung ist hier sehr unglücklich gewählt worden; das predicted Frame während der Enkodierung ist nicht zu verwechseln mit einem P-Frame

Der Dekoder hat sich gegenüber dem JPEG-Decoder nicht wesentlich geändert. Der Hauptunterschied ist, dass ein Differenzbild in der Methode uncompress_sequence() unter Zuhilfenahme des letzten Bildes wiederhergestellt wird.

```
void unpack:: uncompress_Sequence(unsigned char pic_i)
{
  readDVF(gop.p[pic_i]);
// get DVF of next picture
  MotionCompensation(gop.p[pic_i]);
// build old picture + MV of new one (predict frame)
  uncompress(PIC_DIFF, pic_i);
}
```

Als erstes wird das Feld der Motion Vectors eingelesen, mit dieser Information kann die Motion Compensation das predicted Frame herstellen. Anm.: Die Methode der MC ist genau gleich wie im Enkoder. Anschließend wird das empfangene Differenzbild dekomprimiert und zum predicted Frame addiert, das Bild ist rekonstruiert.

### 4.2.3 Ergebnisse

Die folgenden Abbildungen zeigen die Funktionsweise und das Ergebnis der Enkodierung zweier Bilder. In der ersten und zweiten Abbildung sind die von der Kamera eingelesenen Bilder gezeigt. Abbildung 4.9 zeigt das erste Bild mit einem Kompressionsfaktor von 10 auf 29.738 Bit (3,6 kByte) als I-Frame komprimiert. Nachdem die Motion Vectors übertragen wurden, kann die Motion Compensation ein predicted Frame erzeugen, dieses ist in Abbildung 4.10 dargestellt. Wird jetzt das Differenzbild – welches genau 17.454 Bit (2,1 kByte) inkl. MV benötigt – aus Abbildung 4.11 zum predicted Frame addiert, erhält man das rekonstruierte Testbild 2 in Abbildung 4.12.

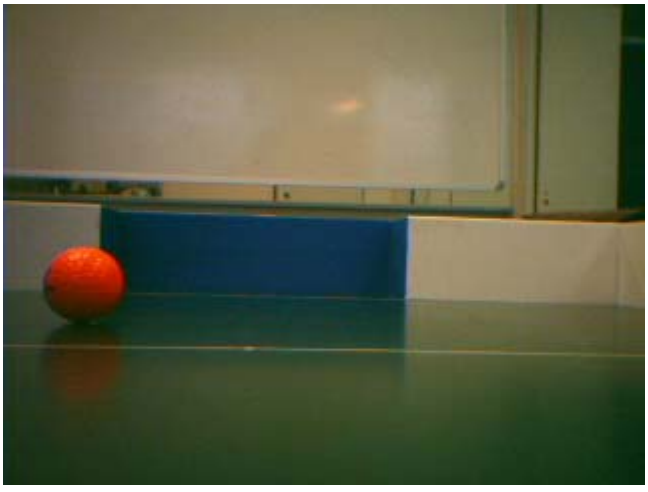

**Abbildung 4.7** Unkomprimiertes Testbild 1

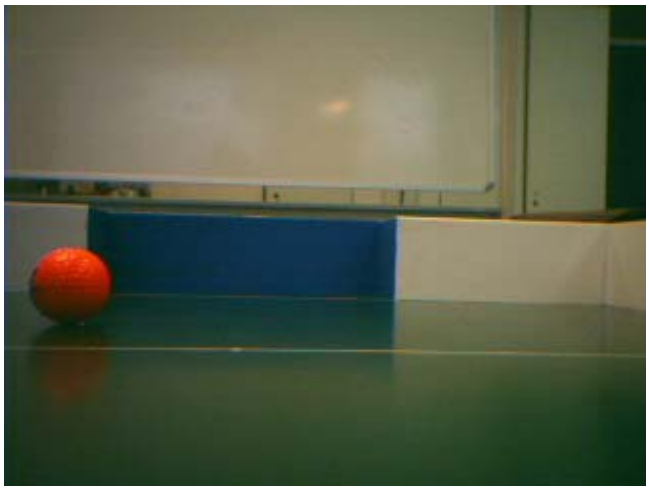

**Abbildung 4.8** Unkomprimiertes Testbild 2

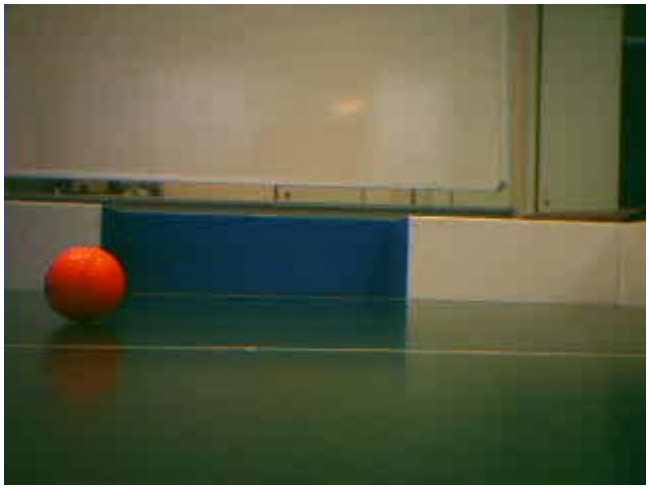

**Abbildung 4.9** Kompressionsergebnis Testbild 1 bei q = 10

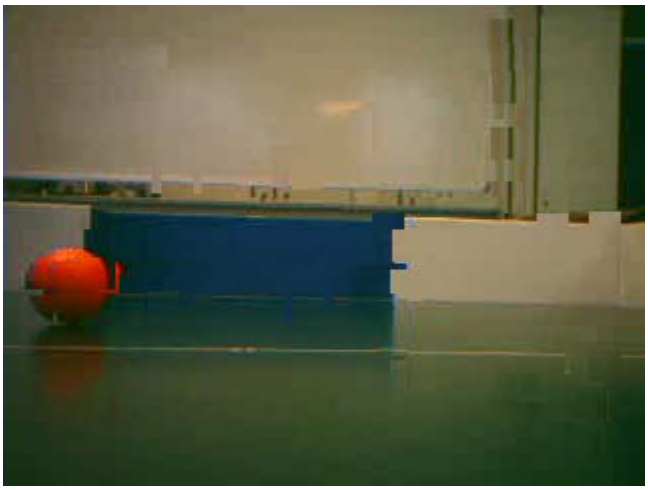

**Abbildung 4.10** Predicted Frame, erzeugt aus Testbild 1 und den Motion Vectors

**Abbildung 4.11** Differenzbild von Testbild 2 und dem predicted Frame

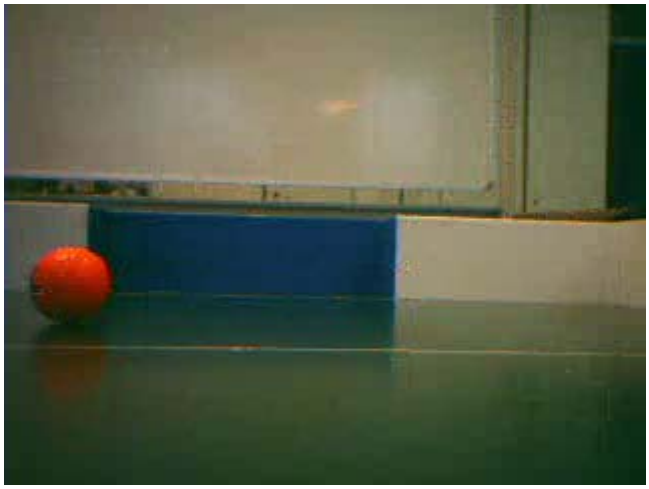

**Abbildung 4.12** Kompressionsergebnis Testbild 2 bei q = 10

### 4.2.4 Funktionsbeschreibung

*(i) Modul Main*

Die Beschreibung des Bitmap-Headers kann 5.1.3.1 entnommen werden.

```
void GOP(void)
```

In dieser Funktion wird das jeweils erste Bild einer Bildergruppe als I-Frame komprimiert. Anschließend wird in einer Schleife das jeweils nächste Bild als P-Frame komprimiert.

*(ii) Modul dct*

siehe 4.1.3

*(iii) Modul pack*

```
pack::pack(int width, int height, unsigned char *I_quant,
unsigned char *BP_quant)
```

Konstruktor, der die Angaben über Breite und Höhe, sowie die beiden Quantisierungsmatritzen übernimmt.

```
picture pack::readData(char *filename, unsigned char
pic_i)
```

Liest ein Bild ein und speichert die Werte in der Makroblock-Struktur. Die Anzahl der Chrominanzen wird dabei mittels Downsampling halbiert.

```
void pack::ReadHex(char* _Hex, unsigned char *dest) und

void pack::ReadHex(char* _Hex, unsigned short *dest)
```

siehe 4.1.3

```
int pack::Huff(bool DC, bool Y, short s11, short s12,
short s2, unsigned long *result)
```

siehe 4.1.3, die VL-Codes wurden dem MPEG-Standard entnommen. Gibt es für ein Symbol keinen Eintrag, wird `void defHuff()` aufgerufen.

```
void pack::defHuff(short s11, short s12, short s2, un-
signed long *result)
```

Im Gegensatz zur Kodierung bei JPEG, definiert MPEG nicht für jedes Run/Length-Symbol einen eigenen Code. Sollte ein Symbol nicht in der Tabelle gefunden werden, muss es speziell dargestellt werden.

```
void pack::compress(const picture pic, bool diff)
```

Diese Funktion arbeitet das Bild Makroblock für Makroblock ab. Im einzelnen Makroblock werden die sechs Blöcke kodiert und die coded_block_pattern ermittelt. Anschließend wird aufgrund des cbp das Makroblock-Increment gesetzt.

Im Falle eines I-Frames werden alle sechs Blöcke gesandt, im Falle eines P-Frames werden zuerst Makroblock-Increment und coded_block_pattern, anschließend die Blöcke, die nicht lauter Nullwerte enthalten, gesandt.

```
void pack::pushBit(short *tmp1, bool Y) und
```

```
void pack::Q(double **data, short *dest, bool diff) und
```

```
void pack::ZZ(short *data) und
```

```
void pack::RLC(short *data, bool Y, bool cb) und
```

```
void pack::DEQ(short *s, double *d, bool diff)
```

siehe 4.1.3

```
void pack::findME_TSS(macroblock *mb, searchwindow sw)
```

Diese Funktion berechnet die Motion Vectors nach dem Three Step Search Algorithmus. Als Kostenfunktion wird `int calDiff` verwendet. Zu beachten ist, dass der Makroblock, mit dem verglichen wird, bei jedem Schleifendurchlauf neu aus dem `searchwindow` zu berechnen ist.

```
int pack::calDiff(macroblock mb_old, block *mb_new_Y,
block mb_new_Cb, block mb_new_Cr)
```

Kostenfunktion zum Vergleich von Makroblöcken. Es wird die MAD (Mean Absolute Difference) zur Berechnung verwendet, wobei die Differenz der Luminanzwerte voll und die Differenz der Chrominanzwerte zum Teil einfließt.

```
void pack::compress_Sequence(unsigned char pic_i)
```

siehe 4.2.1

```
void pack::MotionEstimation(picture act)
```

Berechnet das Suchfenster aus dem aktuellen Frame und ruft den TSS Algorithmus für jeden Makroblock auf.

```
void pack::MotionCompensation(picture act)
```

Verschiebt jeden Makroblock in Frame um seinen Motion Vector, wobei die Makroblöcke vorher gespeichert werden müssen, um nicht von einem Nachbarmakroblock überschrieben zu werden.

```
void pack::sendDVF(picture act)
```

Ruft in einer Schleife für jeden Makroblock `void sendMV()` auf.

```
void pack::sendMV(Motionvector mv)
```

Ermittelt für jeden Motion Vector die VLC's für die vertikale und horizontale Komponente und versendet diese.

```
picture pack::calPicDiff(picture p, picture a)
```

Berechnet die Differenz zweier Bilder, im konkreten Fall zwischen dem aktuellen Frame und dem um die Motion Vectors verschobenen predicted Frame.

```
void pack::createPredicted(void)
```

Erzeugt ein Frame mit lauter Nullwerten.

*(iv) Modul unpack*

```
unpack::unpack(int width, int height, unsigned char
*I_quant, unsigned char *BP_quant)
```

Konstruktor, der die Angaben über Breite und Höhe, sowie die beiden Quantisierungsmatritzen übernimmt.

```
unsigned char* unpack::getBMP(const picture pic)
```

Erzeugt aus dem übergebenen Bild `pic` im YUV-4-1-1 Format ein Bitmap im RGB-Format.

```
void unpack::setData(unsigned long data, unsigned char
pic_i)
```

Setzt einen Zeiger auf die Startadresse des Bildes mit dem Index `pic_i` in der aktuellen GOP.

```
void unpack::uncompress(bool diff, unsigned char pic_i)
```

Dekomprimiert ein I- oder P-Frame. Im Falle eines I-Frames werden alle Blöcke erwartet, im Falle eines P-Frames werden vorher Makroblock-Increment und coded_block_pattern mittels `unsigned char getMbi()` und `unsigned char getCbp()` erwartet.

```
void unpack::getBits(short *tmp1, bool Y) und
```

```
void unpack::RLDE(short *data, bool Y, bool cb, short *dest) und
```

```
void unpack::DEZZ(short *data) und
```

```
void unpack::DEQ(short *s, double *d, bool diff) und
```

```
int unpack::ENHuff(bool DC, bool Y, short *s11, short *s12, short *s2, unsigned long data)
```

siehe 4.1.3

```
void unpack::uncompress_Sequence(unsigned char pic_i)
```

siehe 4.2.2

```
void unpack::readDVF(picture act)
```

Ruft in einer Schleife für jeden Makroblock `unsigned char getMV()` auf.

```
Motionvector unpack::getMV()
```

Ermittelt aus dem Bitstrom die vertikale und horizontale Komponente des Motion Vectors.

```
unsigned char unpack::getMbi()
```

Ermittelt aus dem Bitstrom das Macroblock-Increment.

```
unsigned char unpack::getCbp()
```

Ermittelt aus dem Bitstrom das coded_block_pattern.

```
void unpack::MotionCompensation(picture act) und
```

```
void unpack::createPredicted(void)
```

siehe 4.2.3

## 4.3 Portierung

Da von vorne herein das Ziel war, den Algorithmus auf den Blackfin Signalprozessor zu portieren, wurde in den ersten beiden Programmierphasen großer Wert darauf gelegt, den Code so zu schreiben, dass er auch für den Signalprozessor kompilierfähig ist. Die notwendigen Änderungen für ein funktionierendes Programm waren:

- Umschreiben der readData() Funktion; es muss keine Datei gelesen werden, sondern die Daten stehen als Array im Speicher
- Synchronisation von zwei DMA-Kanälen um den Cache optimal zu nutzen
- Entfernen der ReadHex() Funktionen; die Werte aus dem Speicher müssen nicht mehr umgerechnet werden
- Umschreiben der getData() Funktion; der erzeugte Bitstrom muss über das Funkmodul versandt werden

Da von Analog Devices fertige Programme für die DCT, IDCT und den TSS Algorithmus existieren, wurden diese zur Geschwindigkeitsoptimierung herangezogen.

Die einzige an diesen Teilen des Codes durchgeführte Änderung betrifft den TSS Algorithmus. Die Funktion von Analog führt nach der Berechnung des Motion Vector's noch die sog. Half Pel Correction[2] durch. Da dies jedoch weitere vier Schleifendurchläufe für jeden Makroblock erfordert und MV mit half Pel Auflösung von meinem Dekoder nicht unterstützt werden, wurde dieser Teil des Programms gelöscht.

Die Komprimierergebnisse entsprechen denen in 4.2.3.

[2] Half Pel Correction: Nach dem ein Motion Vector gefunden wurde, werden die Makroblöcke noch um einen halben Pel versetzt in ± x und y Richtung verglichen. Die Half Pel Correction stellt eine Verbesserung von MPEG-2 gegenüber MPEG bei langsam beweglichen Szenen dar, erfordert jedoch aufgrund der Mittelwertbildung für jeden Pel viel Rechenleistung.

# Kapitel 5

# Implementierung der Plattform

Beim Schaltungsentwurf und Routing der Platine und speziell bei der Programmierung der Interfaces gab es einige nicht unwesentliche Schwierigkeiten zu bewältigen. Im folgenden Kapitel soll näher darauf eingegangen werden. Anschließend werden die entstandene Hardware noch einmal vorgestellt und die Interfaces detailliert beschrieben. Die ausführliche Dokumentation der Platine würde den Rahmen dieses Dokuments sprengen, sie wurde deshalb als eigenes Dokument verfasst, [BRU03].

## 5.1 Technische Hürden

Ein großes Problem beim Schaltungsentwurf und Routing der Platine stellte die Neuheit der Komponenten dar. So war das Institut für Computertechnik weltweit der erste Empfänger von Samples des neuen Blackfin Signalprozessors, die Version war Rev 0.1, dementsprechend lange war die dazugehörige Liste der known bugs. Einige für uns (mich und meine Kollegen des Projektteams Tinyphoon) relevanten Einschränkungen sind:

- Es funktionieren nicht alle Boot-Modes
- Die Entwicklungsumgebung kann kein Image des Programms für das ROM erzeugen, wir mussten einen eigenen Bootloader schreiben
- Die Core-Spannung lässt sich nicht einstellen
- Der Prozessor funktioniert mit maximal 400 MHz, der SD-RAM Controller funktioniert mit maximal 80 MHz
- Einige SFR lassen sich nicht schreiben, es ist nur der Reset-Zustand verfügbar

Ein weiteres Problem stellten die von den österreichischen Printplattenherstellern geforderten Routing Rules dar. So gab es im Juni 2003 noch keinen Hersteller, der Vias[1] kleiner als 0,55 mm und Leiterbahnen schmäler als 100 µm herstellen konnte. Der Blackfin wird jedoch in einem BGA Gehäuse mit Ball-Durchmesser 0,5 mm und Ball-Abstand 0,8 mm geliefert. D. h., unter dem Gehäuse des Signalprozessors zwischen den Balls konnte ich keine Vias setzen. Das hat dazu geführt, dass alleine das Routing

[1] Via: Durchkontaktierung

aller relevanten Leitungen des Blackfin ungefähr die Hälfte der Zeit für das Routing der ganzen Platine in Anspruch genommen hat. Abbildung 5.1 zeigt das Ergebnis, man erkennt deutlich, dass von den nicht verwendeten inneren Pins wie H3 oder J2 keiner mehr nach außen hätte gerouted werden können. Ebenso verhält es sich mit den nicht verwendeten Pins in der äußeren Reihe, da – hier nicht gezeigt – sehr viele Leitungen in unmittelbarer Umgebung des Prozessors verlaufen. Die im Bild nicht beschrifteten roten, runden Leiterbahnflächen gehören zu Vias.

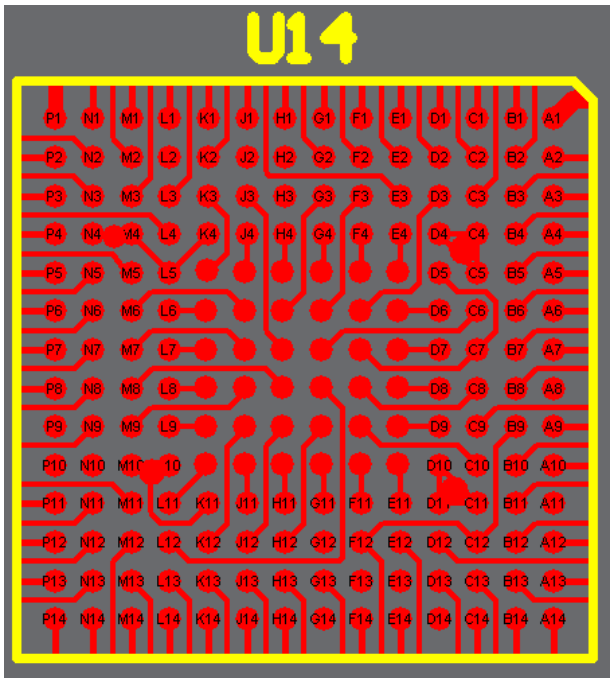


**Abbildung 5.1** Layout unter dem Gehäuse des Blackfin

Ein weiteres, während dem Entwurf nicht erkanntes Problem brachte der Mikrokontroller mit sich. Alle andern in der Schaltung verwendeten Komponenten wie Signalprozessor, Speicher und Funkmodul haben low-active Pins. Die Pins des Mikrokontrollers sind allerdings high-active. Da aber die Pins wie bei den anderen Komponenten angeschlossen wurden, blieb der Mikrokontroller beispielsweise immer im Reset-Zustand. Es mussten daher auf den fertigen Platinen einige wenige Workarounds gelötet werden.

Als Abschlussbemerkung möchte ich anführen, dass manche Hersteller mittlerweile Mikro-Vias fertigen können. Das sind Vias, die auf den äußeren Layern einen Durchmesser von 200 µm und auf den inneren Layern einen Durchmesser von 500 µm aufweisen. Außerdem lassen sich Mikro-Vias direkt unter einen Ball platzieren, da sie kein durchgehendes Loch haben. Konventionelle Vias haben ein durchgehendes Loch, durch das beim Lötvorgang das flüssige Zinn abfließt, so dass das Pad am Gehäuse keinen Kontakt mehr mit der Platine haben kann.

## 5.2 Mechanik

Die folgenden Abbildungen zeigen einen entstandenen Prototypen und den Bestückungsplan mit den wichtigsten Komponenten.

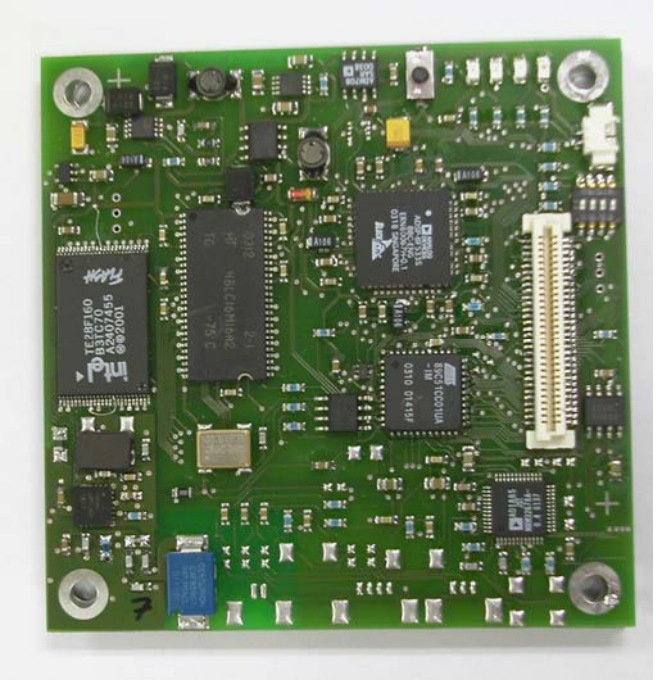

**Abbildung 5.2** Foto der fertigen Platine

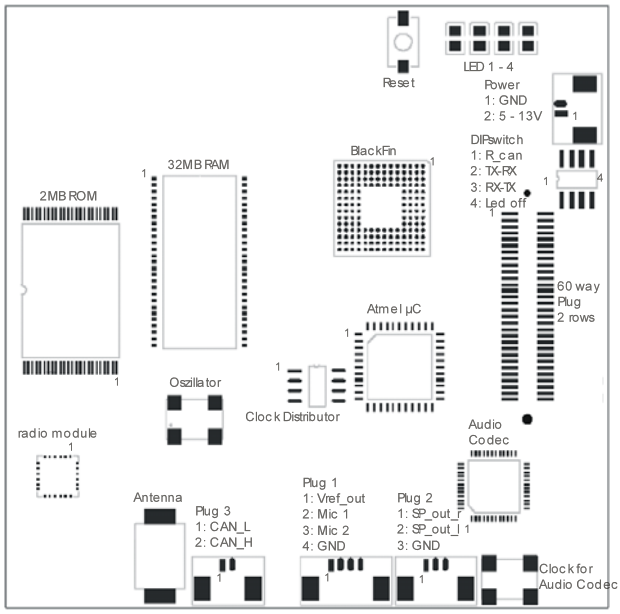


**Abbildung 5.3** Bestückungsplan mit den wichtigsten Komponenten

## 5.3 Interface CAN Controller

Bei der Wahl des CAN Controllers fiel die Wahl aufgrund der größeren Flexibilität auf einen Mikrokontroller mit CAN Controller on Chip obwohl ein Stand Alone CAN Controller gewisse Geschwindigkeitsvorteile gehabt hätte.

Um diese Flexibilität auch nutzen zu können, wurden Pads für eine optional bestückbare Referenzspannungsquelle für den A/D Wandler des Mikrokontrollers vorgesehen, außerdem sind alle 16 Adress/Datenleitungen und 8 A/D Eingänge (oder als GPIO nutzbar) verfügbar. Das Protokoll ist so aufgebaut, dass der Prozessor 32 verschiedene Kommandos unterscheiden kann, alle anderen Werte im typ-Feld werden als „sende Daten weiter“ interpretiert. Von diesen 32 Kommandos sind 8 durch Reservieren und Freigeben von Kanälen und Ein- bzw. Ausschalten der beiden LED’s belegt, die restlichen 24 sind frei erweiterbar, beispielsweise mit Befehlen wie AD Wandler x auslesen oder Daten auf Pin/Port y ausgeben/einlesen.

### 5.3.1 Protokoll

### a) UART

Das Protokoll seitens der seriellen Schnittstelle ist aufgrund folgender Punkte asymmetrisch bezüglich der Paketlängen aufgebaut:

- Die Initialisierungen sind nicht zeitkritisch
- Die Daten sind zeitkritisch und deren Versendung sollte dem Signalprozessor nicht zuviel Rechenleistung abverlangen

*(i) send*[2]

Da der CAN-Bus Paketlängen bis maximal 8 Byte unterstützt und das längste zusammenhängende Datenpaket über den CAN-Bus aus maximal 22 Byte Daten besteht, wurde eine Paketlänge von 24 Byte gewählt. Inklusive MessageID und typ-Feld ergibt sich somit ein Datenpaket wie in Abbildung 5.4 mit einer Länge von 26 Byte.

| ID | typ | L | Data | Dummy-Data |
|---|---|---|---|---|

**Abbildung 5.4** Datenpaket UART – send

ID MessageID

typ Message-typ, folgende Werte sind zulässig:

| | |
|---|---|
| 1 | Reservieren der ID für Empfang |
| 2 | Reservieren der ID zum Versenden |
| 3 oder 4 | Freigeben der ID |
| 5 | Einschalten der LED 1 |
| 6 | Einschalten der LED 2 |
| 7 | Ausschalten der LED 1 |

[2] vom Signalprozessor aus gesehen

| | |
|---|---|
| 8 | Ausschalten der LED 2 |
| 20 – 255 | Weiterleiten der Daten über CAN<br>Nur beim Versenden ist das L(änge) Feld von Bedeutung |
| L | Länge des Datenpakets inklusive L in Byte [1..24] |
| Data | Zu versendende Daten, wenn typ einen Wert zwischen 32 und 255 inklusive hat, sonst ohne Bedeutung |
| Dummy-Data | Daten ohne Bedeutung, dienen nur zum Auffüllen des Pakets auf 26 Byte |

*(ii) receive*[3]

Da der Mikrocontroller immer 8 Byte Pakete über den CAN-Bus empfängt und an Rechenleistung dem Signalprozessor deutlich unterlegen ist, werden empfangene Daten mit ihrer MessageID, wie in Abbildung 5.5 gezeigt, direkt an den Signalprozessor weitergeleitet, der die Daten dann, wenn notwendig, zu einem Gesamtpaket verbindet.

| ID | Data |
|---|---|

**Abbildung 5.5** Datenpaket UART – receive

| | |
|---|---|
| ID | MessageID |
| Data | empfangene Daten (8 Byte) |

Das übergeordnete Protokoll interpretiert das erste Byte des jeweils (im Gesamtpaket, dass aus bis zu 3 Paketen bestehen kann) ersten Pakets als Länge und wartet, wenn notwendig, auf weitere 8 Byte Pakete.

## b) CAN

CAN ist ein standardisiertes Protokoll, bei dem pro Message bis zu 8 Datenbytes übertragen werden. Es wurde für den Einsatz in Fahrzeugen entwickelt und unterscheidet sich von anderen Bussystemen vor allem durch die Adressierung. Beim CAN Protokoll wird nicht jedem Knoten eine eigene Adresse zugewiesen, sondern die Adressierung erfolgt nachrichtenorientiert. D.h. jede Art von Nachricht erhält eine eigene ID und jeder Knoten kann auf ID's, die ihn betreffen, lauschen. Der große Vorteil dieser Adressierungstechnik zeigt sich in Fahrzeugen folgendermaßen: Wenn ein Bauteil ausfällt, muss man nicht ein Originalersatzteil dieses Herstellers mit gleicher Adresse verwenden, sondern man kann auch einen Nachbau verwenden, dieser hat ja, weil nur die Art der Nachricht eine Adresse erhält, im Gesamtsystem dieselben Nachrichten zu bearbeiten oder zu generieren.

Die erste Version des CAN Bus hatte nur 11 Bit zur Adressangabe, mittlerweile hat man die Adresse auf 29 Bit erweitert.

Das CAN Protokoll legt besonderen Wert auf Fehlermanagement, aufgrund dessen ist auch die Nettodatenrate mit ~57% eher bescheiden. Abbildung 5.6 zeigt den Aufbau eines CAN Datenpaketes, genauere Informationen finden sich auf [W3CAN]. Infor-

[3] vom Signalprozessor aus gesehen

mationen über verschiedene Bussysteme finden sich in [DIE94] und über verschiedene Protokolle gibt [LIN00] Auskunft.


**Abbildung 5.6** CAN-Datenpaket

| | |
|---|---|
| SOF | Start of Frame |
| Identifier | 11 oder 29 Bit MessageID |
| RTR | Remote Transmission Request |
| IDE | Identifier Extension |
| DLC | Data Length Code |
| Data | 0 bis 8 Byte Nutzdaten |
| CRC | Cyclic Redundant Check |
| ACK | Acknowledge field |
| EOF | End of Frame |
| IFS | Intermission Frame Space |

### 5.3.2 Programm

### a) Geschichte

Das Programm in seiner ersten Version wurde rein in C in Anlehnung an die von Atmel zur Verfügung gestellten Bespielcodes geschrieben. Die Entwicklung, im speziellen Debuggen und Testen, gestalteten sich als sehr problematisch.

Aufgrund einiger Fehler in den ersten Hardwareaufbauten funktionierten z.B. die Resetbeschaltung, die Bustreiber des CAN-Bus und die Betriebsart (active mode vs. bootloader mode) des Prozessors nicht. Ein weiteres Problem stellt die serielle Schnittstelle eines PC's dar, die sich mit maximal 115 kbaud öffnen lässt. Da der Prozessor über keine Debug-Schnittstelle verfügt, mussten alle Debug-Werte über die serielle Schnittstelle übertragen werden.

Nachdem die Hardwareprobleme gelöst wurden, wobei ein wesentlicher Punkt war, dass mindestens ein Abschlusswiderstand am CAN-Bus angeschlossen sein muss, funktionierte die erste Version des Programms auf dem CAN-UART-USB Board. Der CAN-Bus war auf eine Übertragungsrate von 1 Mbaud eingestellt, die UART (fast UART via USB) sogar auf 1,25 Mbaud.

Dieses Programm funktionierte am Roboter jedoch mit maximal 38,4 kbaud auf der UART. Das Problem war die komplexe Programmierung (Ringbuffer im langsamen XRAM, C).

Die zweite Version des Programms wurde daher komplett in Assembler implementiert, die Ringbuffer wurden durch statische Datenbereiche im schnellen IRAM ersetzt, die ISR bearbeitet nicht mehr einzelne Bytes, sondern eine bestimmte Anzahl auf einmal um Interruptroutinenaufrufe zu sparen. Die Kanäle des CAN-Controllers wur-

den fix vergeben, nicht mehr dynamisch. Diese komfortlose Version unterstützt ein Forewarden von 8 Byte Paketen in beide Richtungen, wobei die UART mit bis zu 625 kbaud betrieben werden kann.

In der letzten Version des Programms wurde die zeitkritische UART-ISR in Assembler implementiert, der Rest jedoch in C. Die Kanäle des CAN-Controllers können jetzt dynamisch allokiert und wieder freigegeben werden und der Prozessor hat einen Ringbuffer zum Versenden von Daten über CAN mit der Größe 16x8 Byte implementiert, d.h. der Signalprozessor kann bis zu 16 Byte Daten ohne Verzögerung an den Mikrocontroller schicken, wenn gleichzeitig am CAN-Bus keine Daten kommen.

## b) Funktionen im Mikrokontroller

Der Mikrokontroller wird, wie die meisten auf der 8051-Architektur basierenden Prozessoren, mit dem C-Compiler der Firma Keil programmiert. Genaue Informationen zu diesem Produkt finden sich auf der Homepage von Keil, [W3KEI].

### *(i) Modul main*

```
void init()
```

Initialisiert den CAN-Controller, die UART und den X2-Mode des Mikrokontrollers.

```
void main()
```

Ruft die Initialisierung auf und pollt anschließend, ob Daten via UART angekommen sind und interpretiert diese (typ; Befehl vs. Weiterleitung der Daten). Sind Daten über den CAN-Bus zu versenden, werden diese in 8 Byte große Pakete aufgeteilt und der Funktion `void CanSendIsr()` übergeben.

```
void UART_command()
```

Werden die Daten nicht direkt weitergeleitet (typ [20..255]), dann wird der Befehl ausgeführt (Kanal frei- oder umschalten, LED ein- oder ausschalten).

### *(ii) Modul can*

```
void CAN_init()
```

Initialisiert alle SFR des CAN-Controllers inklusive Baudrate.

```
void CAN_initChannel(unsigned char ID_channel, bit rec)
```

Wählt den nächsten freien der 15 CAN-Kanäle und weist ihm die angegebene MessageID zu. Bit rec gibt an, ob der Kanal Daten empfängt („1“) oder versendet („0“). War die MessageID schon vergeben, ändert sich jedoch die Übertragungsrichtung, werden die Register entsprechend dem neuen Befehl neu gesetzt.

```
void CAN_clearChannel(unsigned char ID)
```

Überprüft, ob die MessageID tatsächlich vergeben war. Wenn ja, werden alle entsprechenden Register zurückgesetzt.

```
void CanInterrupt() interrupt 7 using 1
```

ISR, die nach jedem Versenden bzw. Empfangen von Daten aufgerufen wird. Kopiert empfangene Daten direkt in den Transmitbuffer der UART-ISR und verwaltet den Sende-Ringbuffer.

Aufgetretene Fehler am CAN-Bus können auch hier behandelt werden.

```
void CanSendIsr(unsigned char ID,  unsigned char *p)
```

Versendet 8 Byte Pakete über den CAN-Bus. Ist der Bus gerade belegt, wird das Paket in den Ringbuffer gelegt.

*(iii) Modul uart*

```
void UART_init()
```

Initialisiert alle SFR der UART.

```
void UARTInterrupt() interrupt 4 using 0
```

ISR, die bei jedem Versenden bzw. Empfangen von Daten aufgerufen wird. Die Routine wird nicht verlassen, bevor nicht alle 26 Byte empfangen bzw. 9 Byte gesendet wurden. Setzt nach dem Empfang einer Nachricht ein Flag, dass von `void main()` gepollt wird.

## c) Funktionen im Signalprozessor

*(i) Modul Can_Config*

```
void Send_to_Can(unsigned char ID, unsigned char typ, unsigned char s[BUF_LEN])
```

Sendet das vorbereitete, 26 Byte Datenpaket über die UART an den Mikrocontroller.

```
void Can_init()
```

Initialisiert Variablen, reserviert die voreingestellten Kanäle zur Datenkommunikation am Roboter.

```
void CAN_Setup(unsigned short SystemClk)
```

Da sich der Mikrocontroller nach Einschalten der Betriebsspannung im Bootloader-Mode befindet, wird die UART entsprechend konfiguriert und der Befehl zum Wechsel in den Active-Mode gesendet (mit dem Aufruf von AtmelStart()). Anschließend wird die UART umkonfiguriert und `void Can_init()` aufgerufen.

```
void CAN_send(unsigned char ID, unsigned char length, unsigned char s[BUF_LEN - 1])
```

Überbleibsel aus Zeiten des 9 Byte Datenpaketes. Leitet jetzt die Nachricht nur an `void Send_to_Can()` weiter.

```
void CAN_reassembly()
```

Je nach Länge der empfangenen Nachricht wird auf weitere Einzelpakete gewartet. Ist eine Nachricht vollständig empfangen, wird sie an `void CANReceiveHandler()` im Modul Command_Interface übergeben.

```
void AtmelStart(void)
```

Sendet den Befehl zum Wechseln vom Bootloader-Mode in den Active-Mode an den Mikrocontroller.

```
void CAN_initChannel(unsigned char ID, bool receive)
```

Sendet den Befehl zum Reservieren eines Kanals.

```
void CAN_setLED(bool one)
```

Sendet den Befehl zum Einschalten einer LED.

```
void CAN_closeChannel(unsigned char ID, bool receive)
```

Sendet den Befehl zum Freigeben eines Kanals.

```
void CAN_clearLED(bool one)
```

Sendet den Befehl zum Ausschalten einer LED.

*(ii) Modul Command_Interface*

```
void CANReceiveHandler(unsigned char messageId, unsigned
char message[BUF_LEN])
```

Statemachine zum Interpretieren einer empfangenen Nachricht.

### 5.3.3 Timing

Die folgenden Tabellen geben die vom Mikrokontroller benötigten Zeiten zum Ausführen der Befehle an. Die Übertragung der Befehle dauert 500 µs. Die Übertragung eines CAN-Paketes dauert 100 µs. Die Verzögerungszeit beim Versenden einer 24 Byte langen Nachricht über CAN errechnet sich dann folgendermaßen:

$T_d = (500 + 90 + 100 + 300 + 100 + 300 + 100)$ µs = 1490 µs.

Dies ist jetzt die vergangene Zeit zwischen dem Senden des ersten Bytes über die UART des Senders und dem Empfang des letzten Bytes beim CAN-Empfänger. Je nach Rechenleistung der beteiligten Prozessoren kommen noch Bearbeitungszeiten wie Kopieren der Nachricht in den UART-Buffer oder Zusammensetzten der 3 Pakete beim Empfänger in der Größenordnung von einigen Mikrosekunden dazu. Die Zeit zwischen Aufruf der Senderoutine und Bearbeitung der Nachricht wird also knapp über 1,5 ms dauern.

| Befehl | Ausführungszeit |
|---|---|
| LED einschalten | 14 µs |
| LED ausschalten | 14 µs |
| bis zu 8 Byte weiterleiten | 90 µs |
| 8 bis 16 Byte weiterleiten (Wartezeit zw. Paketen) | 300 µs |
| 16 bis 24 Byte weiterleiten (Wartezeit zw. Paketen) | 2 x 300 µs |
| 8 Byte von CAN auf UART weiterleiten | 100 µs |

**Tabelle 5.1** Ausführungszeit verschiedener Befehle (ab Empfang des letzten Bytes)

Die benötigten Zeiten zum Weiterleiten von Nachrichten beziehen sich auf den ersten Kanal, der reserviert wurde. Sind mehrere Kanäle reserviert, so entstehen Verzögerungszeiten durch Finden des richtigen Kanals zur angegebenen ID (die aber bei weitem nicht so gravierend sind wie beim Reservieren eines zusätzlichen Kanals).

| bereits reservierte Kanäle | benötigte Zeit zum Reservieren eines weiteren Kanals |
|---|---|
| 0 | 740 µs |
| 1 | 740 µs |
| 2 | 740 µs |
| 3 | 760 µs |
| 4 | 780 µs |
| 5 | 820 µs |
| 6 | 840 µs |
| 7 | 860 µs |
| 8 | 880 µs |
| 9 | 900 µs |
| 10 | 940 µs |
| 11 | 980 µs |
| 12 | 1020 µs |
| 13 | 1060 µs |
| 14 | 1100 µs |

**Tabelle 5.2** benötigte Zeit zum Reservieren eines weiteren CAN-Kanals

## 5.4 Interface Funkmodul

Das Funkmodul der Firma Nordic kann in drei verschiedenen Modi betrieben werden: Configuration Mode, Shock Burst Mode und Direct Mode.

### 5.4.1 Protokoll

Im Configuration Mode kann ein 15 Byte langes Configuration Word geschrieben werden. Es ist jedoch nicht erforderlich, immer alle 15 Byte zu ändern, man kann vom LSB in Byteschritten aufwärts einzelne Bytes ändern. D.h. will man das vierte Byte ändern, müssen nur die unteren vier Byte neu geschrieben werden. Das Configuration Word ist so aufgebaut, dass im Normalbetrieb nur die unteren zwei Byte geändert werden müssen, die folgende Funktionen enthalten: Sendeleistung, Sendefreuquenz, Quartzfrequenz, Umschalten zwischen Senden und Empfangen und Ein- oder Zweikanal-Empfang. Die restlichen Bytes enthalten Empfängeradresse für Kanal 1 und 2

sowie Daten- und Adressbreite für beide Kanäle und zwei Bit zum Konfigurieren der Checksumme.

Im Direct und im Shock Burst Mode lässt sich die Übertragungs-Datenrate auf 250 kbaud oder 1 Mbaud festlegen.

Beim Senden im Direct Mode muss die Clockleitung auf Masse bleiben, die Datenleitung muss die zu übertragenden Daten mit der Sendefrequenz bereitstellen, wobei die ersten acht Bit eine Preamble mit lauter Null-Eins Wechsel zum Synchronisieren sein müssen. Der Empfänger wartet auf den Empfang einer solchen Preamble und stellt dann Clock- und Datenleitung für den Prozessor bereit. Ein Datenwort im Direct Mode ist genau 4000 Bit lang.

Im Shock Burst Mode können bis zu 32 Byte Daten an das Funkmodul übertragen werden, dieses generiert dann Checksumme und Preamble und überträgt die Daten mit der eingestellten Sendefrequenz. Der Vorteil beim Shock Burst Mode ist, dass die bereitgestellten Daten mit beliebiger Datenrate an das Funkmodul übertragen werden können, der Nachteil liegt in den kurzen Datenwörtern und dem damit verbundenen Zeitverlust zum hochfahren des Sendeteils.

Die Nettodatenrate beim Shock Burst Mode liegt bei ca. 33%, da Daten einlesen, Settling Time und Daten versenden gleich lange dauern. Beim Direct Mode liegt die Nettodatenrate bei über 90%, allerdings ist der Direct Mode nicht SPI kompatibel und müsste somit mit GPIO's gelöst werden. Der Signalprozessor müsste einen Interrupt mit einer Periode von 1µs generieren, was bei 400 MHz Taktrate alleine schon zu einer Prozessorauslastung von fast 50% führt.

Da dieser Performanceverlust für die Abarbeitung eines komplexen Algorithmus nicht verkraftbar ist, kann das Funkmodul nur im Shock Burst Mode betrieben werden.

### 5.4.2 Programm

Nachfolgend werden die Funktionen des Interfaces beschrieben. Der Direct Mode wird von den Funktionen unterstützt, jedoch nicht verwendet.

```
void nrf_init(bool shock, bool rec)
```

Schreibt das Configuration Word, verwendet dabei den Core Timer des Blackfin. Initialisiert anschließend das SPI und die Control Register der Pins.

```
void nrf_send(bool shock, unsigned char *data)
```

Konfiguriert das Modul wenn notwendig zum Senden von Daten und übergibt diese dem SPI.

```
void nrf_receive(bool shock, unsigned char *data)
```

Konfiguriert das Modul wenn notwendig zum Empfangen von Daten. Setzt ein Flag, dass vom Hauptprogramm gepollt werden kann.

```
EX_INTERRUPT_HANDLER(CORE_TIMER_ISR)
```

ISR, die vom Core Timer aufgerufen wird. Beim Konfigurieren beträgt die Periode 1µs.

## 5.5 Interface Audio Codec

Wie schon mehrfach erwähnt, stammt das Interface zum Audio Codec von Analog Devices. Der Audio Codec liefert die Daten im AC´97 Format, welches Datensamples mit variabler Abtastfrequenz und Auflösung unterstützt. Der Begriff Studioqualität steht für Audiodaten mit 16 Bit Auflösung und 48 kHz Abtastrate, was eine Datenrate von 1,5 MBit ergibt.

Nachfolgend die Beschreibung der einzelnen Funktionen.

Für das Programm werden zwei Header-Files verwendet, CdefBF533.h und Talkthrough BF533.h. Erstere definiert Zeiger auf alle SFR's von SPORT0 während in der Zweiten globale Variablen und Prototypen der Funktionen sowie eine Liste aller Konfigurationsregister des Audio Codec enthalten sind.

*(i) Modul Initialization*

```
void Init_SPORT0(void)
```

Initialisiert SPORT0.

```
void Init_Interrupts(void)
```

Initialisiert die Interruptvektoren für RX und TX-Interrupt des SPORT0.

```
void Init_Codec(void)
```

Sendet die Konfiguration an den Audio Codec.

*(ii) Modul Process Audio Data*

```
void Process_Audio_Data(void)
```

Überprüft, ob die empfangenen Daten im korrekten Format sind und übergibt sie an eine Callback-Funktion.

```
void Audio_Data_Output(void)
```

Schreibt Ausgabedaten in das TX-SFR von SPORT0.

```
void Wait_For_Codec_Init(void)
```

Diese Funktion schreibt immer ein neues Codec Register in das TX-SFR von SPORT0.

```
void SPORT0_RX_Dummy_Handler()
```

Wird nur während der Initialisierung verwendet, anschließend wird sie durch `void Process_Audio_Data(void)` ersetzt.

```
void SPORT0_TX_Dummy_Handler()
```

Wird nur während der Initialisierung verwendet, anschließend wird sie durch `void Audio_Data_Output(void)` ersetzt.

*(iii) Modul ISRs*

`EX_INTERRUPT_HANDLER(SPORT0_TX_ISR)` und

`EX_INTERRUPT_HANDLER(SPORT0_RX_ISR)`

ISR's, die die verwendeten Flags rücksetzen.

# Kapitel 6

# Ergebnisse und mögliche Erweiterungen

Nach Analyse, Entwicklung und Implementierung ist es nun an der Zeit, die Ergebnisse der abgeschlossenen Arbeit zu präsentieren und diese mit der Spezifikation zu vergleichen.

Die anfangs vorgestellte Vision einer möglichst kleinen, stromsparenden, aber leistungsfähigen Plattform zur Bildbearbeitung wurde Wirklichkeit. Die realisierten Prototypen finden beim Projekt Tinyphoon Verwendung und erweitern die Sensordaten des Roboters um Informationen für eine autonome Ballerkennung. Weiters wurde die Möglichkeit geschaffen, Kamerabilder live auf einen PC zu übertragen, sowie Audiodaten von Mikrophonen einzulesen und an Lautsprecher auszugeben.

Trotzdem sind im Zuge der Entwicklung einige Probleme und Schwachstellen der gefundenen Lösung aufgetaucht, es wurden auch durch die in den letzten Monaten auf den Markt gekommenen Produkte Erweiterungsmöglichkeiten gefunden, die zur Zeit des Systementwurfs noch nicht möglich waren.

Trotz allem, die geforderte Funktionalität des Systems ist gegeben, jedoch gibt es einige Verbesserungsmöglichkeiten, die ich hier erwähnen möchte.

- → Die Verwendung eines Blackfin aus einer aktuellen Großserie würde die Performance des Prozessors um 50% und des Speichers um 65% verbessern.
- → Ein SPI-kompatibles Funkmodul wurde gesucht, gefunden und integriert, jedoch hat sich herausgestellt, dass der günstigste Transmission Mode, der Direct Mode, nicht vollständig SPI-kompatibel ist. In der Zwischenzeit sind Alternativen am Markt, die dieses Phänomen nicht mehr haben. Die Verwendung eines solchen Funkmoduls in Zukunft wird dringend empfohlen.
- → Um die Möglichkeiten einer Bildverarbeitung voll auszuschöpfen und auch Entfernungen von Objekten mit unbekannten Abmessungen angeben zu können ist es erforderlich, zwei versetzte Kamerabilder, also Stereo-Sehen zu verwenden. Das vorgestellte System ist in der Lage, von zwei Kameras Bilder einzulesen. Der Nachteil ist allerdings, dass diese Bilder hintereinander aufgenommen werden und somit nicht mehr dieselbe Umgebung darstellen.

Wenn also Stereo-Sehen für eine Anwendung erforderlich ist, sollte eine Erweiterung des Systems um einen Dual-Core-Prozessor überlegt werden.

→ Die neuen Standards JPEG2000 und Motion JPEG2000 verwenden zur Transformationskodierung die DWT. JPEG2000 kodierte Bilder benötigen bei gleicher Bildqualität im Vergleich zu JPEG nur ca. 50% Speichervolumen. Die Verwendung der DWT bei der Videokompression würde eine größere Bildrate oder bessere Bildqualität bei gleicher Bildrate ermöglichen.

→ Der Mikrokontroller ist aufgrund seiner externen Beschaltung immer im Bootloader Mode und muss vom Signalprozessor durch Senden eines Initialisierungsstrings in den Active Mode gebracht werden. Da der Bootloader Mode zum Neuprogrammieren ebenfalls notwendig ist, empfehle ich den zusätzlichen Einsatz eines Dip-Switches zum Wechseln der Modi.

→ Zusätzliche Sensoren wie Ultraschall- oder Infrarot-Abstandssensoren, die über die A/D-Wandlereingänge des Mikrokontrollers eingelesen werden können, würden das Bild der Umgebung weiter vervollständigen.

→ Bei Verwendung von BGA-Gehäusen bei RAM und ROM würde auf der Platine Platz für weitere Sensoren geschaffen.

→ Die Videokompressions-Standards MPEG und p * 64 definieren Schichten für die Übertragung von Audio- und Videodaten sowie eine Synchronisationsschicht. Weiters gibt es Methoden, Audiodaten effizient zu komprimieren. Die Implementation solcher Algorithmen würde eine gleichzeitige Übertragung von Audio- und Videodaten ermöglichen. Bei diesem Projekt wurde jedoch davon Abstand genommen, die eingelesenen Audiodaten weiter zu bearbeiten, da diese beim Roboterfussball keine nützlichen Sensordaten darstellen.

→ Sowohl der Mikrokontroller als auch der Signalprozessor verfügen über eine UART, die mit mehr als 500 kbaud betrieben werden kann. Normale PC-Schnittstellen lassen sich jedoch nur mit maximal 115,2 kbaud öffnen. Um eine schnelle Datenübertragung vom PC an einen der beiden Prozessoren zu ermöglichen, muss man den Umweg über USB und eine zusätzlich Platine nehmen. Dieses Problem lässt sich durch Mitintegration eines USB-Treiber-Chips lösen.

→ Bei einer Verwendung als batteriebetriebene Überwachungseinheit bietet sich die Integration von Bewegungssensoren an. Der Stromverbrauch könnte drastisch reduziert werden.

→ Die Nutzung modernster Herstellungsverfahren wie Mikro-Vias, mehr als 4 Layer und Leiterbahnbreiten von ≤ 100 µm würde die Größe der Platine drastisch reduzieren und so Platz für Zusatzsensoren oder andere Peripherie schaffen.

Die Liste der Erweiterungsmöglichkeiten ist zwar lang und könnte noch verlängert werden, sie spiegelt jedoch auch den Umfang dieser Arbeit wieder. Das Ziel dieser Diplomarbeit war die Realisierung von Prototypen, die in der Lage sind, Bild- und Videodaten komprimiert per Bus- oder Funksystem an einen PC zu übertragen. Dieses Ziel wurde erreicht. Die Implementierung sämtlicher möglicher Erweiterungen würde den Rahmen einer Diplomarbeit bei weitem sprengen.

Schon alleine weil jedes Quartal neue, leistungsfähigere Hardwarekomponenten am Markt erscheinen, kann die Weiterentwicklung an einem solchen Projekt nicht gestoppt werden, es muss aber auch zu bestimmten Zeiten ein als funktionierend erachtetes System zur Verfügung stehen.

Zusammengefasst stellt das realisierte System eine flexible und gut erweiterbare Lösung dar. Sowohl die Hardware als auch die Software erfüllen die Spezifikation. Aufgrund des geringen Stromverbrauchs wird ein Batteriebetrieb ermöglicht, das Funkmodul und die geringe Baugröße erlauben eine Installation praktisch überall als Überwachungseinheit und auf jedem Industrieroboter als intelligenter Sensor. Die Anzahl der möglichen Applikationen erscheint bereits aus heutiger Sicht sehr groß zu sein.

Aufgrund der praktischen Speicherung und Weiterverarbeitung für den Privatanwender und den enormen Möglichkeiten der Sensordaten für die Industrie werden optische Sensoren auch weiterhin Einzug in unser aller Leben halten. Diese Arbeit soll nur einen kleinen Schritt in die richtige Richtung darstellen und Möglichkeiten für die Zukunft aufzeigen.

# Literatur


[BER01] Berger, A. S.; „*Embedded Systems Design*“; CMP Books, 2001

[BRU03] Bruckner, D.; „*Dokumentation ubBlackfin*“; Dokumentation zum Projekt Tinyphoon; TU Wien; Institut für Computertechnik, 2003

[DIE94] Dietrich, D.; "*Bussysteme und Rechnerkommunikation*"; Vorlesungsskriptum; TU Wien; Institut für Computertechnik, 1994

[GEI96] Geier, J.; „*Wireless Networking Handbook*“; Macmillan Computer Pub, 1996

[IEE91] IEEE Standard *„Specifications fort he Implementations of 8x8 Inverse Discrete Cosine Transform“*; IEEE Std 1180-1990, 1991

[ITU93] Recommendation H.261; „*Video Codec for Audiovisual Services at px64 kBits*“; ITU-T (CCITT), 1993

[ITU95] Draft Recommendation H.263; „*Video Coding for Low Bitrate Communication*“; ITU-T (CCITT), 1995

[ITU98] Recommendation ITU-R BT.656-4*; „Interfaces for digital component video signals in 525-line and 625-line television systems operating at the 4:2:2 level of recommendation“;* ITU-R BT.601 (part A); ITU, 1998

[KUH98] Kuhn P.; “*Algorithms, Complexity Analysis and Vlsi Architectures for Mpeg-4 Motion Estimation”*, Dissertation, 1998

[LAB99] Labrosse, J. J.; „*Embedded Systems Building Blocks*“; CMP Books, 1999

[LIN00] Lindner, M.; "*Datenkommunikation und Netzwerke*"; Vorlesungsskriptum; TU Wien; Institut für Computertechnik, 2000

[MPEG1] ISO/IEC 11172-2; "*Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1,5 Mbit/s - Video*"; ISO, 1993

[MPEG2] ISO/IEC JTC1/SC29/WG11 N0702 Rev; "*Information Technology - Generic Coding of Moving Pictures and Associated Audio, Recommendation H.262*"; Draft International Standard, ISO, 1994.

[MPF96] Mitchell, Pennebaker, Fogg and LeGall; „*MPEG Video Compression Standard*“; Chapman & Hall, 1996

[MPG85] Musmann, Pirsch, Grallert; „*Advances in Picture Coding*“; Proc. IEEE, 73(4):523-48, 1985

[PRO97-1] Hägermann, Schneider; „*Propyläen Technik Geschichte, 750 v. Chr – 1000 n. Chr.“;* Ullstein Verlag, 1997

[PRO97-2] Ludwig, Schmidtchen; „*Propyläen Technik Geschichte, 1000 – 1600“;* Ullstein Verlag, 1997

[PRO97-3] Paulinyi, Troitzsch; „*Propyläen Technik Geschichte, 1600 – 1840“;* Ullstein Verlag, 1997

[PRO97-4] König, Weber; „*Propyläen Technik Geschichte, 1840 – 1914“;* Ullstein Verlag, 1997

[PRO97-5] Braun, Kaiser; „*Propyläen Technik Geschichte, 1914 – jetzt“;* Ullstein Verlag, 1997

[WEI98] Weinrichter, H.; „*Einführung in die Nachrichtentechnik*“; Vorlesungsskriptum; TU Wien, Institut für Nachrichtentechnik und Hochfrequenztechnik, 1998

# Internetreferenzen

| | | |
|---|---|---|
| [W3AD] | Analog Devices Inc. | www.analog.com |
| [W3ATM] | Atmel Corporation | www.atmel.com |
| [W3BMP] | Erklärung des BMP-Headers | http://www.isa-verein.de/VBO/Tut/Tut_BMP_Info.htm |
| [W3CAN] | CAN in Automation | www.can-cia.org/can |
| [W3CEN] | Centurion | |
| | Wireless Technologies, Inc. | www.centurion.com |
| [W3FS] | Fairchild | |
| | Semiconductor Corporation | www.fairchildsemi.com |
| [W3261] | H.261 Einführung | www.lpr.e-technik.tu-muenchen.de/courses/seminar/realzeit_bv/mpeg/node4.html |
| [W3263] | H.263 Einführung | www.lpr.e-technik.tu-muechen.de/courses/seminar/realzeit_bv/mpeg/node4.html#SECTION00043000000000000000 |
| [W3INF] | Infenion Technologies AG | www.infenion.com |
| [W3IDT] | Integrated Devices | |
| | Technologies, Inc. | www.idt.com |
| [W3INT] | Intel Corporation | www.intel.com |
| [W3ISO] | International | |
| | Organization for Standarization | www.iso.org |
| [W3JAU] | Jauch Quartz GmbH | www.jauch.de |
| [W3JP] | JPEG-Homepage | www.jpeg.org |
| [W3JP2] | JPEG2000-Homepage | www.jpeg.org/JPEG2000.htm |
| [W3KEI] | Keil<br>Software Development Tools | www.keil.com |

| | | |
|---|---|---|
| [W3MAX] | Maxim Integrated Products | www.maxim-ic.com |
| [W3MIC] | Micron Technologies, Inc. | www.micron.com |
| [W3MX] | Molex Inc. | www.molex.com |
| [W3MOT] | Motorola, Inc. | www.motorola.com |
| [W3MDE] | MPEG-2 Archiv | www.mpeg2.de |
| [W3MP] | MPEG-Homepage | www.mpeg.org |
| [W3NOR] | Nordic VLSI ASA | www.nvlsi.no |
| [W3OV] | OmniVision Technologies, Inc. | www..com |
| [W3RFW] | RFWaves Ltd. | www.rfwaves.com |
| [W3TI] | Texas Instruments Incorporated: | www.ti.com |
| [W3COS] | Homepage von Takuya Ooura | |
| [W3TUC] | TU-Chemnitz | www.tu-chemnitz.de |
| [W3WÜR] | Würth Elektronik eiSos | |
| | GmbH & Co. KG | www.we-online.com |
| [W3XVD] | Home of the XviD Codec | www.xvid.org |

# Gleichungsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Anhang A: MPEG Kodierungstabellen

| motion vector magnitude | motion_code VLC |
|---|---|
| 0 | 1 |
| 1 | 01s |
| 2 | 001s |
| 3 | 0000 11s |
| 4 | 0000 11s |
| 5 | 0000 101s |
| 6 | 0000 100s |
| 7 | 0000 011s |
| 8 | 0000 0101 1s |
| 9 | 0000 0101 0s |
| 10 | 0000 0100 1s |
| 11 | 0000 0100 01s |
| 12 | 0000 0100 00s |
| 13 | 0000 0011 11s |
| 14 | 0000 0011 10s |
| 15 | 0000 0011 01s |
| 16 | 0000 0011 00s |

**Tabelle 6.1** VLC der Motion Vectoren

| Y Code | C Code | Größe | Amplitude |
|---|---|---|---|
| 100 | 00 | 0 | 0 |
| 00 | 01 | 1 | -1\|1 |
| 01 | 10 | 2 | -3,-2\|2,3 |
| 101 | 110 | 3 | -7..-4\|4..7 |
| 110 | 1110 | 4 | -15..-8\|8..15 |
| 1110 | 1111 0 | 5 | -31..-16\|16..31 |
| 1111 0 | 1111 10 | 6 | -63..-32\|32..63 |
| 1111 10 | 1111 110 | 7 | -127..-64\|64..127 |
| 1111 110 | 1111 1110 | 8 | -255..-128\|128..255 |
| 1111 1110 | 1111 1111 0 | 9 | -511..-256\|256..511 |
| 1111 1111 0 | 1111 1111 10 | 10 | -1023..-512\|512..1023 |
| 1111 1111 1 | 1111 1111 11 | 11 | -2047..-1024\|1024..2047 |

**Tabelle 6.2** VLC des Symbol 1 nach der Lauflängenkodierung

| run/level | VLC | bits |
|---|---|---|
| 0/1 | 1s (first) | 2 |
| 0/1 | 11s (next) | 3 |
| 0/2 | 0100 s | 5 |
| 0/3 | 0010 1s | 6 |
| 0/4 | 0000 110s | 8 |
| 0/5 | 0010 0110 s | 9 |
| 0/6 | 0100 0001 s | 9 |
| 0/7 | 0000 0010 10s | 11 |
| 0/8 | 0000 0001 1101 s | 13 |
| 0/9 | 0000 0001 1000 s | 13 |
| 0/10 | 0000 0001 0011 s | 13 |
| 0/11 | 0000 0001 0000s | 13 |
| 0/12 | 0000 0000 1101 0s | 14 |
| 0/13 | 0000 0000 1100 1s | 14 |
| 0/14 | 0000 0000 1100 0s | 14 |
| 0/15 | 0000 0000 1011 1s | 14 |
| 0/16 | 0000 0000 0111 11s | 15 |
| 0/17 | 0000 0000 0111 10s | 15 |
| 0/18 | 0000 0000 0111 01s | 15 |
| 0/19 | 0000 0000 0111 00s | 15 |
| 0/20 | 0000 0000 0110 11s | 15 |
| 0/21 | 0000 0000 0110 10s | 15 |
| 0/22 | 0000 0000 0110 01s | 15 |
| 0/23 | 0000 0000 0110 00s | 15 |
| 0/24 | 0000 0000 0101 11s | 15 |
| 0/25 | 0000 0000 0101 10s | 15 |
| 0/26 | 0000 0000 0101 01s | 15 |
| 0/27 | 0000 0000 0101 00s | 15 |
| 0/28 | 0000 0000 0100 11s | 15 |
| 0/29 | 0000 0000 0100 10s | 15 |
| 0/30 | 0000 0000 0100 01s | 15 |
| 0/31 | 0000 0000 0100 00s | 15 |
| 0/32 | 0000 0000 0011 000s | 16 |
| 0/33 | 0000 0000 0010 111s | 16 |
| 0/34 | 0000 0000 0010 110s | 16 |
| 0/35 | 0000 0000 0010 101s | 16 |
| 0/36 | 0000 0000 0010 100s | 16 |
| 0/37 | 0000 0000 0010 011s | 16 |
| 0/38 | 0000 0000 0010 010s | 16 |
| 0/39 | 0000 0000 0010 001s | 16 |
| 0/40 | 0000 0000 0010 000s | 16 |
| 1/1 | 011s | 4 |
| 1/2 | 0001 10s | 7 |
| 1/3 | 0010 0101 s | 9 |
| 1/4 | 0000 0011 00s | 11 |
| 1/5 | 0000 0001 1011 s | 13 |
| 1/6 | 0000 0000 1011 0s | 14 |
| 1/7 | 0000 0000 1010 1s | 14 |
| 1/8 | 0000 0000 0011 111s | 16 |
| 1/9 | 0000 0000 0011 110s | 16 |
| 1/10 | 0000 0000 0011 101s | 16 |
| 1/11 | 0000 0000 0011 100s | 16 |
| 1/12 | 0000 0000 0011 011s | 16 |
| 1/13 | 0000 0000 0011 010s | 16 |
| 1/14 | 0000 0000 0011 001s | 16 |
| 1/15 | 0000 0000 0001 0011 s | 17 |
| 1/16 | 0000 0000 0001 0010 s | 17 |

| run/level | VLC | bits |
|---|---|---|
| 1/17 | 0000 0000 0001 0001 s | 17 |
| 1/18 | 0000 0000 0001 0000 s | 17 |
| 2/1 | 0101 s | 5 |
| 2/2 | 0000 100s | 8 |
| 2/3 | 0000 0010 11s | 11 |
| 2/4 | 0000 0001 0100 s | 13 |
| 2/5 | 0000 0000 1010 0s | 14 |
| 3/1 | 0011 1s | 6 |
| 3/2 | 0010 0100 s | 9 |
| 3/3 | 0000 0001 1100 s | 13 |
| 3/4 | 0000 0000 1001 1s | 14 |
| 4/1 | 0011 0s | 6 |
| 4/2 | 0000 0011 11s | 11 |
| 4/3 | 0000 0001 0010 s | 13 |
| 5/1 | 0001 11s | 7 |
| 5/2 | 0000 0010 01s | 11 |
| 5/3 | 0000 0000 1001 0s | 14 |
| 6/1 | 0001 01s | 7 |
| 6/2 | 0000 0001 1110 s | 13 |
| 6/3 | 0000 0000 0001 0100 s | 17 |
| 7/1 | 0001 00s | 7 |
| 7/2 | 0000 0001 0101 s | 13 |
| 8/1 | 0000 111s | 8 |
| 8/2 | 0000 0001 0001 s | 13 |
| 9/1 | 0000 101s | 8 |
| 9/2 | 0000 0000 1000 1s | 14 |
| 10/1 | 0010 0111 s | 8 |
| 10/2 | 0000 0000 1000 0s | 17 |
| 11/1 | 0010 0011 s | 9 |
| 11/2 | 0000 0000 0001 1010 s | 17 |
| 12/1 | 0010 0010 s | 9 |
| 12/2 | 0000 0000 0001 1001 s | 17 |
| 13/1 | 0010 0000 s | 9 |
| 13/2 | 0000 0000 0001 1000 s | 17 |
| 14/1 | 0000 0011 10s | 11 |
| 14/2 | 0000 0000 0001 0111 s | 17 |
| 15/1 | 0000 0011 01s | 11 |
| 15/2 | 0000 0000 0001 0110 s | 17 |
| 16/1 | 0000 0010 00s | 11 |
| 16/2 | 0000 0000 0001 0101 s | 17 |
| 17/1 | 0000 0001 1111 s | 13 |
| 18/1 | 0000 0001 1010 s | 13 |
| 19/1 | 0000 0001 1001 s | 13 |
| 20/1 | 0000 0001 0111 s | 13 |
| 21/1 | 0000 0001 0110 s | 13 |
| 22/1 | 0000 0000 1111 1s | 14 |
| 23/1 | 0000 0000 1111 0s | 14 |
| 24/1 | 0000 0000 1110 1s | 14 |
| 25/1 | 0000 0000 1110 0s | 14 |
| 26/1 | 0000 0000 1101 1s | 14 |
| 27/1 | 0000 0000 0001 1111 s | 17 |
| 28/1 | 0000 0000 0001 1110 s | 17 |
| 29/1 | 0000 0000 0001 1101 s | 17 |
| 30/1 | 0000 0000 0001 1100 s | 17 |
| 31/1 | 0000 0000 0001 1011 s | 17 |
| EoB | 10 | 2 |
| Escape | 0000 01 | 6 |

**Tabelle 6.3** VLC des Symbol 2 nach der Lauflängenkodierung

| increment value | macroblock_ address_increment |
|---|---|
| macroblock_escape | 0000 0001 000 |
| macroblock_stuffing | 0000 0001 111 |
| 33 | 0000 0011 000 |
| 32 | 0000 0011 001 |
| 31 | 0000 0011 010 |
| 30 | 0000 0011 011 |
| 29 | 0000 0011 100 |
| 28 | 0000 0011 101 |
| 27 | 0000 0011 110 |
| 26 | 0000 0011 111 |
| 25 | 0000 0100 000 |
| 24 | 0000 0100 001 |
| 23 | 0000 0100 010 |
| 22 | 0000 0100 011 |
| 21 | 0000 0100 10 |
| 20 | 0000 0100 11 |
| 19 | 0000 0101 00 |
| 18 | 0000 0101 01 |
| 17 | 0000 0101 10 |
| 16 | 0000 0101 11 |
| 15 | 0000 0110 |
| 14 | 0000 0111 |
| 13 | 0000 1000 |
| 12 | 0000 1001 |
| 11 | 0000 1010 |
| 10 | 0000 1011 |
| 9 | 0000 110 |
| 8 | 0000 111 |
| 7 | 0001 0 |
| 6 | 0001 1 |
| 5 | 0010 |
| 4 | 0011 |
| 3 | 010 |
| 2 | 011 |
| 1 | 1 |

**Tabelle 6.4** VLC des Makroblock Adressinkrement

| macroblock _intra | macroblock _pattern | macroblock _motion _backward | macroblock _motion _forward | macroblock _quant | macroblock _type VLC code |
|---|---|---|---|---|---|
| I-pictures | | | | | |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 01 |
| P-pictures | | | | | |
| 0 | 0 | 0 | 1 | 0 | 001 |
| 0 | 1 | 0 | 0 | 0 | 01 |
| 0 | 1 | 0 | 0 | 1 | 0000 1 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0001 0 |
| 1 | 0 | 0 | 0 | 0 | 0001 1 |
| 1 | 0 | 0 | 0 | 1 | 0000 01 |
| B-pictures | | | | | |
| 0 | 0 | 0 | 1 | 0 | 0010 |
| 0 | 0 | 1 | 0 | 0 | 010 |
| 0 | 0 | 1 | 1 | 0 | 10 |
| 0 | 1 | 0 | 1 | 0 | 0011 |
| 0 | 1 | 0 | 1 | 1 | 0000 11 |
| 0 | 1 | 1 | 0 | 0 | 011 |
| 0 | 1 | 1 | 0 | 1 | 0000 10 |
| 0 | 1 | 1 | 1 | 0 | 11 |
| 0 | 1 | 1 | 1 | 1 | 0001 0 |
| 1 | 0 | 0 | 0 | 0 | 0001 1 |
| 1 | 0 | 0 | 0 | 1 | 0000 01 |
| D-pictures | | | | | |
| 1 | 0 | 0 | 0 | 0 | 1 |

**Tabelle 6.5** VLC des Makroblock Typs

| coded_block _pattern binary | block number 0 Y | 1 Y | 2 Y | 3 Y | 4 $C_b$ | 5 $C_r$[1] | coded_block _pattern VLC code |
|---|---|---|---|---|---|---|---|
| 000000 | . | . | . | . | . | . | forbidden |
| 000001 | . | . | . | . | . | c | 0101 1 |
| 000010 | . | . | . | . | c | . | 0100 1 |
| 000011 | . | . | . | . | c | c | 0011 01 |
| 000100 | . | . | . | c | . | . | 1101 |
| 000101 | . | . | . | c | . | c | 0010 111 |
| 000110 | . | . | . | c | c | . | 0010 011 |
| 000111 | . | . | . | c | c | c | 0001 1111 |
| 001000 | . | . | c | . | . | . | 1100 |
| 001001 | . | . | c | . | . | c | 0010 110 |
| 001010 | . | . | c | . | c | . | 0010 010 |
| 001011 | . | . | c | . | c | c | 0001 1110 |
| 001100 | . | . | c | c | . | . | 1001 1 |
| 001101 | . | . | c | c | . | c | 0001 1011 |
| 001110 | . | . | c | c | c | . | 0001 0111 |
| 001111 | . | . | c | c | c | c | 0001 0011 |
| 010000 | . | c | . | . | . | . | 1011 |
| 010001 | . | c | . | . | . | c | 0010 101 |
| 010010 | . | c | . | . | c | . | 0010 001 |
| 010011 | . | c | . | . | c | c | 0001 1101 |
| 010100 | . | c | . | c | . | . | 1000 1 |
| 010101 | . | c | . | c | . | c | 0001 1001 |
| 010110 | . | c | . | c | c | . | 0001 0101 |
| 010111 | . | c | . | c | c | c | 0001 0001 |
| 011000 | . | c | c | . | . | . | 0011 11 |
| 011001 | . | c | c | . | . | c | 0000 1111 |
| 011010 | . | c | c | . | c | . | 0000 1101 |
| 011011 | . | c | c | . | c | c | 0000 0001 1 |
| 011100 | . | c | c | c | . | . | 0111 1 |
| 011101 | . | c | c | c | . | c | 0000 1011 |
| 011110 | . | c | c | c | c | . | 0000 0111 |
| 011111 | . | c | c | c | c | c | 0000 0011 1 |

**Tabelle 6.6** VLC des Coded Block Pattern

[1]Mit “c” gekennzeichnete Blöcke werden kodiert, mit “.” gekennzeichnete Blöcke übersprungen.

| coded_block _pattern binary | block number 0 Y | 1 Y | 2 Y | 3 Y | 4 $C_b$ | 5 $C_r$ [1] | coded_block _pattern VLC code |
|---|---|---|---|---|---|---|---|
| 100000 | c | . | . | . | . | . | 1010 |
| 100001 | c | . | . | . | . | c | 0010 100 |
| 100010 | c | . | . | . | c | . | 0010 000 |
| 100011 | c | . | . | . | c | c | 0001 1100 |
| 100100 | c | . | . | c | . | . | 0011 10 |
| 100101 | c | . | . | c | . | c | 0000 1110 |
| 100110 | c | . | . | c | c | . | 0000 1100 |
| 100111 | c | . | . | c | c | c | 0000 0001 0 |
| 101000 | c | . | c | . | . | . | 1000 0 |
| 101001 | c | . | c | . | . | c | 0001 1000 |
| 101010 | c | . | c | . | c | . | 0001 0100 |
| 101011 | c | . | c | . | c | c | 0001 0000 |
| 101100 | c | . | c | c | . | . | 0111 0 |
| 101101 | c | . | c | c | . | c | 0000 1010 |
| 101110 | c | . | c | c | c | . | 0000 0110 |
| 101111 | c | . | c | c | c | c | 0000 0011 0 |
| 110000 | c | c | . | . | . | . | 1001 0 |
| 110001 | c | c | . | . | . | c | 0001 1010 |
| 110010 | c | c | . | . | c | . | 0001 0110 |
| 110011 | c | c | . | . | c | c | 0001 0010 |
| 110100 | c | c | . | c | . | . | 0110 1 |
| 110101 | c | c | . | c | . | c | 0000 1001 |
| 110110 | c | c | . | c | c | . | 0000 0101 |
| 110111 | c | c | . | c | c | c | 0000 0010 1 |
| 111000 | c | c | c | . | . | . | 0110 0 |
| 111001 | c | c | c | . | . | c | 0000 1000 |
| 111010 | c | c | c | . | c | . | 0000 0100 |
| 111011 | c | c | c | . | c | c | 0000 0010 0 |
| 111100 | c | c | c | c | . | . | 111 |
| 111101 | c | c | c | c | . | c | 0101 0 |
| 111110 | c | c | c | c | c | . | 0100 0 |
| 111111 | c | c | c | c | c | c | 0011 00 |

**Tabelle 6.7** VLC des Coded Block Pattern (Fortsetzung)

[1] Mit "c" gekennzeichnete Blöcke werden kodiert, mit "." gekennzeichnete Blöcke übersprungen.

| quantiser_scale[0] decimal | quantiser_scale[1] decimal | quantiser_scale_code | |
|---|---|---|---|
| | | decimal | binary |
| forbidden | forbidden | 0 | 00000 |
| 2 | 1 | 1 | 00001 |
| 4 | 2 | 2 | 00010 |
| 6 | 3 | 3 | 00011 |
| 8 | 4 | 4 | 00100 |
| 10 | 5 | 5 | 00101 |
| 12 | 6 | 6 | 00110 |
| 14 | 7 | 7 | 00111 |
| 16 | 8 | 8 | 01000 |
| 18 | 10 | 9 | 01001 |
| 20 | 12 | 10 | 01010 |
| 22 | 14 | 11 | 01011 |
| 24 | 16 | 12 | 01100 |
| 26 | 18 | 13 | 01101 |
| 28 | 20 | 14 | 01110 |
| 30 | 22 | 15 | 01111 |
| 32 | 24 | 16 | 10000 |
| 34 | 28 | 17 | 10001 |
| 36 | 32 | 18 | 10010 |
| 38 | 36 | 19 | 10011 |
| 40 | 40 | 20 | 10100 |
| 42 | 44 | 21 | 10101 |
| 44 | 48 | 22 | 10110 |
| 46 | 52 | 23 | 10111 |
| 48 | 56 | 24 | 11000 |
| 50 | 64 | 25 | 11001 |
| 52 | 72 | 26 | 11010 |
| 54 | 80 | 27 | 11011 |
| 56 | 88 | 28 | 11100 |
| 58 | 96 | 29 | 11101 |
| 60 | 104 | 30 | 11110 |
| 62 | 112 | 31 | 11111 |

**Tabelle 6.8** Zuordnung quantiser scale code zum tatsächlich verwendeten Quantisierungsfaktor

# Anhang B: Schaltpläne

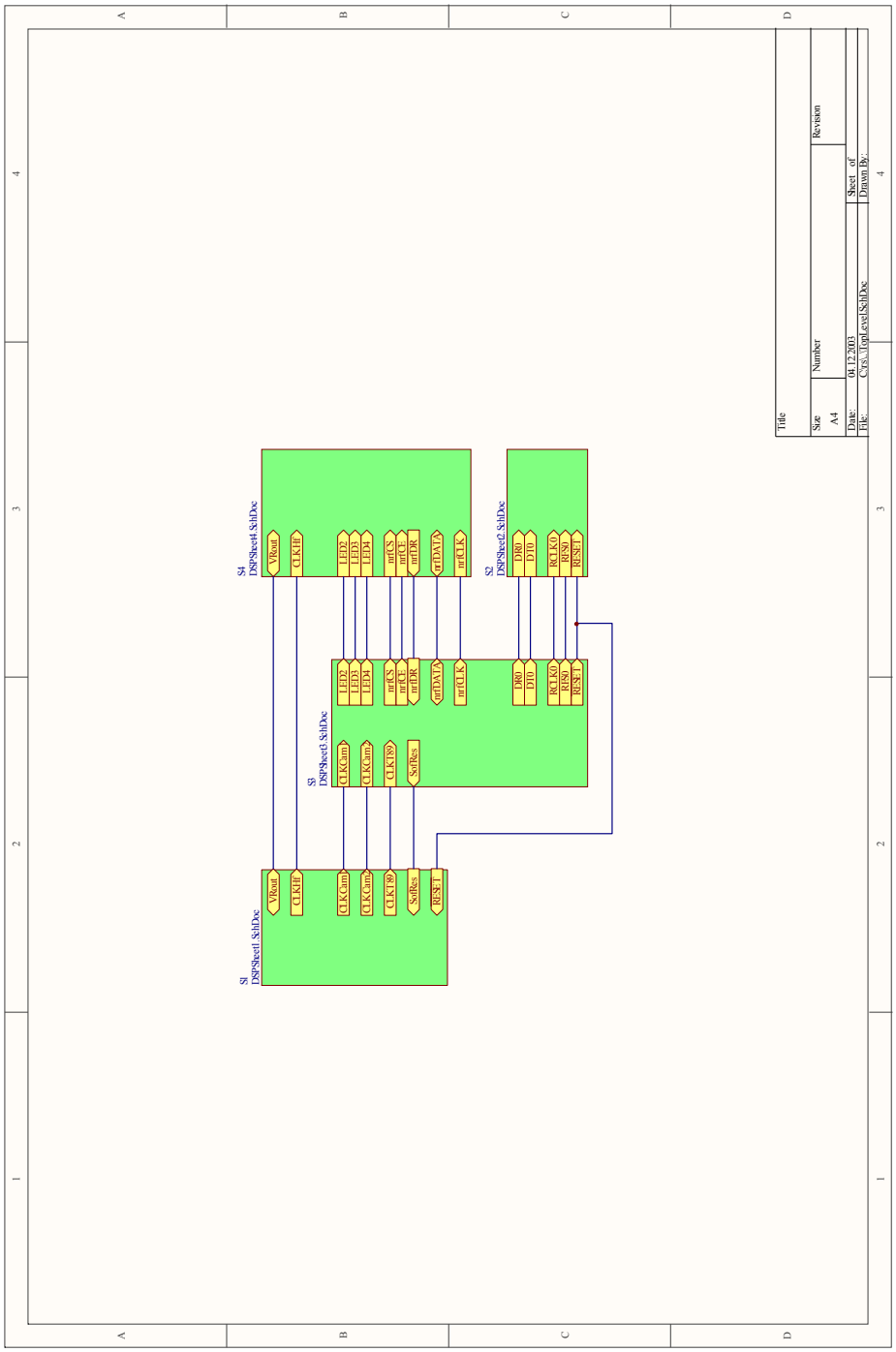


**Abbildung 6.1** Schaltplan: Toplevel Design, verbindet die einzelnen Blätter

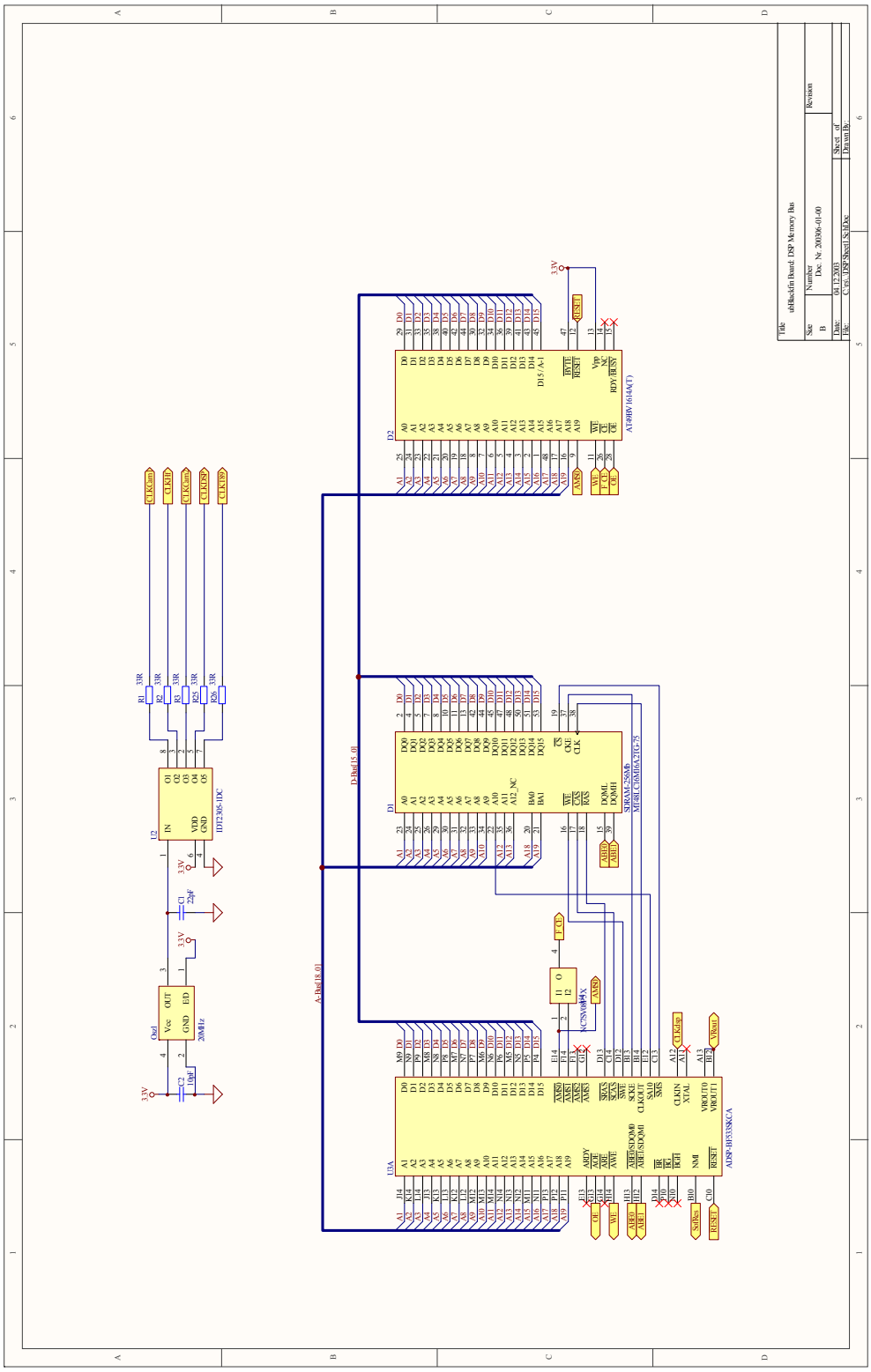

**Abbildung 6.2** Schaltplan: Signalprozessor (Memory-Controller) und Speicher

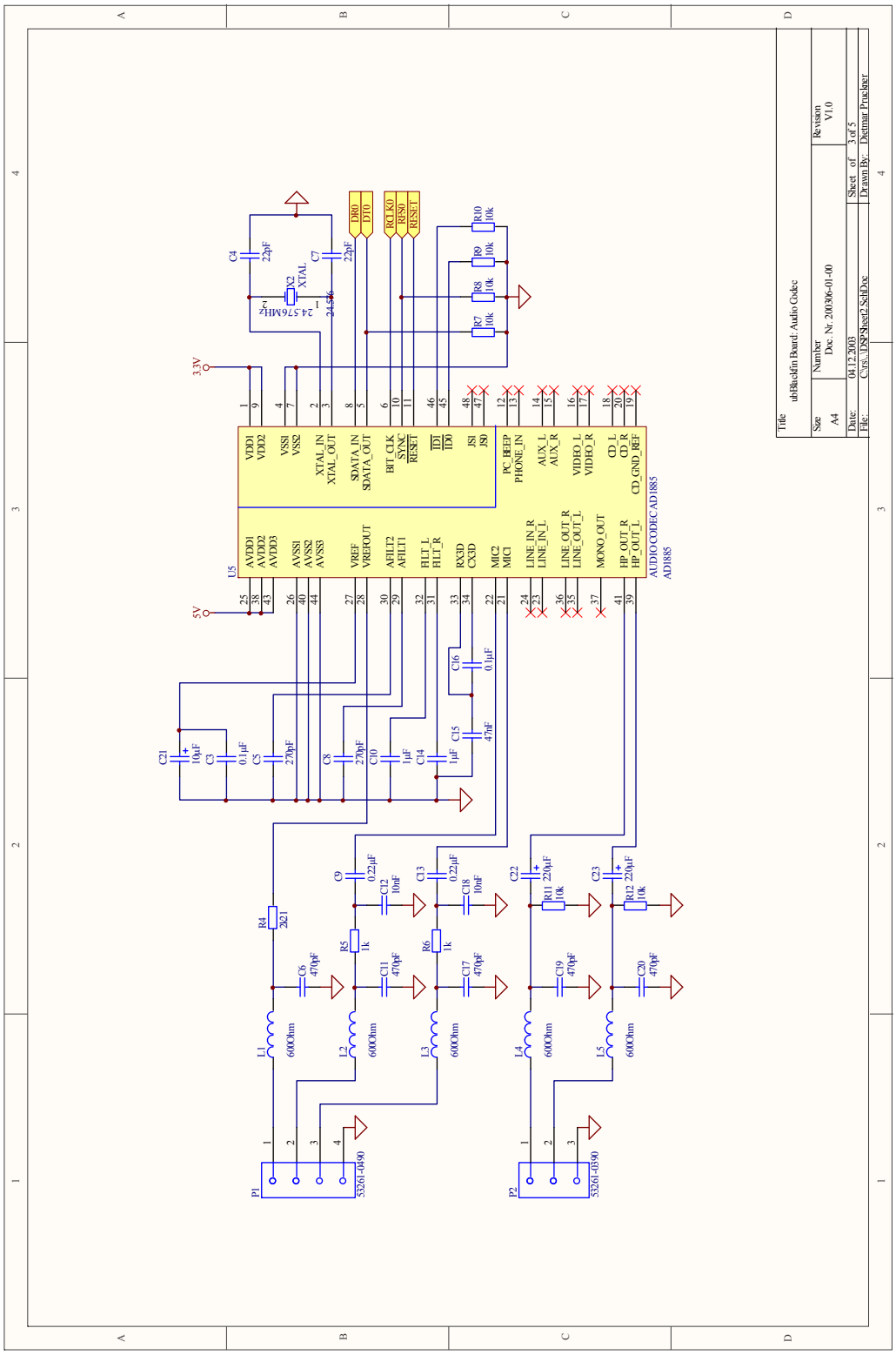


**Abbildung 6.3** Schaltplan: Audio Codec

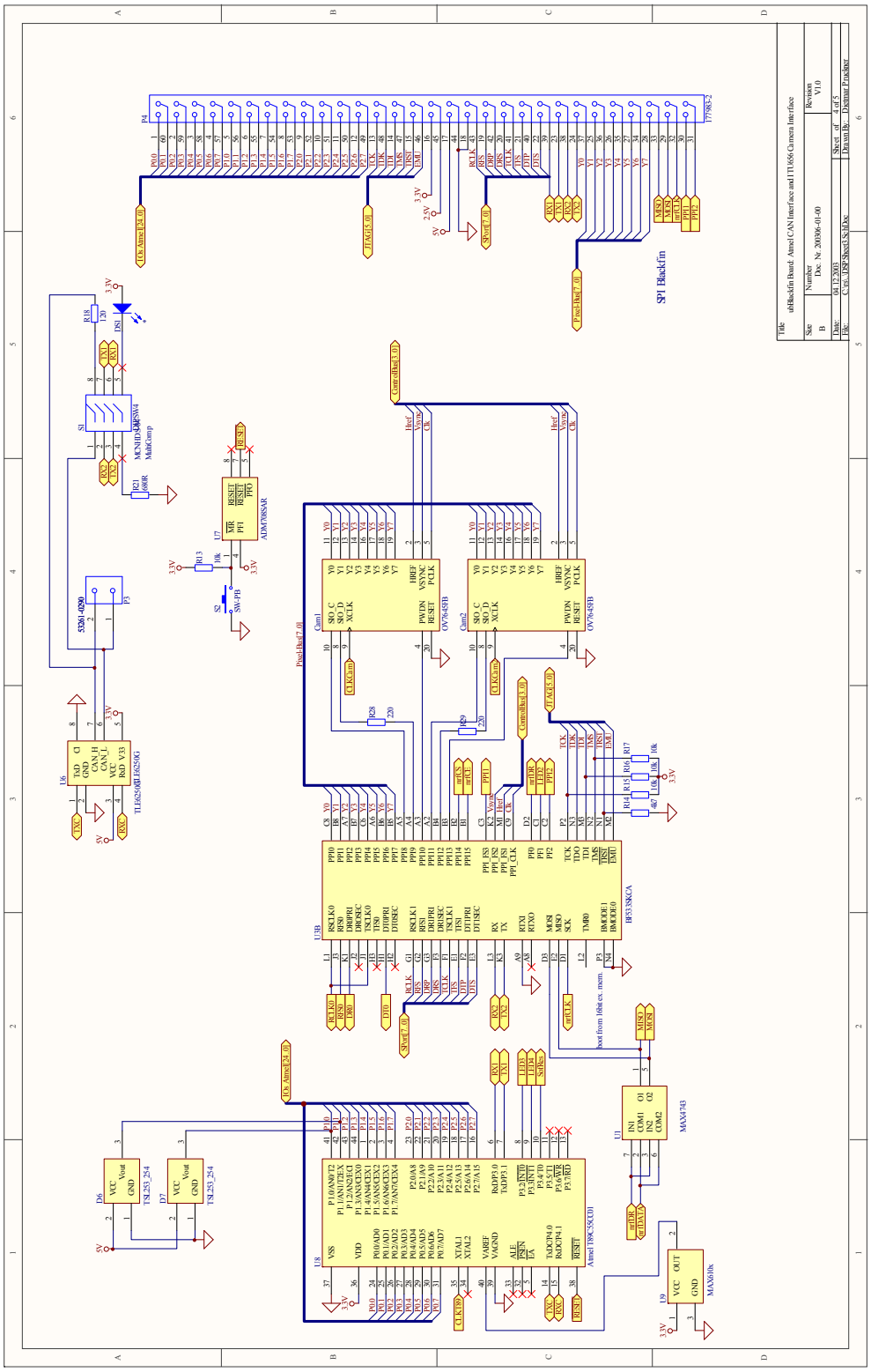

**Abbildung 6.4** Schaltplan: Mikrokontroller, Kameras, 60 pol. Stecker, Signalprozessor

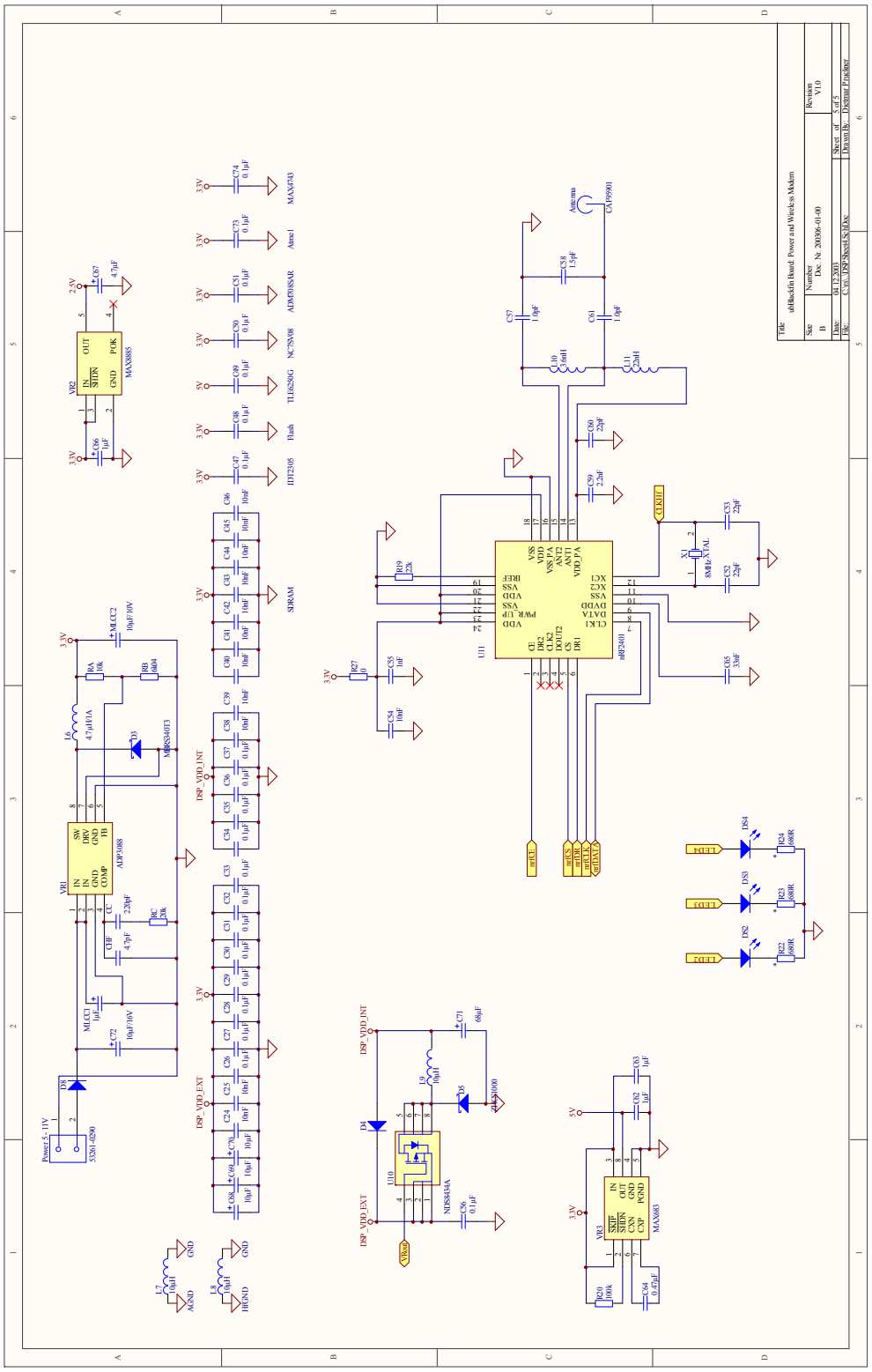

**Abbildung 6.5** Schaltplan: Versorgung, Funkmodul