Bakkalaureatsarbeit

Entwicklung eines FireWire Kamera Interface als eingebettetes System

Universität Hamburg, Fachbereich Informatik

Erstellt von: Sebastian Annies, Matr. 5121541 im Juli/August 2003

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

Im Arbeitsbereich ist ein Stereokamerakopf vorhanden, der mit zwei Sony DFW-VL-500 Kameras (Abbildung 1-1) bestückt ist. Zur Zeit werden alle Steuerungs- und Analyseaufgaben von einem PC-System bearbeitet. Es soll nun ein Gerät entwickelt werden, das den Hostrechner um einige dieser Aufgaben – im ersten Schritt um die Autofokussierung – entlastet, da die Kamera selbst keinen Autofokus implementiert.

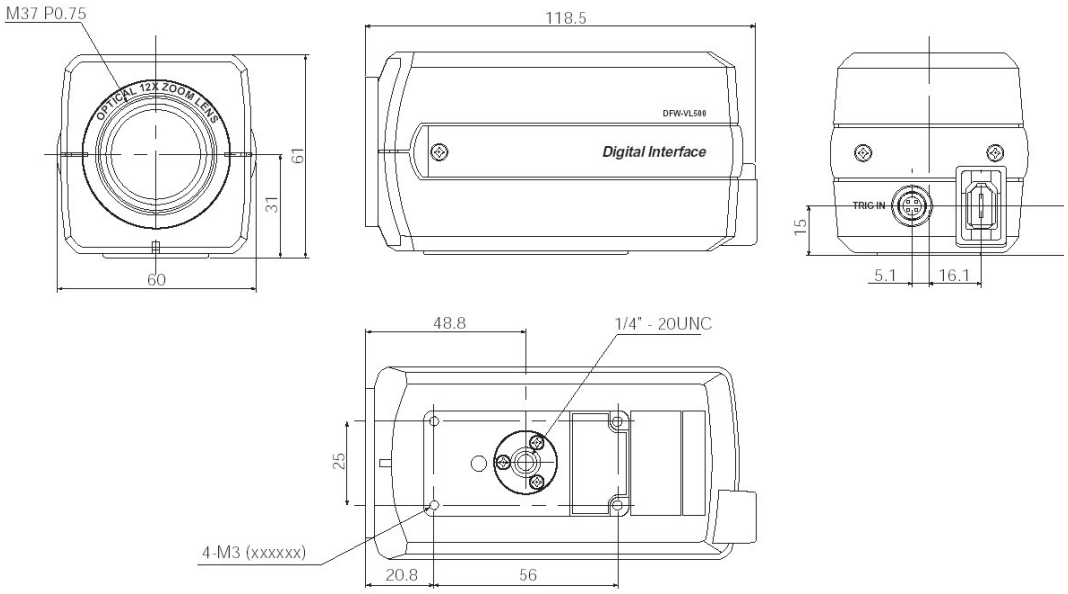

Abbildung 1-1: Die DFW-VL500

Kapitel 2 beschreibt den FireWire® und den Kamerastandard oberflächlich, der verwendete FireWire IC wird ausführlich beschrieben; Seine Konfigurationsmöglichkeiten und die Paketstrukturen werden umfassend dargestellt.

Kapitel 3 schließlich beschäftigt sich mit den Anforderungen an ein FireWire Gerät im Allgemeinen, dessen Verhalten bei der Initialisierung und dem eigentlichen Empfang der Bilddaten.

2 Grundlagen

2.1 IEEE1394a

IEEE1394a ist die aktuell (Sept. 2003) gebräuchliche FireWire Version. IEEE1394a ergänzt IEEE1394 und ist die erste wirklich umgesetzte FireWire Version. IEEE1394 ließ noch viele Detailfragen offen und war nicht hinreichend präzise, um herstellerübergreifende Interoperabilität zu gewährleisten.

Im Nachtrag IEEE1394-a-2000 wurde dann noch einmal die PHY-Link-Schnittstelle präzisiert und IEEE1394-a wurde zur besseren Unterscheidung seit dem häufig IEEE1394-a-1995 genannt.

Prinzipiell ist FireWire ein baumartiges Bussystem, d.h. eine jede Verbindung ist zulässig, solange dadurch kein Kreis entsteht.

Sobald ein Gerät mit dem Bus verbunden wird, wird jedem Gerät einen neue Node-Id zugewiesen und der sogenannte Buskonfigurationsprozess ausgelöst.

Die Übertragungsgeschwindigkeit beträgt je nach Gerät 100, 200 oder 400 Mbits/s. Die Übertragungsgeschwindigkeit zwischen zwei Geräten ist auf die des langsamsten Geräts auf dem Weg zwischen den beiden Kommunikationspartnern beschränkt.

FireWire spannt einen 64 Bit-Adressraum auf, welcher sich in 1023 plus eins Busse unterteilt. Die höchste Busnummer bezeichnet immer den lokalen Bus, vergleichbar mit „localhost“ in TCP/IP Netzwerken. Jeder Bus umfasst bis zu 63 Knoten plus eine Broadcast Adresse, wobei jedem Knoten ein Adressraum von 48 Bit zur Verfügung steht. Abbildung 2-1 zeigt diesen Aufbau schematisch. Im allgemeinen wird nur die 1023 (Local Bus) als Busnummer verwendet.

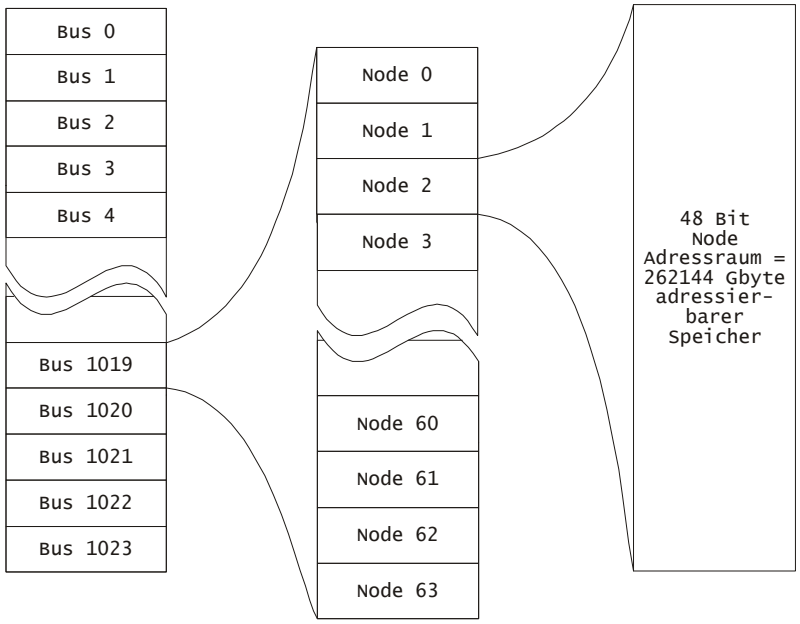

Abbildung 2-1: FireWire Adressräume

FireWire unterstützt zwei Arten des Datentransfers. Die asynchrone Datenübertragung als Quittungsdienst und die isochrone für zeitkritische Daten.

Die asynchrone Datenübertragung ist ein Wechselspiel von Request und Response, wobei stets zwei Geräte kommunizieren. Sowohl der Request als auch die Response werden quittiert; Alle Daten sind durch einen CRC gesichert.

Es gibt drei verschiedene Transaktionen, den Read Request, den Write Request und den Lock Request. Der Read Request liest Daten von einer Adresse oder einem Adressbereich eines anderen FireWire Knotens, der Write Request schreibt Daten in den Adressraum eines anderen FireWire Knotens. Der Lock Request wird benutzt, um unteilbare Aktionen auf einer Speicherstelle auszuführen, so dass „Last-Writer-Wins" Konflikte umschifft werden können.

Die isochrone Datenübertragung wird vor allem für Datenströme benutzt, wie sie beispielsweise von Videokameras produziert werden – die Erzeugung kann nicht unterbrochen werden und vereinzelte Fehler sind weniger schwerwiegend als eine Unterbrechung der Übertragung. Bei isochroner Datenübertragung ist nicht ein einzelnes Gerät Ziel der Daten, vielmehr wird auf einen von 64 virtuellen Kanälen geschrieben. Jedes zum Empfang von isochronen Paketen befähigte Gerät kann nun angewiesen werden, die Daten zu empfangen. Der „Kanal" ist dabei bloß eine Markierung am Paket und nicht zu verwechseln mit einem physikalischen Kanal.

2.1.1 IEEE1394b

IEEE1394b ist die Weiterentwicklung von IEEE1394a und ist ab der Link-Schicht voll abwärtskompatibel, Software muss nicht geändert werden. Wichtige Änderungen sind:

- Neue Steckverbinder
- Neue Medien (LWL, STP, UTP)
- Geschwindigkeit bis 1600Mbit/s
- 8B/10B Kodierung

2.1.2 IEEE1394.1

Erst mit IEEE1394.1 erfährt die zuvor erwähnte Unterteilung in 1024 Busse eine Umsetzung. Informationen über den Stand der Standardisierung sind nur schwer zugänglich. Der Standard befindet offenbar aber noch im Draft Stadium.

Ziele von 1394.1:

1. Mehr als 63 kommunizierende Geräte
2. Verteilung von Geräten auf mehr als einen Bus
3. Verbindung der Busse durch Bridges

In der ersten Phase soll die Architektur sehr einfach sein. Es gibt einen Bus (Branch Bus) an den jeder weitere Bus (Leaf Bus) über eine Bridge angeschlossen ist. Die Anzahl der Bus Hops wird dadurch auf maximal zwei beschränkt (Leaf Bus – Branch Bus – Leaf Bus); Abbildung 2-2 zeigt diese Anordnung.

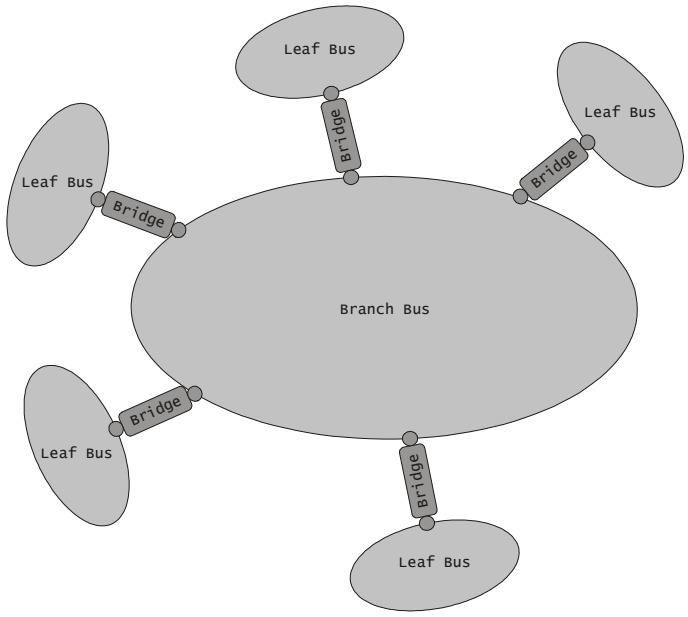

Abbildung 2-2: Branch Bus - Leaf Bus

2.2 1394-based Digital Camera Specification

Die "1394-based Digital Camera Specification" standardisiert eine Kontroll- und Statusregister Architektur, die eine einheitliche Ansteuerung aller Digitalkameras ermöglicht, dazu zählen vor allem Videokameras mit und ohne eigenen Speicher und Photokameras.

Der Standard definiert 17 Features: Helligkeit (Brightness), Automatische Belichtungssteuerung (Auto Exposure), Schärfe (Sharpness), Weißabgleich (White Balance), Farbnuancierung (Hue), Farbsättigung (Saturation), Gamma, Verschlusszeit (Shutter), Verstärkung (Gain), Iris, Fokus (Focus), Farbtemperatur (Temperature), Auslöser (Trigger), Zoom, Pan, Schieflage (Tilt) und Optischer Filter (Optical filter).

Es ist möglich, zu jedem Feature abzufragen, ob das Feature implementiert ist, ob eine automatische Einstellung möglich ist, ob manuelle Kontrolle möglich ist und welcher Wertebereich zulässig ist.

Details sind dem leicht verständlichen Standard [2] zu entnehmen. Einzelne mit dem Autofokus in Verbindung stehende Punkte sind Kapitel 3.4 zu entnehmen

2.3 IEEE1394 in Hardware umgesetzt

Die meisten Hersteller vertreiben drei Arten von FireWire spezifischen Schaltungen, (1.) PHY-Schicht Bausteine, (2.) Link-Schicht Bausteine und (3.) kombinierte Bausteine.

Die Entscheidung fiel für eine Lösung von TI mit zwei Bausteinen (TSB41AB3 & TSB12LV01B), da kombinierte Bausteine offenbar entscheidende Nachteile mit sich bringen; dabei ist vor allem zu nennen:

- Schnittstelle zur Anwendung meist nur PCI

- Fehlende Möglichkeit isochronen Datenverkehr zu hören oder
- Fehlende Möglichkeit asynchrone Lesezugriffe zu initiieren oder
- Fortgeschrittene Spezialisierung auf Anwendungsgebiet, so beispielsweise Schnittstellen zu ATA oder SCSI

Die nächsten zwei Abschnitte beschreiben die beiden ausgewählten Bausteine sowie deren Schnittstellen zur Außenwelt in Kürze.

Weitere Hersteller von Allzweck-FireWire ICs sind Philips und Fujitsu; andere Hersteller stellen offenbar nur FireWire Spezialhardware her.

2.4 Texas Instruments TSB41AB3

Der TSB41AB3 ist ein PHY-Schicht Baustein mit bis zu drei Verbindungen zu anderen FireWire Geräten. Die Anschlüsse des Bausteins lassen sich grob in fünf Gruppen unterscheiden:

(1.) FireWire Anschlüsse

(2.) PHY to Link Schnittstelle

(3.) Versorgung (Strom, Frequenz und andere Größen)

(4.) TI interne Testanschlüsse

(5.) Anwendungsspezifische Anschlüsse

(i) PC0, PC1, PC2

(ii) PD

Für die (1.)-(4.) ist die Belegung vorgegeben. Es ist ausreichend dem schematischen Referenzschaltplan ([6]) zu folgen. Nur die unter (5.) aufgeführten Anschlüsse bedürfen näherer Betrachtung:

(i) Power Class

Da sich Geräte über den Bus mit Strom versorgen können, muss – bevor die Link-Schicht aktiviert werden kann – sichergestellt werden, dass genügend Leistung zur Verfügung steht. Dadurch wird es notwendig diese Information schon auf der PHY-Schicht zur Verfügung zu stellen. Die an den Anschlüssen PC0 bis PC2 angelegten Werte werden zur Programmierung der entsprechenden internen Register benutzt.

Tabelle 2-1: Power Class

PC0-PC2	Beschreibung
000	Node braucht keinen Strom und gibt keinen Strom
001	Node braucht keinen Strom und versorgt den Bus mit 15 Watt
010	Node braucht keinen Strom und versorgt den Bus mit 30 Watt

PC0-PC2	Beschreibung
011	Node braucht keinen Strom und versorgt den Bus mit 45 Watt
100	Node wird vom Bus versorgt und verbraucht 3 Watt oder sie versorgt eventuell auch den Bus. Die Höhe der eingespeisten Leistung ist im Configuration ROM gespeichert.
101	Reserviert
110	Node wird vom Bus versorgt und verbraucht 3 Watt. Weitere 3 Watt werden gebraucht um die Link-Schicht zu aktivieren.
111	Node wird vom Bus versorgt und verbraucht 3 Watt. Weitere 7 Watt werden gebraucht um die Link-Schicht zu aktivieren.

(ii) Power Down

Wenn der PD Anschluss auf High gezogen wird, werden alle internen Schaltkreise bis auf den Repeater deaktiviert und ein Reset ausgelöst. Im normalen Betrieb muss der Anschluss auf Masse gelegt werden.

2.5 Texas Instruments TSB12LV01B

Der TSB12LV01B ist ein FireWire Link-Schicht Kontroller (LLC – Link Layer Controller). Er verfügt über eine „PHY to Link" Schnittstelle, die es ermöglicht Daten aus den Sende-FIFOs mit 100, 200 oder 400Mbits/s auf den PHY-Schicht Baustein zu schreiben. Ein 32-Bit Hostbus fungiert als Schnittstelle zur Anwendung. Der TSB12LV01B empfängt und übermittelt nur korrekt formatierte 1394 Pakete. CRC Prüfsummen werden selbstständig überprüft und generiert. Es können isochrone Daten auf bis zu zwei Kanälen empfangen werden. Der LLC ermöglicht es der Anwendung auch auf die Kontroll- und Statusregister des PHY-Schicht Kontrollers zuzugreifen.

Der interne 2 KByte FIFO Pufferspeicher kann variabel auf die drei Puffertypen aufgeteilt werden – allgemeiner Empfangspuffer (GRF), asynchroner (ATF) und isochroner (ITF) Sendepuffer.

Features des Bausteins:

- Unterstützt den IEEE 1394a-1995 Standard
- Empfängt und sendet nur korrekt formatierte Pakete
- Unterstützt asynchrone und isochrone Datenübermittlung
- Kann als „Cycle-Master" fungieren
- Erzeugt und überprüft 32-Bit CRC Prüfsummen der Paketheader
- Findet verlorene „Cycle-Start" Nachrichten
- PHY to Link Schnittstelle
    - Geschwindigkeit: 100, 200 oder 400 Mbits/s
    - Timing nach IEEE 1394a-2000

- Hostbus Schnittstelle
    - Schnittstelle zum Chip selber wie auch zum FireWire
    - Interrupt gesteuert um Polling zu vermeiden

Die Steuerung des TSB12LV01B geschieht ausschließlich über die Hostbus Schnittstelle – einem Speicherinterface, d.h. es gibt außer Adress- und Datenleitungen nur wenige Steuerleitungen. Der Abschnitt „2.5.2 Hostschnittstelle“ beschreibt diese Schnittstelle grob.

2.5.1 Der Abschnitt „1.1.1

Registerabbildung im Speicher“ beschreibt die Konfigurationsmöglichkeiten des ICs und schlägt eine Konfiguration für den Autofokus vor.

Das Senden und Empfangen von FireWire Paketen ist Gegenstand der Abschnitte „2.5.4 Paketformate“ und „2.5.5 Sende- und Empfangspuffer“.

2.5.2 Hostschnittstelle

Die Schnittstelle zur Anwendung stellt sich beim TSB12LV01B über 32 bidirektionale Datenleitungen, 8 Adressleitungen und 4 Steuerleitungen dar. Lese- und Schreiboperationen folgen immer einem ähnlichen Muster. Abbildung 2-3 illustriert das Lesen und Schreiben im einfachsten Fall.

Als Sonderfall seien die Burst-Read und -Write Zugriffe genannt.

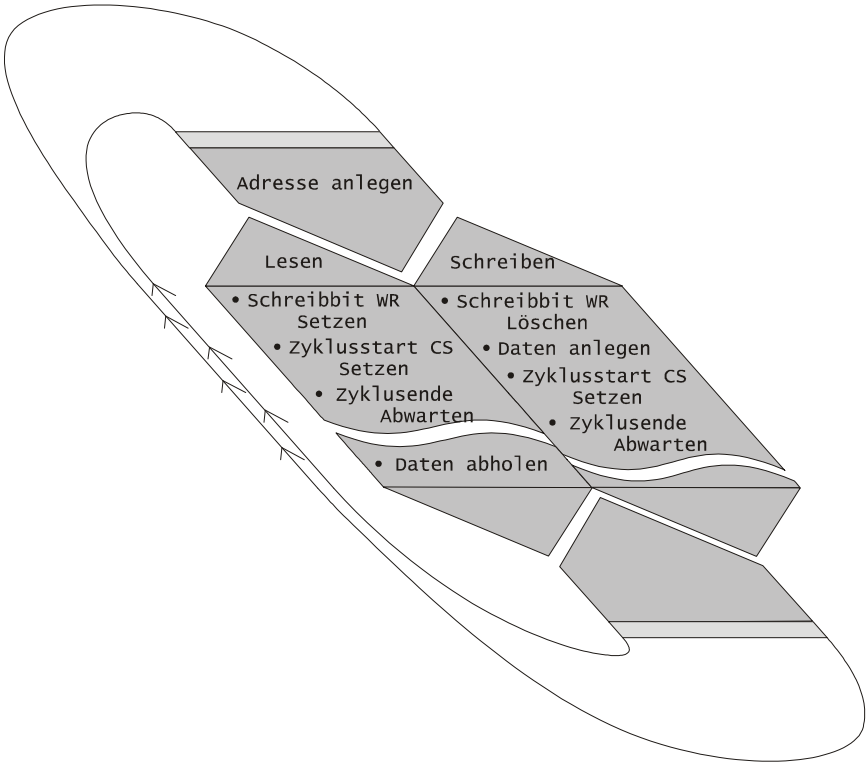

Abbildung 2-3: Lese- und Schreibzyklus

Für Details der Schnittstelle und Timingdiagramme sei auf [3] verwiesen.

(i) Datenleitungen (E/A)

Data[0-31]. Data[0] ist das höchstwertigste Bit. Data[0-7] ist das höchstwertigste Byte.

(ii) Adressleitungen (E)

Addr[0-7]. Addr[0] ist das höchstwertigste Bit. Da der Speicher an 4 Byte Grenzen ausgerichtet ist sind Addr[6-7] stets auf 0 zu legen.

(iii) CA (A)

Durch ein Low auf CA wird dem Hostbus das Ende eines Zyklus signalisiert.

(iv) CS (E)

Ein Low auf CS signalisiert dem TSB12LV01B den Beginn eines Zyklus.

(v) INT (A)

Ein Low auf INT signalisiert eine Unterbrechung. Siehe: „2.5.3.2 Interrupt (0Ch) und Interrupt Mask (10h) Register".

(vi) WR (E)

Fungiert als Umschalter für die Datenleitungen von Eingabe zu Ausgabe. Um auf den Hostbus zu schreiben muss eine 0 angelegt werden, für Leseoperationen eine 1.

2.5.3 Registerabbildung im Speicher

Der TSB12LV01B bildet alle Register in den Speicherraum von Adresse 0h bis 47h ab, d.h. ein Lesen, bzw. Schreiben auf diese Adressen wirkt sich direkt auf die Register des Kontrollers aus. Im Folgenden soll versucht werden, die im Zusammenhang mit der Aufgabenstellung wichtigsten Register vorzustellen. Alle Angaben sind [3] entnommen, vielfach sind dort auch weiterführende Informationen zu finden.

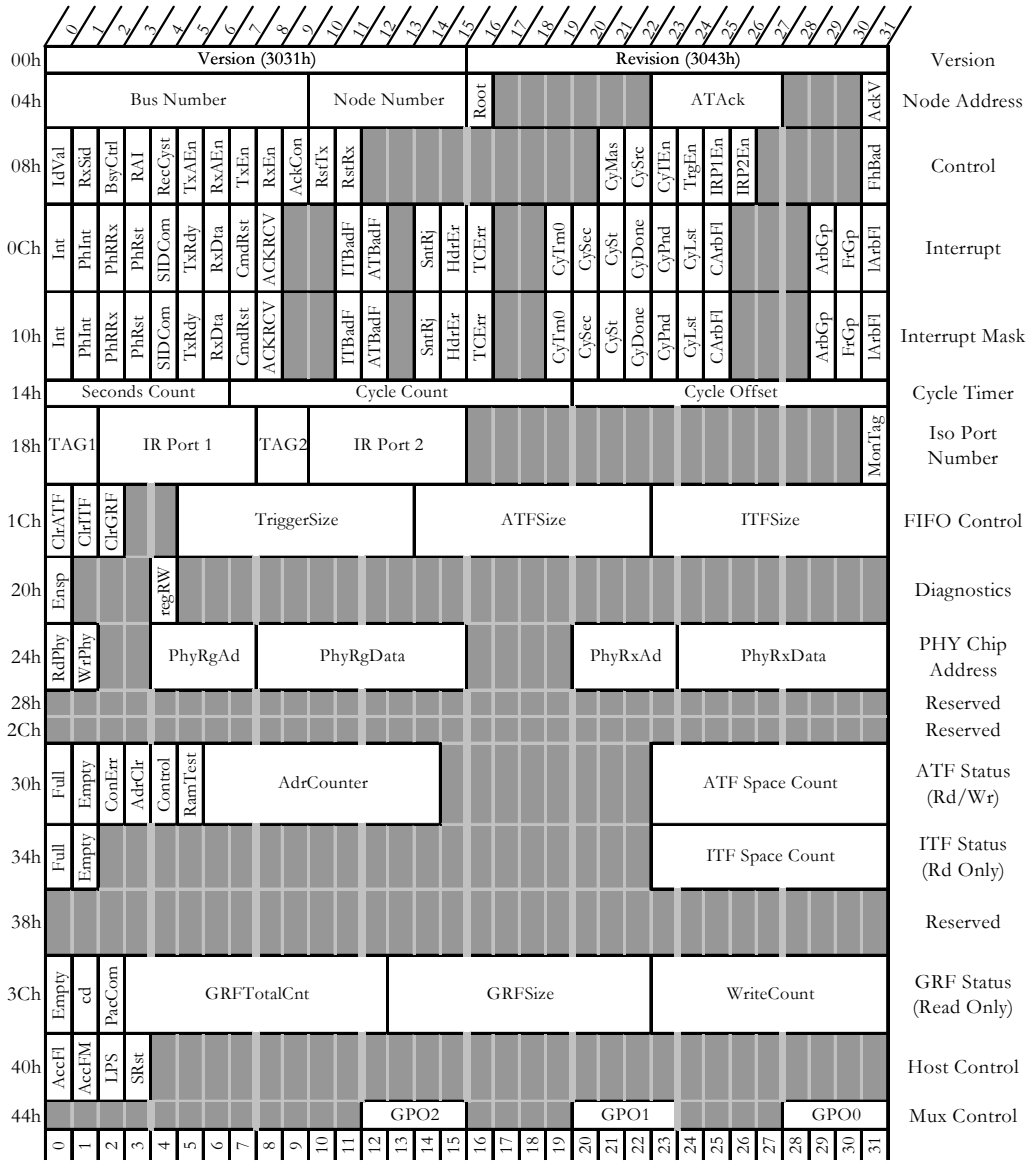

Abbildung 2-4: Register Speicher Abbildung

2.5.3.1 Control Register (08h)

Nach einem Reset werden alle Bits gelöscht. Einige Bits müssen im Normalbetrieb gesetzt werden. Kapital 3.2 beschreibt welche Bits abhängig vom Betriebszustand gesetzt werden müssen.

(i) ID Valid (Bit 0)

Wenn dieses Bit gelöscht ist, werden nur Broadcast Pakete empfangen. Um den Sender zu aktivieren ist dieses Bit zu setzen.

(ii) Receive Self-Id Packets

Das Setzen dieses Bits sorgt dafür, dass „Self-Id", „Link-On" und alle weiteren PHY-Schicht Pakete empfangen werden.

(iii) Transmit Asynchronous Enable (Bit 5)

Wenn dieses Bit gelöscht ist, wird kein asynchrones Paket versendet. Um den Sender zu aktivieren ist dieses Bit zu setzen.

(iv) Receive Asynchronous Enable (Bit 6)

Wenn dieses Bit gelöscht ist werden keine asynchronen Pakete empfangen. Zum Empfangen asynchroner Pakete sollte dieses Bit gesetzt werden.

(v) Receive Isochronous Enable (Bit 8)

Wenn dieses Bit gelöscht ist werden keine isochronen Pakete empfangen. Zum Empfangen isochroner Pakete sollte dieses Bit gesetzt werden.

(vi) Acknowledge Complete Enable (Bit 9)

Nur wenn dieses Bit gesetzt ist, wird auf der Link Schicht das „Acknowledge Packet" (Empfangsquittung) gesendet und Pakete aus Sicht eines initierenden Geräts erfolgreich empfangen.

(vii) Trigger Size Function Enabled (Bit 23)

Dieses Bit erlaubt die An- und Abschaltung der Paketpartionierung. Siehe dazu: 2.5.3.4(ii)

(viii) IR Port 1 Enable (Bit 24)

Wenn dieses Bit gelöscht ist, werden keine isochronen Pakete auf dem ersten isochronen Kanal empfangen. Um auf Kanal 1 zu empfangen, ist dieses Bit zu setzen. Siehe dazu: IR Port (Bit [2-7]) 1 und 2 (Bit [10-15]) in Kapitel 2.5.3.3(ii).

(ix) IR Port 2 Enable (Bit 25)

Wenn dieses Bit gelöscht ist, werden keine isochronen Pakete auf dem zweiten isochronen Kanal empfangen. Um auf Kanal 2 zu empfangen, ist dieses Bit zu setzen. Siehe dazu: IR Port (Bit [2-7]) 1 und 2 (Bit [10-15]) in Kapitel 2.5.3.3(ii).

(x) Flush Bad Packets (Bit 31)

Durch setzen dieses Bits werden alle Pakete verworfen, deren Fehlerhaftigkeit schon durch CRC-Fehler oder Unvollständigkeit aufgefallen sind. Achtung: Schaltet Bit 23 (vii) aus.

2.5.3.2 Interrupt (0Ch) und Interrupt Mask (10h) Register

Der TSB12LV01B signalisiert durch eine logische 0 am INT Anschluss das Auftreten eines Interrupts. Nachdem eine Unterbrechung durch INT = 0 signalisiert wird, ist es notwendig das Interrupt Register zu lesen, um zu erkennen welche spezifische Unterbrechung ausgelöst worden ist. Jedem Bit des Quadlets ist einer Unterbrechung zugeordnet.

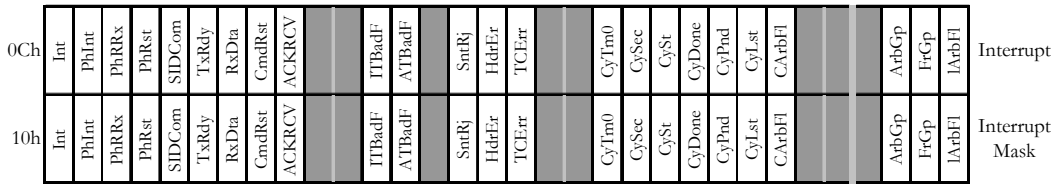

Abbildung 2-5: Interrupt und Interrupt Mask Register

Um nicht jede Unterbrechung behandeln zu müssen, werden nur Unterbrechungen signalisiert, deren Bits im Interrupt Mask Register gesetzt sind. Nach einem Reset sind alle Interrupt Mask Bits gelöscht.

Im Folgenden werden alle im Normalbetrieb interessanten Unterbrechungen beschrieben.

(i) Interrupt (Bit 0)

Das erste Bit ist direkt mit dem INT Anschluss verbunden; Es ist die NOR-Verknüpfung aller „Interrupt AND Interrupt Mask" -Verknüpfungen.

Es ist keiner spezifischen Unterbrechung zugeordnet.

(ii) Phy reset started (Bit 3)

Wenn diese Unterbrechung ausgelöst wird, beginnt der FireWire Bus Reset. Auf diese Unterbrechung sollte in jeder Situation gehört werden, da von diesem Zeitpunkt an keine Transfers zum FireWire Bus mehr stattfinden dürfen.

(iii) Receiver Has Data (Bit 6)

Diese Unterbrechung signalisiert, dass im GRF Pakete vorliegen. Muss gesetzt sein, um Pakete interruptgesteuert zu empfangen.

(iv) Command Reset Received (Bit 7)

Die „Command Reset Received" Unterbrechung wird ausgelöst, wenn eine Node durch Schreiben in die CSR einen Reset ausgelöst hat. Das korrekte Verhalten ist das sofortige Rücksetzen der angeschlossenen Autofokus-Hardware.

(v) Bad Packet Formatted In ITF (Bit 11) & Bad Packet Formatted In ATF (Bit 12)

Die Unterbrechung wird in vier Fällen ausgelöst:

(1.) wenn das erste asynchrone Quadlet nicht auf die A[i]TF_First geschrieben wurde

(2.) ... oder nicht auf A[I]TF_Continue&Update geschrieben wurde

(3.) wenn das Paketformat ungültig ist

(4.) wenn ein Underflow auftritt

In allen Fällen muss der entsprechende Transfer FIFO zurückgesetzt werden. Dies geschieht durch das Setzen von Clear ATF, bzw. Von Clear IRF (siehe 2.5.3.4(i)).

(vi) Cycle Second Incremented (Bit 20)

Wird nach jeder Sekunde (bzw. 8000 * 3072 Takten) ausgelöst und dient der Implementation der CYCLE_TIME (0x200) & BUS_TIME (0x204) Register, siehe (Abschnitt 3.3.2(i)).

2.5.3.3 Isochron Port Register (18h)

Dieses Register steuert den spezifischen Empfang von isochronen Paketen.

(i) Tag 1 (Bit [0-1]) und 2 (Bit [8-9])

Beide Bitfelder sollten jeweils mit [00] belegt sein, da keine anderen Kodes definiert sind. Siehe dazu: 2.5.4.10 - Asynchronous streaming packet – Empfangen.

Wenn der hier hinterlegte Marker nicht mit dem Marker am Paket übereinstimmt, wird das Paket verworfen.

(ii) IR Port (Bit [2-7]) 1 und 2 (Bit [10-15])

Diese beiden Bitfelder geben an, welche isochronen Kanäle mitgehört werden sollen. Nach setzen des Kanals in IR Port 1 oder IR Port 2 ist es notwendig, den jeweiligen Port über IR Port 1 Enable (Bit 24) bzw. IR Port 2 Enable (Bit 25) (siehe 2.5.3.1(viii) und (ix))anzuschalten, erst jetzt werden Pakete empfangen und RxDta Unterbrechungen generiert.

2.5.3.4 FIFO Control (1Ch)

Dieses Register steuert die drei Pufferspeicher. Nach einem Reset belegt der GRF den kompletten Pufferspeicher. Es ist dadurch nicht möglich Daten zu senden. Wenn Daten gesendet werden sollen müssen die jeweiligen Sendepuffer (ITF & ATF) vergrößert werden.

(i) Clear Asynchronous Transfer FIFO (Bit 0), Clear Isochronous Transfer FIFO (Bit 1) & Clear General Receive FIFO (Bit 2)

Das Setzen dieses Bits löscht den jeweiligen FIFO, es ist unnötig das Bit zurückzusetzen, da es sich selbsttätig zurücksetzt.

(ii) Trigger Size in Quadlets (Bits [5-13])

Wenn das Trigger Size Function Bit gesetzt ist, werden alle ankommenden Pakete in Pakete mit einer Nutzdatenlänge von maximal der eingestellten Größe geteilt. Das letzte Teilpaket ist mit PacCom=1 gekennzeichnet (vgl. 2.5.4.11(viii)).

(iii) Asynchronous Transmitter FIFO Size (Bits [14-23]) & Isochronous Transmitter FIFO Size (Bits[25-32])

Die Größe der jeweiligen Pufferspeicher gemessen in Quadlets. Die Verhältnisse der Puffergrößen gehorchen folgender Gleichung:

```
GRF Größe = (512 – [ATF Größe] – [ITF Größe]) Quadlets
```

2.5.3.5 GRF Status

(i) GRF Controller Bit (Bit 1)

Wenn dieses Bit gesetzt ist, ist das nächste Quadlet auf dem GRF ein Paket Header Quadlet.

(ii) Packet Complete (Bit 2)

Wenn (i) gesetzt ist und dieses Bit ebenfalls gesetzt ist, so ist das nun folgende Teilpaket das letzte eines ursprünglichen FireWire Pakets

2.5.4 Paketformate

Die Paketformate werden in aller Ausführlichkeit in [3] beschrieben. Hier sollen nur die im Rahmen des Projekts verwendeten Pakete beschrieben werden. Jedes Paket wird mit einem Paket Header begonnen, unabhängig davon, ob es gesendet oder empfangen wird. Das Senden von isochronen Daten bildet hier eine Ausnahme, da der Header strukturell nicht mit den restlichen Headern vergleichbar ist.. Dieser Fall ist für den Autofokus allerdings nicht von Belang, da keine isochronen Daten gesendet werden.

2.5.4.1 Paket Header

Alle Header haben zur Unterscheidung den Transaktionskode (tCode) auf Bits [24 – 27] hinterlegt.

Der tCode gibt an welche Paketart vorliegt. Tabelle 2-1 verschafft einen Überblick über die Transaktionsarten, die Transaktionskodes sind durch IEEE 1394 spezifiziert. Nur Kode „E“ ist implementationsabhängig und kündigt beim TSB ein PHY-Schicht Paket[1] an.

Die gewählte Reihenfolge der Vorstellung paart der Pakete. Wenn ein Read [Write] Request gesendet (empfangen) wird, wird im nächsten Schritt eine Read [Write] Response empfangen (gesendet).

Transaktionsname	Transaktionskode
Write Request for data quadlet (2.5.4.6 & 2.5.4.8)	0

[1] Wie z.B. “Link-On” oder “Receive Self-Id”

Transaktionsname	Transaktionskode
Write Request for data block	1
Write Response (2.5.4.7 & 2.5.4.9)	2
Reserved	3
Read Request for data quadlet (2.5.4.2 & 2.5.4.4)	4
Read Request for data block	5
Read Response for data quadlet (2.5.4.3 & 2.5.4.5)	6
Read response for data block	7
Cycle start	8
Lock request	9
Asynchronous streaming packet (2.5.4.10)	A
Lock response	B
Reserved	C
Reserved	D
PHY-Schicht Paket (TSB12LV01 spezifisch)	E
Reserved	F

Tabelle 2-2: Transaktionskodes

2.5.4.2 Read Request for data quadlet – Senden

Der Request ermöglicht es dem TSB12LV01 ein einzelnes Quadlet aus dem FireWire Adressraum zu lesen. Abbildung 2-6 zeigt das verwendete Paketformat. Die Antwort auf diesen Request ist „Read Response for data quadlet – Empfangen“ (2.5.4.3).

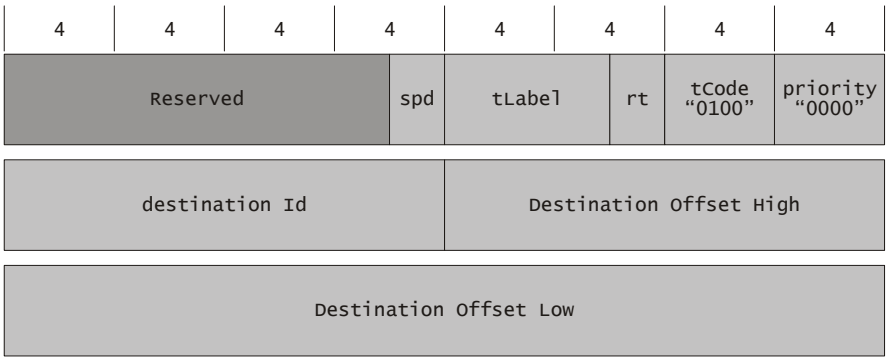

Abbildung 2-6: TX - Read Request for data quadlet

2.5.4.3 Read Response for data quadlet – Empfangen

Stellt das Gegenstück zu „Read Request for data quadlet – Senden" dar und überträgt ein Quadlet an den TSB12LV01 von einer zuvor dazu aufgeforderten Node.

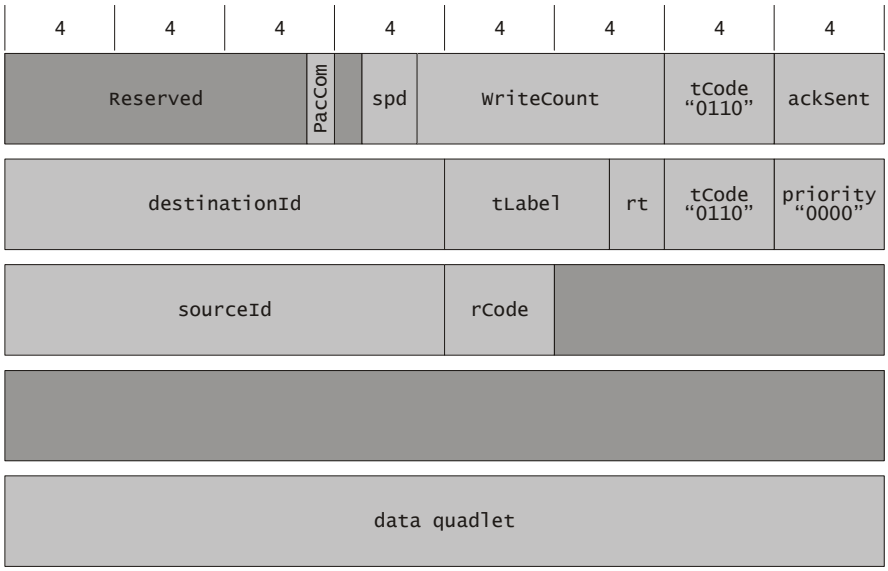

Abbildung 2-7: RX - Read Response for data quadlet

2.5.4.4 Read Request for data quadlet – Empfangen

Dieses Paket fordert den TSB12LV01 – und damit die Anwendung – auf den Inhalt der gegebenen Speicheradresse auszulesen und durch „Read Response for data quadlet – Senden" (2.5.4.5) zu antworten.

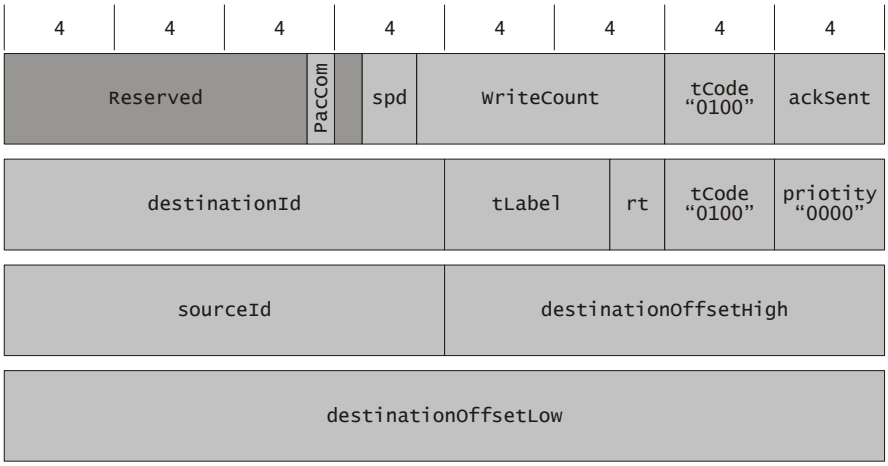

Abbildung 2-8: RX - Read Request for data quadlet

2.5.4.5 Read Response for data quadlet – Senden

Dieses Paket ist die Antwort auf „Read Request for data quadlet – Empfangen“ (2.5.4.4). Gibt Inhalt einer Speicherstelle an den anfragenden Transaktionspartner zurück. Wenn die angesprochene Speicherstelle nicht belegt ist, so wird das Paket mit beliebiger quadlet data und einen rCode = Resp_address_error gesendet.

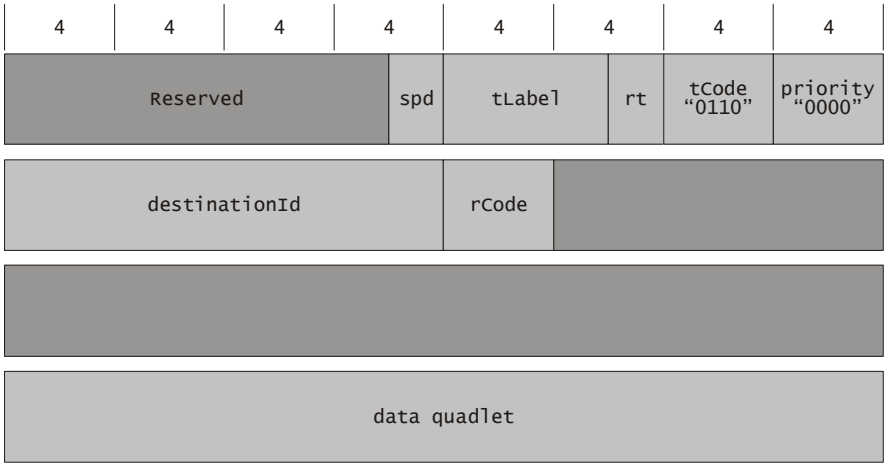

Abbildung 2-9: TX - Read Response for data quadlet

2.5.4.6 Write Request for data quadlet – Senden

Das Paket schreibt ein einzelnes Quadlet in den FireWire Adressraum und damit natürlich in den Adressraum einer beliebigen Node. Das Paket wird gemäß Abbildung 2-10 strukturiert in den ATF geschrieben. Es ähnelt dem „Read Request for data quadlet – Senden“ Paket, doch ist der tCode verschieden und ein Quadlet Nutzlast angehängt.

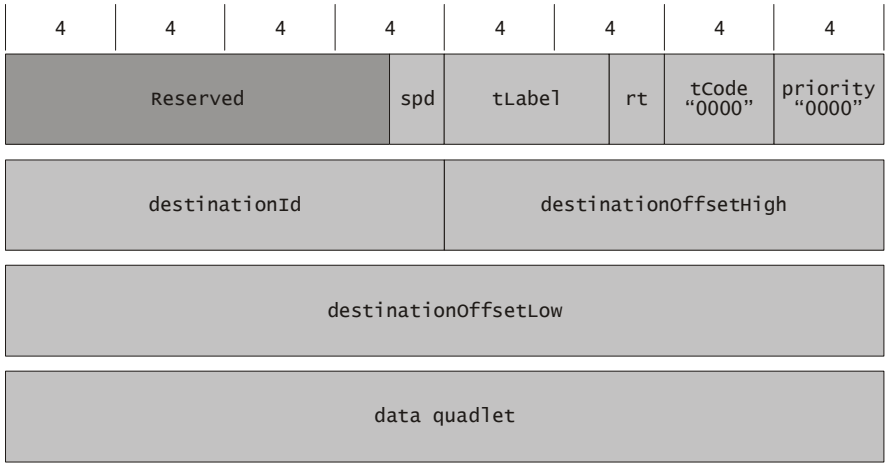

Abbildung 2-10: TX - Write Request for data quadlet

2.5.4.7 Write Response – Empfangen

Das Paket ist eine Reaktion auf „Write Request for data quadlet – Senden". Der rCode ist zu überprüfen, der Schreibvorgang war nur erfolgreich, wenn rCode gleich Resp_complete (=0) ist. Abbildung 2-11 zeigt die Struktur des Pakets im GRF.

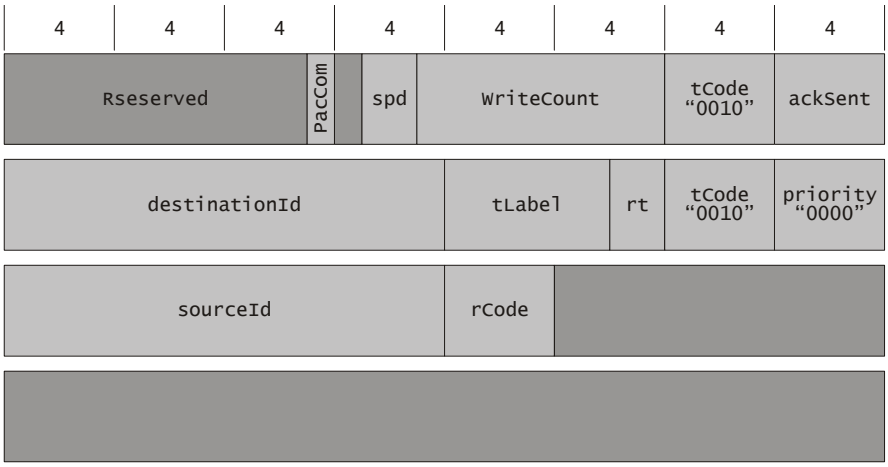

Abbildung 2-11: RX - Write Response

2.5.4.8 Write Request for data quadlet – Empfangen

Ein ankommender „Write request for data quadlet" fordert die betreffende Node auf Daten auf eine gegebene Adresse innerhalb seines Node Adressraums zu schreiben. Abbildung 2-12 zeigt in welcher Struktur ein solches Paket im GRF liegt.

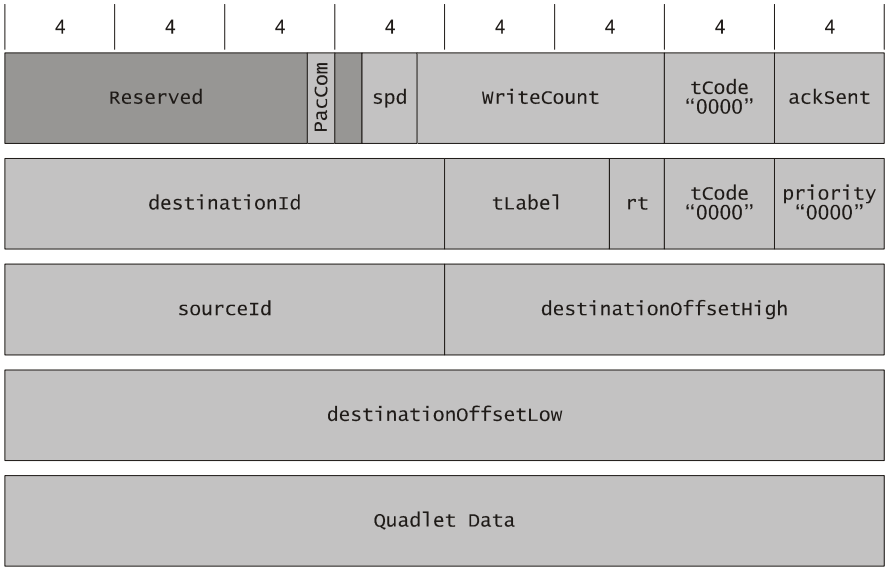

Abbildung 2-12: RX - Write Request for data quadlet

2.5.4.9 Write Response – Senden

Das Write Response Paket ist das Antwortpaket auf „Write Request for data quadlet – Empfangen“. Abbildung 2-13 zeigt die Struktur des Pakets.

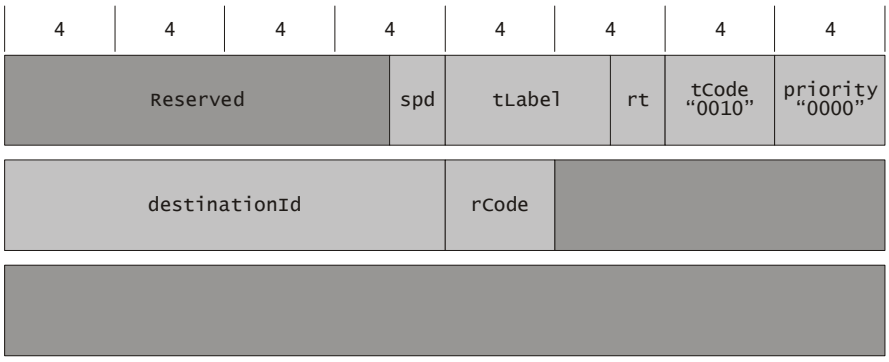

Abbildung 2-13: TX - Write Response

2.5.4.10 Asynchronous streaming packet – Empfangen

Eigentlich ist das „asynchronous streaming packet“ ein isochrones Paket, doch benutzt der FireWire Standard den missverständlichen Ausdruck “asynchron”. Da Paket nicht an eine Node adressiert ist, sondern an einen Kanal, wird es nur empfangen, wenn der entsprechende Kanal und Tag im Isochron Port Register (18h) (siehe 2.5.3.3) eingestellt ist. Die Nutzlast ist variabel aber trotzdem je nach Datenrate begrenzt (dazu Tabelle 2-4: Verhältnis Datenrate - Nutzlast). Abbildung 2-14 zeigt den Aufbau des Pakets.

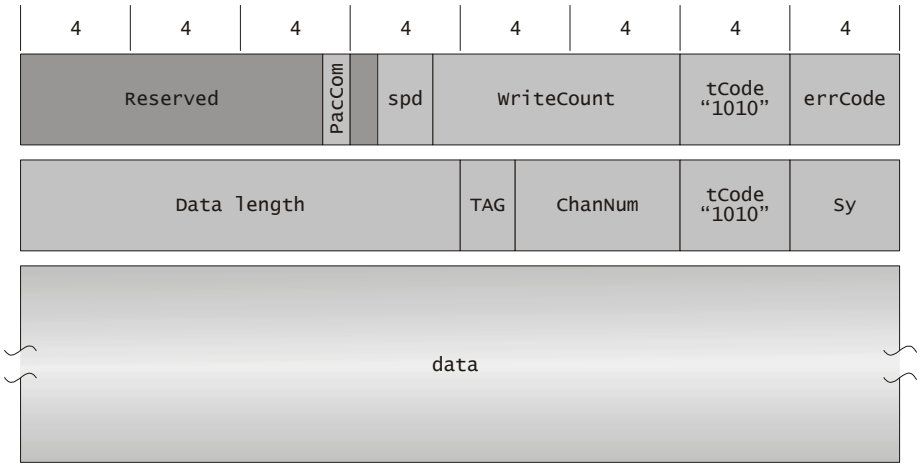

Abbildung 2-14: RX – Asynchronous Streaming Paket

2.5.4.11 Paket Felder

In diesem Kapitel werden die in den vorangegangen Abschnitten benutzten Feldnamen in alphabetischer Reihenfolge erläutert.

Spd	Geschwindigkeit
00	100Mbits/s
01	200Mbits/s
10	400Mbits/s
11	Nicht definiert

Rt	Retry Art
00	New
01	Retry_X
10	Retry_A
11	Retry_B

Tabelle 2-3: Geschwindigkeits- und Retry Kodes

Datenrate	Maximale Nutzlast
100 Mbits/s	512 Byte
200 Mbits/s	1 KByte
400 Mbits/s	2 KByte
800 Mbits/s	4 KByte
1600 Mbits/s	8 KByte
[3200 Mbits/s][1]	[16 KByte]

Tabelle 2-4: Verhältnis Datenrate - Nutzlast

(i) ackSent

Das ackSent Feld enthält den Bestätigungskode der an den anfragenden Knoten gesendet wurde. Die Kodierung dieses 4 Bit Feldes ist Tabelle 2-5 dargestellt.

Kode	Name	Bedeutung
0x0	Reserved	
0x1	Ack_complete	Zielknoten hat das Paket akzeptiert. Transaktion beendet.
0x2	Ack_pending	Der Knoten hat das Paket akzeptiert und wird später ein Antwortpaket generieren.
0x3	Reserved	
0x4	Ack_Busy_X	Der Zielknoten hat das Paket nicht akzeptiert. Eventuell kann das Paket zu einem späteren Zeitpunkt akzeptiert werden.
0x5	Ack_Busy_A	Paket nicht akzeptiert. Retry Mimik A für erneutes Senden benutzen.

[1] Noch nicht spezifiziert

Kode	Name	Bedeutung
0x6	Ack_Busy_B	Paket nicht akzeptiert. Retry Mimik B für erneutes Senden benutzen.
0x7 – 0xC	Reserved	
0xD	Ack_data_Error	Paket nicht akzeptiert wg. CRC Fehler oder fehlerhafte Länge. Nur für Block Pakete benutzt
0xE	Ack_type_Error	Paket nicht akzeptiert wg. fehlerhafter Werte im Header.
0xF	Reserved	

Tabelle 2-5: Acknowledge Kodes

(ii) ChanNum

Das ChanNum Bitfeld enthält die Nummer des isochronen Kanals auf dem das Paket gesendet wurde.

(iii) Data

Das Data Feld enthält die Nutzlast des Pakets. Die Länge in Quadlets wird durch WriteCount (xvii) bestimmt. Die Länge in Bytes wird durch (iv) bestimmt.

(iv) DataLength

Anzahl der Bytes, die mit dem Paket übertragen werden. WriteCount allein ist nicht ausreichend, da es nur quadletgenau ist: Bei einem WriteCount von vier kann die DataLength Werte zwischen 13 und 16 einnehmen. Die überschüssigen – aber trotzdem übertragenen – Bytes haben keinen sinnvollen Inhalt.

(v) DestinationId

Enthält die Bus Nummer und die Node-Id des Ziels. Der Bus Nummer Anteil von DestinationId sollte z.Z. immer 1023 für „Local Bus“ enthalten.

(vi) DestinationOffsetLow /DestinationOffsetHigh

Enthält die Adresse innerhalb des Knotenadressraumes. Siehe Abbildung 3-2 zeigt den Adressraum. Achtung: Adresse muss immer durch 4 teilbar sein (quadlet aligned).

(vii) errCode

Zeigt an, ob das Paket korrekt empfangen wurde. Die Kodierung entspricht der des ackSent Felds (siehe (i)) allerdings sind nur die Werte „DataErr“, („CRCErr[1]“) und „Complete“ zulässig.

(viii) PacCom

Wenn PacCom=1 ist das abzuholende Paket das letzte Teilpaket des ursprünglich empfangenem einzelnen FireWire Pakets. Beim Empfang von Paketen, die nur ein Quadlet als Nutzlast tragen, ist PacCom immer 1.

PacCom=0 kann nur auftreten wenn die Partitionierung von Paketen (vgl. 2.5.3.1(vii)) angeschaltet ist und die Nutzlast des Pakets größer als die eingestellte Trigger Size ist (vgl. 2.5.3.4(ii)). Im Falle des Autofokus können das nur isochrone Pakete sein.

(ix) Priority

Reserviert für zukünftige Backplaneimplementationen, zur Zeit immer 0x0000.

(x) Quadlet data

32 Bit Nutzlast eines Pakets.

(xi) rCode

Der Result Code gibt Auskunft über den Abschluss einer Transaktion. Mögliche rCodes und ihre Bedeutung sind in Tabelle 2-6 abgebildet.

Rcode	Name	Erläuterung
0	Resp_complete	Transaktion erfolgreich
1-3	Reserved	
4	Resp_conflict_error	Ressourcenkonflikt auf Seite des Antwortenden entdeckt. Erneuter Versuch könnte Problem lösen.
5	Resp_data_error	Hardwarefehler. Übertragende Daten sind ungültig.
6	Resp_type_error	Ein Feld im Request Paket Kopf enthält ungültige Werte. Auch für Schreibversuche auf Nur-Lesen Adressen.
7	Resp_address_error	Die angegebene Adresse ist nicht zugreifbar
8 – 0xF	Reserved	

Tabelle 2-6: Response Kodes

[1] CRCErr ist im Final Draft des Standards – neuere Versionen standen nicht zur Verfügung – nicht ausgewiesen. Die einzige Quelle dafür ist das Datenblatt des TSB12L01B. Die Kodierung ist unklar.

(xii) rt

Enthält den Retry Code, der festlegt welches Retry Schema zu benutzen ist, falls Fehler in der Kommunikation auftreten. Siehe dazu: Tabelle 2-3.

Bei Nodes, die kein Retry unterstützen, ist stets 0b01 zu benutzen. Vgl. „Anforderungen an ein FireWire Gerät (4.)“ Seite 33.

(xiii) Spd

Geschwindigkeit mit der übertragen werden soll. Tabelle 2-3 zeigt die möglichen Werte und ihre Bedeutung.

(xiv) TAG

Zeigt an welcher Formatierung die isochronen Daten gehorchen. Z.Z. ist nur 00 für „unformatiert“ definiert. Siehe „2.5.3.3(i) - Tag 1 (Bit [0-1]) und 2 (Bit [8-9])“ auf Seite 17.

(xv) tCode

Siehe dazu „2.5.4.1 - Paket Header“ auf Seite 18.

(xvi) tLabel

Das Transaction Label der Transaktion. Wenn gesendet wird, generiert der Sender eine möglichst noch nicht verwendete Id, das Antwortpaket benutzt diese Id ebenfalls, so dass die Response dem Request zugeordnet werden. Der Empfangsfall funktioniert entsprechend.

(xvii) WriteCount

Enthält die Anzahl der Quadlets, die zu diesem Paket gehören. Wenn diese Anzahl an Quadlets abgeholt wurde, liegt wieder ein Paket Header auf dem GRF.

2.5.5 Sende- und Empfangspuffer

Die Sende- und Empfangspuffer sind nicht in ihre Gänze im Speicher des TSB12LV01B abgebildet; das ist nicht nötig und auch nicht möglich aufgrund der Größe des FIFOs von 2 KByte gegenüber dem adressierbaren Raum von 256 Byte. Da die Puffer alle als FIFO agieren gibt es einen Anfang und ein Ende, der Lesende Agent liest immer vom Anfang, der schreibende schreibt immer an das Ende.

Im Speicher ist also nur der jeweils relevante Bereich abgebildet. Für die Schreibpuffer das Ende; für den Lesepuffer der Anfang. Abbildung 2-15 zeigt die Belegung des entsprechenden Speicherbereichs, bzw. die Adressen der Gucklöcher auf die Pufferanfänge und -enden.

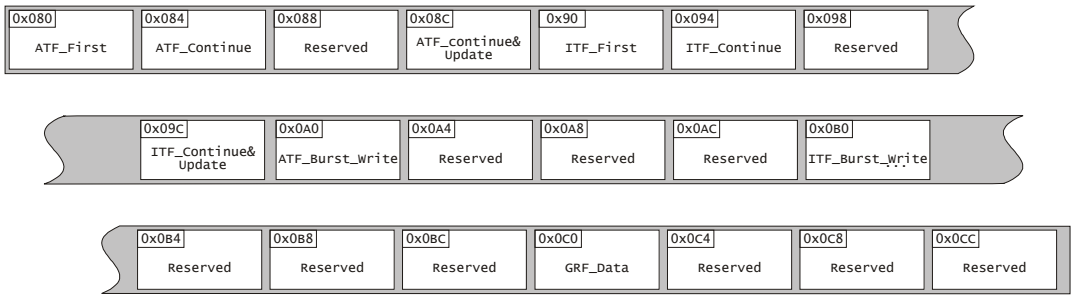

Abbildung 2-15: Abbildung der FIFOs im Speicher

2.5.5.1 ATF & ITF

Beide FIFOs arbeiten auf die gleiche Art und Weise, die Trennung von beiden ist aber aus FireWire Erfordernissen heraus notwendig, denn isochrone müssen Pakete unabhängig von asynchronen Paketen versendet werden können.

Wenn beim Senden ein Fehler durch ein fehlformatiertes Paket oder Underflow ausgelöst wird, wird dies durch einen Bad Packet Interrupt (siehe 2.5.3.2(v) auf Seite 17) signalisiert wird und die Puffer müssen durch ein Schreiben auf das entsprechenden FIFO Control (1Ch) Bit zurückgesetzt werden.

Für beide Transfer FIFOs gibt es jeweils 4 Fenster auf das Pufferende:

*(i) *_First*

Hier wird das erste Quadlet eines Pakets, also der Paket Header hinein geschrieben.

*(ii) *_Continue*

Lädt Daten in den Pufferspeicher ohne sie zum Senden freizugeben. In das *_Continue Fenster sollten bis auf das erste und letzte alle Pakete geschrieben werden.

*(iii) *_Continue&Update*

Durch Schreiben auf *_Continue&Update werden die bisher geschriebenen Daten zum übermitteln freigegeben, d.h. dass entweder schon alle Quadlets geschrieben sein sollten oder dass die Anwendung die nachfolgenden Daten schneller in Continue&Update schreibt, als sie von der Sendeeinheit nachgefragt werden.

Pakete zum Senden freizugeben, bevor alle Daten des Pakets im Puffer liegen, ist fehlerträchtig und sollte nur für geschwindigkeitskritische Anwendungen überhaupt nötig sein. Die Autofokus Implementation sollte davon Abstand nehmen.

*(iv) *_Burst_Write*

Auf die Burst_Write Speicherstellen werden stets ganze Pakete inkl. Header geschrieben und sofort von der Sendeinheit bearbeitet, das Verhalten dieses Registers ist dem Verhalten von

*_Continue&Update sehr ähnlich, doch können die Daten durch die Nutzung des Burst-Schreibmodus schneller geschrieben werden

2.5.5.2 GRF

Ein Fenster auf den Anfang des Empfangsspeichers steht an der Speicherstelle 0x0C0 zur Verfügung. Das GRF Status Register (siehe Abschnitt 2.5.3.5) stellt Informationen über das zu holende Quadlet zur Verfügung; das cd Bit signalisiert – wenn es gesetzt ist, dass das nächste Quadlet ein Paket Header ist.

3 Digitaler Autofokus für FireWire Kameras

Zuerst gibt Abschnitt 3.1 einen Überblick über das geplante System. Abschnitt 3.2 beschreibt wie der TSB12LV01 zu konfigurieren ist um ein FireWire konformes Verhalten vom Anschalten des Geräts bis hin zum Anschalten der Link-Schicht. Anschließend beschreibt Abschnitt 3.3 wie sich das Gerät im Normalbetrieb zu Verhalten hat, sich identifiziert und z.B. die anderen Teilnehmer über eigene Fähigkeiten unterrichtet. Bis zu diesem Abschnitt sind noch keine Autofokus spezifischen Aufgaben bearbeitet worden. Die Arbeit des Autofokus ist dann abschließend Gegenstand von Abschnitt 3.4.

3.1 Systemaufbau

Der Systemaufbau ist einfach gehalten, die erste und zweite Kamera, Host PC und das Autofokus-System sind über den FireWire Bus mit einander verbunden, wie in Abbildung 3-1 dargestellt. Das Autofokus-System besteht aus einem Nios Development Board und einer Tochterplatine, die für die Verbindung zum FireWire Bus sorgt.

Die Tochterplatine enthält im Wesentlichen die zuvor beschriebenen Bausteine TSB41AB3 & TSB12LV01A, sowie die FireWire Steckverbinder und mehrere Pfostenstecker auf denen der Hostbus nach außen geführt wird.

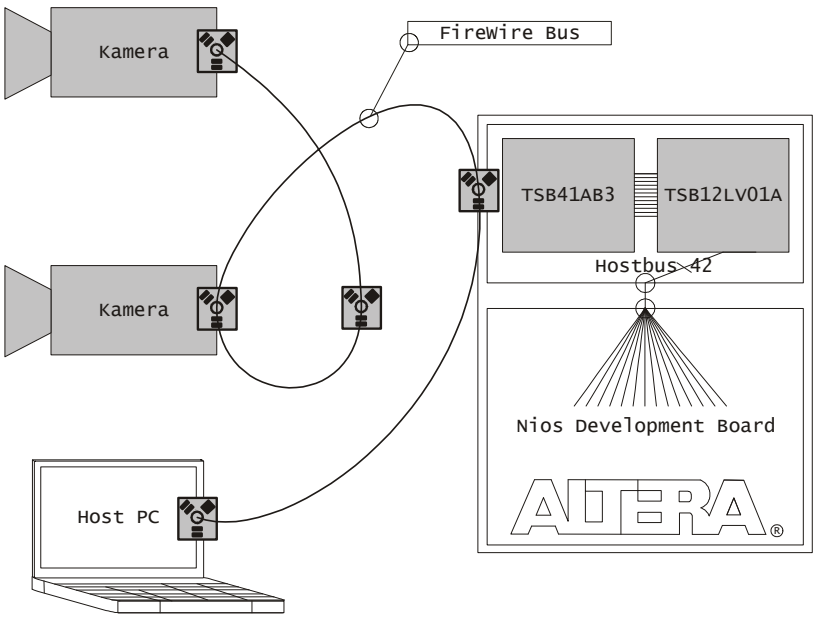

Abbildung 3-1: Systemaufbau

Auf dem NIOS Board befindet sich ein Altera FPGA Baustein, auf dem mit Hilfe des SOPC Builder (System-On-a-Programmable-Chip) ein einfacher CPU mit mindestens 42 I/O Pins erzeugt wird. Diese I/O Pins werden über ein Flachbandkabel mit den Pfostensteckern auf der Tochterplatine und auf diese Weise mit dem Hostbus verbunden.

Ein Compiler gehört ebenfalls zum Lieferumfang des SOPC Builders, so dass der eben erzeugte Prozessor in der Hochsprache C programmiert werden kann. Mit dem CPU werden gleichzeitig Header Dateien – zur Systemarchitektur passend, sowie C Dateien zur Ansteuerung der I/O Ports und Betriebssystemfragmente erzeugt.

3.2 Initialisierung des TSB12LV01B

Die Initialisierung beginnt, sobald das Gerät angeschaltet ist und gliedert sich in 4 Phasen:

Power On, Bus Reset Started, Self Id Complete und Link On.

Es ist nicht nötig diese vier Phasen in der Implementation zu unterscheiden. Die ersten drei Phasen können sogar sehr einfach zusammengelegt werden, da jeweils nur auf einzelne Interrupts gewartet wird, dadurch wird es allerdings nötig einen etwas komplizierteren Interrupthandler zu schreiben.

Der in den folgenden Kapiteln benutzte Pseudocode läuft in Gedanken auf dem per SOPC erzeugten Prozessor ab.

3.2.1 „Power On" Initialisierungsphase

Die „Power On" Phase ist eine aktionslose Wartephase.

Zu Beginn der „Power On" Phase wird der PhRst Interrupt aktiviert um das Ende der Phase zu erkennen.

Die „Power On" Phase endet mit dem Physical Reset Interrupt, der zum Beispiel durch einen Konfigurationsprozess (also durch Verbindung mit einem anderen Gerät) ausgelöst wird.

(i) Pseudocode

```
WriteToHostBus (a_InterruptMask, m_PhRst | m_Int);    // Interupt Mask
While (INT == 1);                                     // Auf Interrupt warten
```

3.2.2 „Bus Reset Started" Initialisierungsphase

Die „Bus Reset Started" Phase ist eine aktionslose Wartephase.

Zu Beginn der „Bus Reset Started" Phase wird der SIDComp Interrupt aktiviert, um das Ende der Phase zu erkennen. Der SIDComp Interrupt wird durch das Ende des Konfigurationsprozesses ausgelöst.

(i) Pseudocode

```
WriteToHostBus (a_InterruptMask, m_SIDComp | m_Int);  // Interupt Mask
While (INT == 1);                                     // Auf Interrupt warten
```

3.2.3 „Self Id Complete" Initialisierungsphase

Die „Self Id Complete" Phase beginnt nachdem der Konfigurationsprozess jedem Gerät eine Node-Id zugewiesen hat. In dieser Phase wird auf den Befehl zur Aktivierung der Link-Schicht oder einen erneuten Bus Reset gewartet.

Der Befehl zur Aktivierung der Link-Schicht wird durch ein an das Gerät adressiertes „Link-On“ Paket gegeben.[1] Im Gegensatz zu früheren Phasen muss das Gerät nun auf mehr als ein Ereignis reagieren, dazu zählt – außer dem „Link-On“ Paket – vor allem der Bus Reset, der das Gerät in die „Bus Reset Started“ Initialisierungsphase zurückfallen lässt. Erstmalig ist ein komplexer Interrupthandler notwendig.

Damit der Empfang von Paketen nicht unbemerkt bleibt, muss der Receiver Has Data (Bit 6) Interrupt aktiviert sein.

Auch „Self Id Complete“ ist eine aktionslose Phase.

(i) Pseudocode

```
WriteToHostBus(a_Control, m_RxSld);        //Control Register
WriteToHostBus(a_InterruptMask, m_Int | m_RxDta | m_PhRst);
       //Interrupt Mask
While (nok) {       // Schleife solange Phase nicht beendet
 While (INT == 1);
 Quadlet quad = ReadFromHostBus(a_Interrupt);
 if (quad & m_PhRst) {
  Goto Bus Reset Started;
 } else if (quad & m_RxDta) {
  quad = ReadFromHostBus(a_GRFData);       // lesen aus GRF
    count = (quad & m_WriteCount) >> o_WriteCount;
  if ((0xE == getTransactionCode(quad)) && // unformatiertes Paket,
       (getErrorCode(quad))) {             // alles Ok kein Fehler
   quad = ReadFromHostBus(a_GRFData);      // lesen aus GRF
   count-=1;
   if ((quad & m_PhyPacketId) ^ (0x4000 0000)) { // PacketId == 01
    id = (ReadFromHostBus(a_NodeAddress)&&m_BusNumber)>>o_BusNumber;
    if ((quad & m_SIDphy_ID) >> o_SIDphy_ID) == id) {
     nok = false;
    }
   }
  }
  while (count != 0) { // Empfangspuffer räumen
   quad = ReadFromHostBus(a_GRFData);      // lesen aus GRF
  }
 }
} // elihW (kon)
```

3.2.4 „Link On“ Initialisierungsphase

Der Eintritt in diese Initialisierungsphase fordert das Gerät auf die Link-Schicht zu aktivieren.

[1] Der Bus Manager entscheidet nach Sichtung der Self Id Pakete, ob genügend Strom bereitsteht und aktiviert dementsprechend die Link-Schichten einiger oder aller am Bus angeschlossenen Geräte.

Im Betrieb ist es erwünscht (1.) Alle an das Gerät gerichteten Pakete zu erhalten, (2.) Asynchrone Pakete zu versenden und (3.) zu empfangen, (4.) Isochrone Pakete zu empfangen und (5.) Den Empfang von Paketen zu quittieren, sowie (6.) Pakete partitioniert zu empfangen.

Im Control Register werden dazu folgende Bits gesetzt:

(1.) ID Valid (Bit 0)

(2.) Transmit Asynchronous Enable (Bit 5)

(3.) Receive Asynchronous Enable (Bit 6)

(4.) Receive Isochronous Enable (Bit 8)

(5.) Acknowledge Complete Enable (Bit 9)

(6.) Trigger Size Function Enabled (Bit 23)

Bedingt durch die aktivierten Funktionalitäten müssen eine Reihe von Interrupts behandelt werden:

(1.) Phy reset started (Bit 3)

(2.) Receiver Has Data (Bit 6)

(3.) Command Reset Received (Bit 7)

(4.) Bad Packet Formatted in ATF (Bit 11)

Zusätzlich wird das FIFO Control (1Ch) Register konfiguriert, so dass alle Pakete in maximal 32 Quadlet große Teile partitioniert werden und der ATF 128 Quadlets halten kann.

(i) Pseudocode

```
// Eigentlicher Start der Init Phase
WriteToHostBus(a_Control, m_IdVal | m_TxAEn | m_RxAEn | m_RxIEn |
       m_AckCEn | m_TrgEn);            // Control Register
WriteToHostBus(a_InterruptMask, m_Int | m_PhRst | m_RxDta | m_CmdRst |
       m_ATBadF);                      // Interrupt Mask
WriteToHostBus(a_FIFOControl, 128 << o_ATFSize | 32 << o_TriggerSize);
       ;                               // FiFO Control
```

3.3 Anforderungen an ein FireWire Gerät

FireWire implementiert den ISO/IEC Standard 13213. Dieser Standard definiert eine generische Busarchitektur für parallele und serielle Busse; weite Teile dieses Standards werden alleine durch die Verwendung der FireWire spezifischen integrierten Schaltkreise abgedeckt und bedürfen somit keiner genaueren Betrachtung.

Kapitel 7 und 8 von ISO/IEC 13213 beschreiben allerdings die Control and Status Register (CSR) und das Configuration ROM. Beide sind im Adressraum der Node in die obersten 2 Kbyte der untersten 256Mbyte abgebildet (siehe dazu Abbildung 3-2).

Die CSR liefern Informationen über den Gerätestatus und ermöglichen auch eine grundlegende Steuerung des Geräts.

Das Configuration ROM nimmt gerätespezifische Informationen auf.

Beide Bereiche können von anderen Geräten am Bus gelesen und zum Teil beschrieben werden (Schreiben als Kommando). Dies geschieht durch die Transaktionspaare „Read [Write] request for data quadlet“ – „Read [Write] response for data quadlet“. Die Generierung der Antworten liegt in der Verantwortung des jeweiligen Herstellers.

Einige Entscheidungen über die Fähigkeiten des Geräts müssen im Vorfeld getroffen werden. Um die Implementation einfach zu halten sei hier vorgeschlagen folgende Punkte nicht zu unterstützen:

(1.) Bus Master

(2.) Cycle Master

(3.) Abdication

(4.) Transaction Retry

(5.) Isochronous Resource Manager

Der Beschreibung der Register entspricht nicht unbedingt der im Standard veröffentlichten Beschreibung, vielmehr wird hier die eine Minimal-Implementation im Zusammenhang mit dem Anwendungsgebiet „Autofokus“ und den auferlegten Restriktionen vorgeschlagen.

Register die hier nicht beschrieben sind müssen/können/sollen nicht implementiert werden, d.h. ein Lesen oder Schreiben führt zu einer Antwort mit dem rCode „resp_address_error“ (=7). Einige Register sind optional, andere werden generell bei FireWire nicht implementiert und wieder andere sind spezifisch für vom Autofokus *nicht* unterstützte Funktionen.

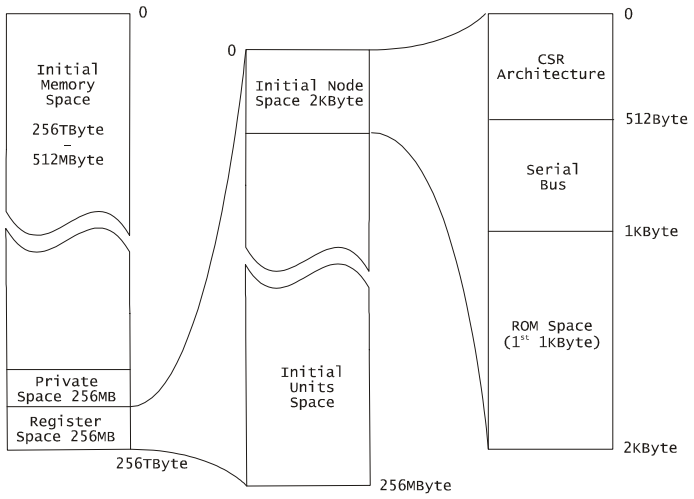

Abbildung 3-2: Adressraum eines FireWire Knotens

3.3.1 ISO/IEC 13213 spezifische Register

In folgenden Kapitel soll der Kernregistersatz eines ISO/IEC 13213 Geräts mit den Besonderheiten des FireWire Standards beschrieben werden. Abbildung 3-3 gibt einen Überblick über den Registeradressraum.

Nicht aufgeführte Register sind zum Teil gänzlich unpassend oder optional in einer FireWire Implementation, dennoch sei für weitergehenden Informationen an dieser Stelle auf [8], bzw. [1] verwiesen.

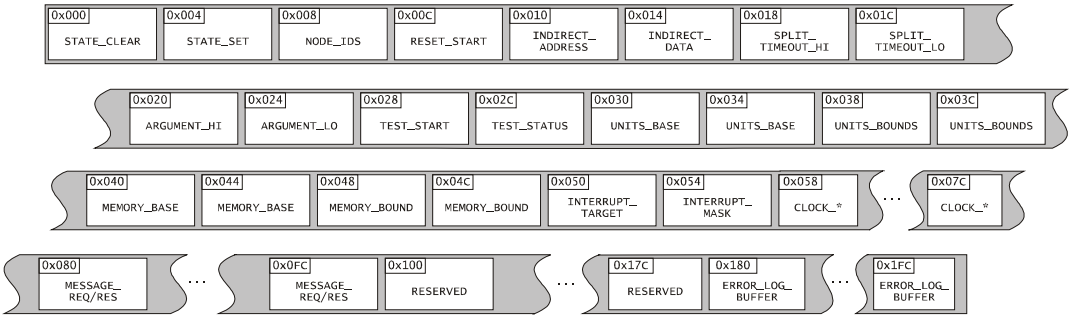

Abbildung 3-3: CSR Kern Register

(i) STATE_CLEAR (0x000) und STATE_SET (0x004)

Das Format beider Register (siehe Abbildung 3-4: STATE Register) ist identisch, da sie nur zwei Sichtweisen auf ein und das selbe Register sind. Ein Schreiben auf STATE_CLEAR löscht die gesetzten Bits, ein Schreiben auf STATE_SET setzt die gesetzten Bits, so dass bei beiden Register ein Schreiben von 0x0000 0000 0000 0000 keine Veränderung verursacht.

```
1: STATE := STATE & ^STATE_CLEAR
2: STATE := STATE | STATE_SET
```

Abbildung 3-4: STATE Register

Das „Unit dependent“ Bitfeld steht dem jeweiligen Gerät zur freien Verfügung. Die Implementation ist gänzlich frei.

Das „Bus dependent“ Bitfeld ist FireWire spezifisch und definiert vier der acht Bits.

- „gone“ – wird im Falle eines Power-, Bus- oder Command Resets auf 1 gesetzt. Ein Gerät, dessen „gone“ gesetzt ist, darf keine Anfragen auf den Bus übermitteln. Offenbar kann dieses Feld immer auf 0 gesetzt bleiben, da Anfragen senden und Anfragen empfangen hier zur gleichen Zeit aktiviert wird. Eine 1 sollte nur gelesen werden können, wenn es nicht möglich ist die Leseanforderung zu empfangen, also kann sie niemals gelesen werden. (*Spekulation – Zweck des Bits unklar*)
- „Reserved“ – reserviert
- „Abdicate“ – wurde eingeführt um die Rolle des Bus Managers zwischen zwei Nodes zu tauschen. Wenn ein Tausch der Bus Manager Rolle zum Vorteil des Geräts nicht nötig oder gewünscht ist kann dieses Bit konstant 0 bleiben.

- „Linkoff“ – das Schreiben einer 1 in dieses Bit sollte eine stromverbrauchende Link-Schicht abschalten. Wenn die Link-Schicht keinen Strom verbraucht kann das Setzen ignoriert werden. Maßgeblich ist die Power Class (siehe 2.4(i)) der Node.
- „Cmstr“ – Nur von Cyclemaster fähigen Nodes zu implementieren. Eine 1 macht diese Node zum Cycle Master. Siehe dazu Bus_info_block (Abschnitt 3.3.3.2)

Das „Lost“ Bit muss vom Gerät gelöscht werden, sobald sie nach einem Reset wieder betriebsbereit ist.

Wenn das „DReq“ Bit gesetzt ist, darf die Node keine Request senden.

„Res“ – reserved

Das „Elog“ Bit steuert das node-interne Fehler Log. Wird für den Autofokus nicht benötigt und kann deswegen immer 0 bleiben.

Das „Atn“ Bit ist für 1394 Nodes „Reserved“, d.h. immer 0.

Das „Off“ Bit ist für 1394 Nodes „Reserved“, d.h. immer 0.

Die “State” Bits geben Auskunft über den Status der Node: 00 = Running, 01=Initializing, 10=Testing in progress und 11=Fatal Error.

(ii) NODE_IDS (0x008)

Das NODE_IDS Register enthält die Identifikation des Geräts. Abbildung 3-5 zeigt das Register. Im oberen Bereich ist die Bezeichnung der Felder im ISO/IEC 13213 Standard genannt und im unteren Bereich der Name des Feldes in der FireWire Implementation.

Das „Bus_Id” Feld enthält die Busnummer. Das Feld sollte initial den Wert 0b1111111111 (1023 – Local Bus) enthalten. Ein Schreiben auf dieses Feld muss den Wert jedoch ändern können.

Das Feld „Offset_id“, bzw. „Physical_Id” enthält die der Node zugewiesenen Node-Id. Ein Schreiben muss ignoriert werden.

Die „Reserved“ Bits müssen immer 0 enthalten und können auch durch einen Schreibvorgang nicht geändert werden.

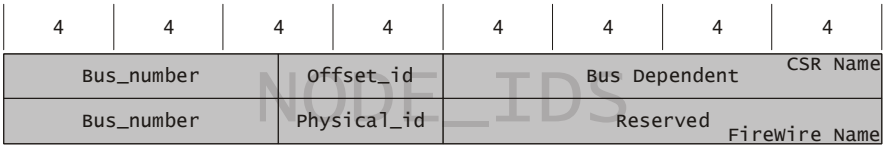

Abbildung 3-5: NODE_IDS Register

(iii) RESET_START (0x00C)

Jedes schreiben in dieses Register (siehe Abbildung 3-6) löst einen Reset des Geräts aus. Wenn das Nt-Bit (Nt = No Test) nicht gesetzt wird, wird optional ein Selbsttest ausgeführt. Kein Endgerät ist verpflichtet diesen Selbsttest zu implementieren.

Ein Lesen des Register lässt die anfragende Node stets 0 erhalten.

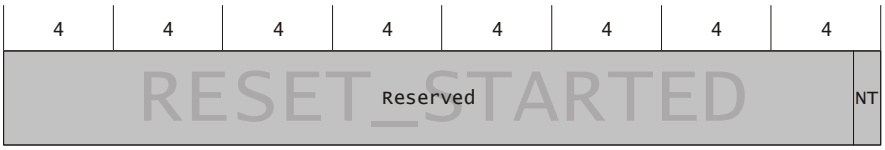

Abbildung 3-6: RESET_START Register

(iv) SPLIT_TIME_HI (0x018) und SPLIT_TIME_LO (0x01C)

Wenn eine Node eine Anforderung erhält steht der Verkehr auf dem Bus nicht still, bis diese Anforderung beantwortet ist – zwischen Request und Response vergeht Zeit. Der Inhalt der SPLIT_TIMEOUT Register zeigt an wie viel Zeit verstreichen darf bis die Antwort eintrifft. Wenn diese Schranke überschritten wird, wird eine noch später eintreffende Antwort verworfen.

Der Mechanismus des zeitgesteuerten Verwerfens ist von der Autofokus Hardware zu implementieren.

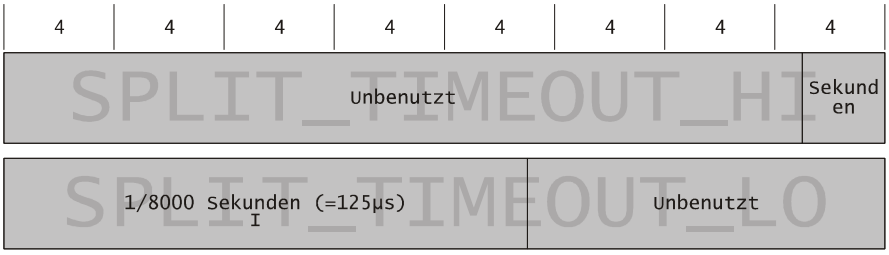

Abbildung 3-7: SPLIT_TIMEOUT Register

3.3.2 FireWire spezifische Register

Im CSR Standard ist von 0x200 bis 0x3FF ein busabhängiger Bereich von Control und Status Registern definiert. Abbildung 3-8 gibt einen Überblick über diesen Speicherbereich in der FireWire Implementation.

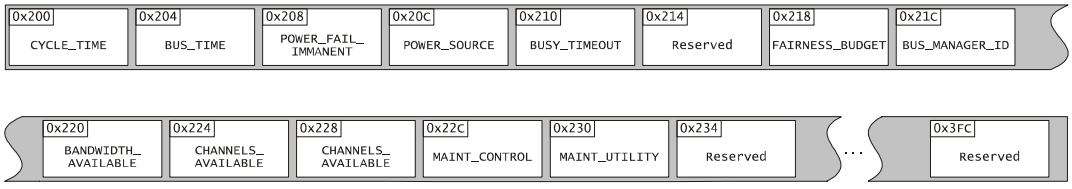

Abbildung 3-8: FireWire spezifische Register

(i) CYCLE_TIME (0x200) & BUS_TIME (0x204)

Das CYCLE_TIME Register ist ein Fenster zum Cycle Timer Register des TSB12LV01 (siehe Abbildung 2-4 Offset 0x14h auf Seite 14).

- Cycle_offset wird durch den 24,576MHz Takt inkrementiert. Nach 3072 Takten sind 125µs verstrichen
- Cycle_count zählt 125µs Zeitabschnitte. Nach 8000 * 125µs ist eine Sekunde verstrichen und Second_count wird inkrementiert.
- Second_count zählt die Sekunden.

Die untersten 7 Bit des BUS_TIME Registers sind Aliase auf die obersten 7 Bit des CYCLE_TIME Registers, die oberen 25 Bit vergrößern den Zahlbereich – beides ist nur bei Cycle Master fähigen Nodes verpflichtend. Zur Implementation ist der Cycle Second Incremented (Bit 20) Interrupt hilfreich, der nach jedem Cycle_count Überlauf ausgelöst wird.

Ein Nutzen – auch des Cycle Time Registers – scheint nur bei Cycle Master fähigen Nodes gegeben zu sein.

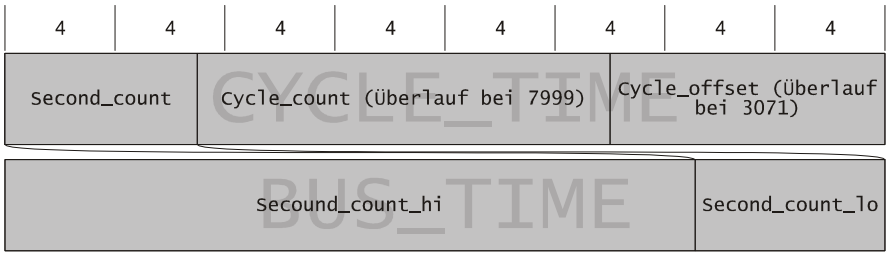

Abbildung 3-9: CYCLE_TIME & BUS_TIME Register

(ii) POWER_FAIL_IMMINENT Register (0x208) & POWER_SOURCE Register (0x20C)

Beide Register werden bei einer kabelgestützten FireWire Implementation im Allgemeinen nicht implementiert. Durch POWR_FAIL_IMMINENT und POWER_SOURCE wird eine Benachrichtigung über einen bevorstehenden Stromausfall ermöglicht.

(iii) BUSY_TIMEOUT Register (0x210)

Wird nur von Nodes implementiert, die „Transaction Retry" unterstützen.

(iv) FAIRNESS_BUDGET Register (0x218)

Offenbar unterstützt der TSB12LV01B keinen "Priority Arbitration Service", so dass das Register keine Funktion erfüllt. Die genaue Verwendung ist unklar, keine hier vorliegende Literatur ist hilfreich.

Schreibvorgänge auf das Register zu ignorieren und auf eine Leseanforderung stets 0x0000 0000 0000 0000 zurückzuliefern erscheint angebracht (*spekulativ*).

(v) BUS_MANAGER_ID (0x21C), BANDWIDTH_AVAILABLE (0x220) & CHANNELS_AVAILABLE Register (0x224)

Alle drei Register werden nur von Nodes implementiert, die die Rolle „Isochronous Resource Manager“ unterstützen, also nicht vom Autofokus.

(vi) MAINT_CONTROL Register (0x22C)

Optionales Register. Ermöglicht absichtliche Fehlererzeugung, wenn implementiert.

(vii) MAINT_UTIL Register (0x230)

Optionales Register. Gibt den zuletzt geschriebenen Wert zurück. Ermöglicht ein seiteneffektfreies Schreiben auf unbekannten Nodes zu Diagnosezwecken.

3.3.3 Configuration ROM

Das im FireWire Standard verwendete Configuration ROM ist nach IEEE 1212-1994 ([7]), bzw. ISO/IEC 13213 Kapitel 8 ([8]) spezifiziert. Es sind zwei Formate spezifiziert: Abbildung 3-10 zeigt das allgemeine ROM Format, Abbildung 3-11 zeigt das minimale ROM Format.

Der Header des allgemeinen ROM Formats besteht aus drei Feldern: Info_length (8 Bit), Crc_length (8 Bit) und Rom_crc_value (16 Bit).

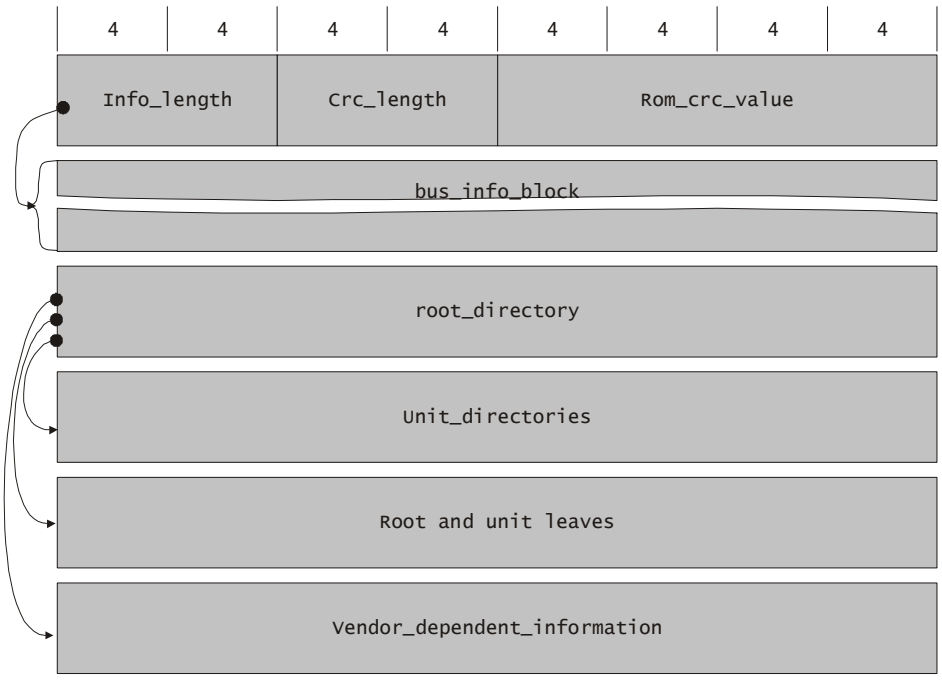

Abbildung 3-10: Allgemeines ROM Format

Dem allgemeinen ROM Format steht das minimale ROM Format gegenüber, es wird durch eine Info_length von 1 gekennzeichnet.

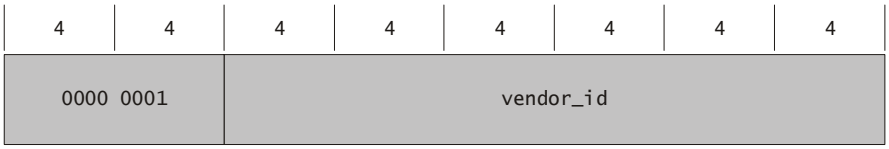

Abbildung 3-11: Minimales ROM Format

3.3.3.1 Header

(i) Info_length

Wenn Info_length gleich 1 ist, so liegt ein minimales ROM vor; alle anderen Längen weisen auf das allgemeine ROM Format. Der für 1394a spezifizierte Bus_info_block ist 4 Quadlets lang. In Zukunft sind auch andere Längen denkbar.

(ii) Crc_length

Länge des mit CRC gesicherten Bereichs. Der „Bus info block" sollte zumindest abgesichert sein.

(iii) Rom_crc_value

CRC Prüfsumme berechnet über die mit Crc_length abgedeckten Quadlets nach dem in 5.1.1 - CRC Berechnung ausgeführten Verfahren.

3.3.3.2 Bus_info_block

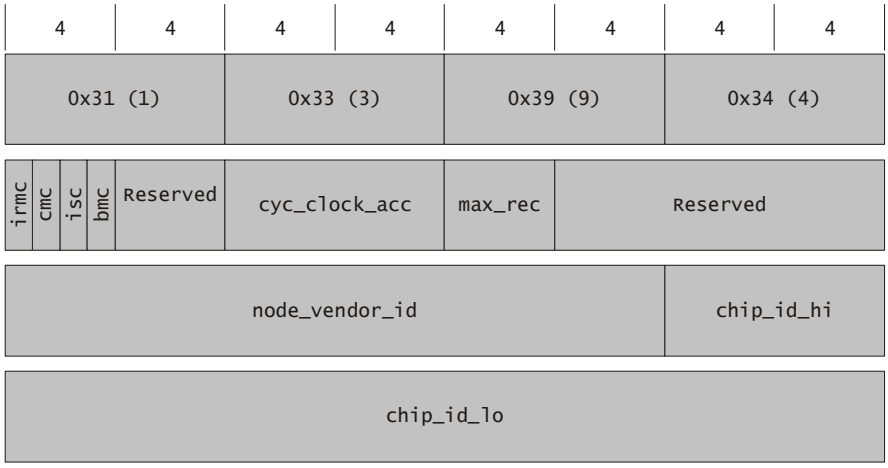

Abbildung 3-12: Bus_info_block

Der Bus_info_block identifiziert die Node global eindeutig und zeigt an welche spezifischen Sonderaufgaben[1] von der Node übernommen werden können. Das erste Quadlet wird durch den CSR Standard vorgegeben und ist die Bus Id („1394"). Abbildung 3-12 zeigt den Aufbau des Bus_info_block.

[1] wie z.B.: Bus Manager, Cycle Master, Isochronous Resource Manager etc.

Der gesamte Bereich des „Bus_info_blocks“ ignoriert jegliche Schreibzugriffe, der „Reserved“ Bereich ist stets mit Nullen besetzt.

(i) Irmc - Isochronous Resource Manager Capable

Für den Autofokus immer 0, der Autofokus ist nicht RM fähig.

(ii) Cmc - Cycle Manager Capable

Für den Autofokus immer 0, der Autofokus ist nicht CM fähig.

(iii) Isc - Isochronous Capable

Muss gesetzt sein, wenn die Node isochrone Transfers unterstützt. Für den Empfang von isochronen Daten muss das Bit nicht gesetzt sein. Für den Autofokus also immer 0.

(iv) Bmc - Bus Manager Capable

Für den Autofokus immer 0, der Autofokus ist nicht BM fähig.

(v) Cyc_clk_acc – Cycle Clock Accurency

Muss für Nodes, die das Cycle Manager Capable nicht gesetzt haben immer 0x1111 1111 sein.

(vi) Max_rec

Gibt die maximale Größe der Nutzlast eines an die Node gerichteten asynchronen Pakets an. Da keine Blockschreibzugriffe auf den Autofokus benötigt werden, ist 0x0000 ein passender Wert.

Die Maximalnutzlast ist $2^{max\_rec + 1}$ Bytes, bzw. 0x0000 für „not specified“.

(vii) Node_vendor_id

Die Node_vendor_id wird jedem Hersteller über das Registration Authority Committee zugewiesen:

Registration Authority Committee (RAC)
The Institute of Electrical and Electronical Engineers, Inc.
445 Hoes Lane
Piscateway, NJ 08855-1331

Für die uni-interne Anwendung „Autofokus“ ist auch ein zufälliger Wert brauchbar.

(viii) Chip_id_hi & Chip_id_lo

Zur freien Verwendung durch den Hersteller des Geräts. Aus “Node_vendor_id” + “Chip_id_hi” + “Chip_id_lo” wird die Unique Node-Id gebildet.

3.3.3.3 Root_directory

Das Root_directory beschreibt die Struktur des weiteren ROMs. Es besteht aus einem Verzeichniskopf und n-Verzeichniseinträgen. Abbildung 3-13 zeigt den allgemeinen Aufbau.

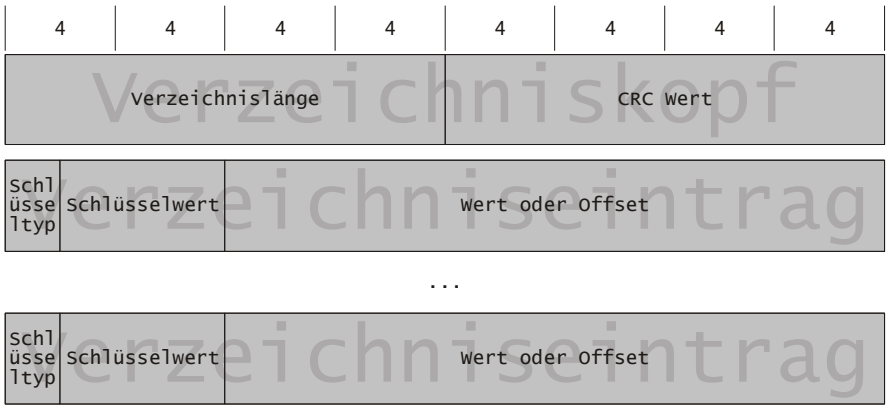

Abbildung 3-13: Configuration ROM Verzeichnisse

Der Verzeichniskopf enthält die Anzahl der Verzeichniseinträge und eine CRC Prüfsumme über das gesamte Verzeichnis.

Key Type (Schlüsseltyp)	Name	Interpretation des 24Bit Werts
0b00	Immediate	Der 24Bit Wert selbst ist die Information
0b01	Offset	Offeset in Quadlets zu einer Speicherstelle
0b10	Leaf	Offset in Quadlets zu einem Leaf, das die Information enthält.
0b11	Directory	Offset in Quadlets zu einem Verzeichnis mit dem gleichen Aufbau wie das Root_directory

Tabelle 3-1: Configuration ROM Verzeichnis - Schlüsseltypen

Die Verzeichniseinträge enthalten drei Werte. Der erste Wert, der Schlüsseltyp, gibt an wie der dritte Wert zu benutzen ist – entweder kann der Wert selbst die Information sein oder er verweist auf eine Speicherstelle, ein so genanntes Leaf oder auf ein ganzes Verzeichnis. Tabelle 3-1 listet die Schlüsseltypen auf. Die Interpretation dieser Information wird durch den zweiten Wert, den Schlüsselnamen, festgelegt.

Drei Verzeichniseinträge (bzw. Schlüsselwerte) müssen in jedem Root_directory eines FireWire Geräts vorhanden sein:

(i) Module_Vendor_Id values

Dem Eintrag ist der Schlüsselwert 0x03 und Typ 0b00 (immediate) zugeordnet. Der Wert kann/sollte mit dem Node_Vendor_Id Wert aus dem Bus_Info_Block übereinstimmen.

(ii) Node_Capabilities value

Dem Eintrag ist der Schlüsselwert 0xC und Typ 0b00 (immediate) zugeordnet. Aufbau und Bedeutung der Inhalte sind in Abbildung 3-14 und Tabelle 3-2 aufgeführt. Die Einträge beziehen sich vornehmlich auf ISO/IEC 13213 spezifische Register (siehe S.34).

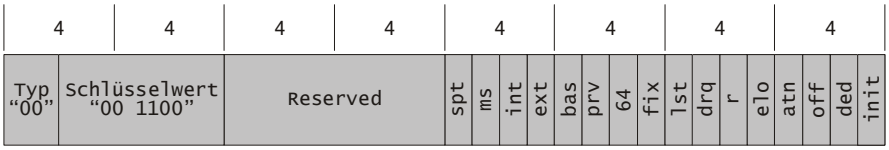

Abbildung 3-14: Node_Capabilities Format

Feld-name	Beschreibung
spt	SPLIT_TIME_OUT implementiert? [Autofokus = 1] (siehe S.37)
ms	Message Passing implementiert? [Autofokus=0]
int	Interrupt_target und Interrupt_mask implementiert? [Autofokus=0] siehe ISO/IEC 13213 spezifische Register auf Seite 34
ext	ARGUMENT Register implmentiert? [Autofokus=0]
bas	TEST_START & TEST_STATUS Register implementiert? [Autofokus=0]
prv	Implementiert die Node den "private space"[1]? [Autofokus=0]
64	Wird 64 Bit Adressierung benutzt? [Alle 1394 Geräte=1]
fix	Wird der fix Adressierungsmodus benutzt? [Alle 1394 Geräte=1]

Feld-name	Beschreibung
lst	Ist das Lost Bit im STATE_CLEAR & STATE_SET Register implementiert? [immer 1, da Lost Bit Required ist]
drq	Ist das Dreq Bit im STATE_CLEAR & STATE_SET Register implementiert? [immer 1, da Drq Bit Required ist]
r	Reserved
elo	Ist das Elog Bit implementiert? [Alle 1394 Geräte=0]
atn	Ist das Atn Bit implementiert? [Alle 1394 Geräte=0]
off	Ist das Off Bit implementiert? [Alle 1394 Geräte=0]
ded	Unterstützt die Node den „Dead" Status? [Autofokus=0]
init	Unterstützt die Node den initializing Status (siehe STATE Register)? [Autofokus=?]

Tabelle 3-2: Bit Interpretation Node_Capabilities Feld

(iii) Node_Unique_Id offset

Dem Eintrag ist der Schlüsselwert 0xD und Typ 0x2 (leaf) zugeordnet. Die 24 „freien" Bits sind ein Offset von der Adresse des Verzeichniseintrages. Die Adresse der Node_Unique_Id Datenfelder

berechnet sich aus „Adresse Verzeichniseintrag“ + („Offset Wert“ * 4). Siehe dazu: Abbildung 3-15. Die Einträge sind identisch mit den namesgleichen Einträgen des Bus_info_blocks (vgl. 3.3.3.2).

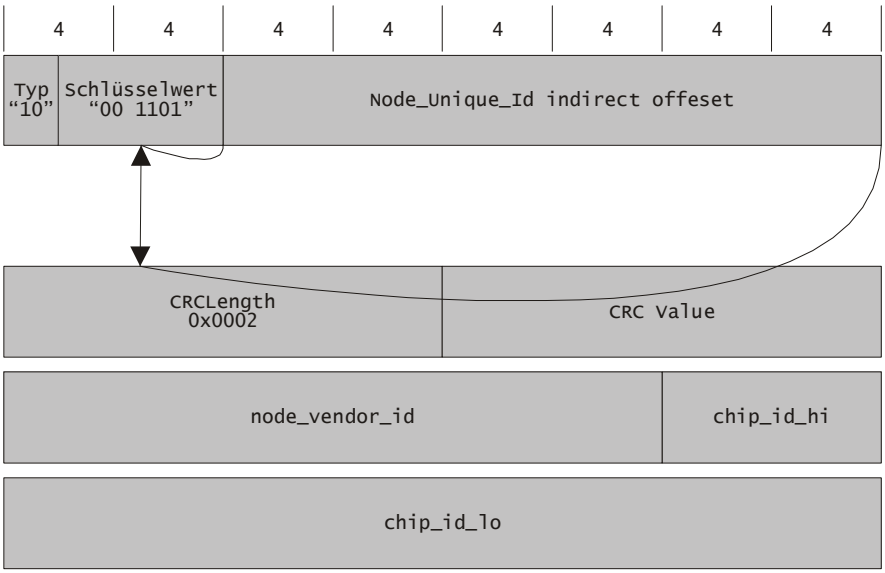

Abbildung 3-15: Format und Inhalt des Node_Unique_Id Eintrags und Felds

3.4 Funktionsweise des FireWire Autofokus

Die Arbeit des Autofokus lässt sich grob in vier Aufgaben einteilen:

Der erste Teil ist festzustellen, ob es sich bei einer gegebenen Id wirklich um eine Kamera handelt und ob diese Kamera im richtigen Modus betrieben wird. Der unterstützte Modus der Kamera soll die 640 x 480er Auflösung im YUV 4:2:2 Format sein.

Der zweite Teil ist das Empfangen der isochronen Pakete, da ein Bild aus dutzenden Paketen zusammengesetzt wird, muss dabei zuerst einmal das Startpaket gefunden werden.

Der dritte Teil ist die Analyse des Bilds und nicht Teil dieser Arbeit.

Der vierte Teil ist das Senden von Steuerbefehlen, beim Autofokus sind diese Befehle natürlich auf den Fokus der Kamera beschränkt.

3.4.1 Teilaufgabe 1: Kamera identifizieren

Nachdem dem Autofokus die Node-Id einer potentiellen Kamera gegeben wurde, ist der erste Schritt zu überprüfen, ob es sich tatsächlich um eine Kamera nach dem FireWire Kamerastandard ([2]) handelt. Der zweite Schritt ist die Lage der Kontrollregister herauszufinden, so dass im dritten Schritt die relevanten Betriebsdaten abgefragt werden können.

---

[1] Der „Private Space“ ist der 256Mbyte Speicherblock direkt vor dem „Register Space“. Siehe Abbildung 3-2.

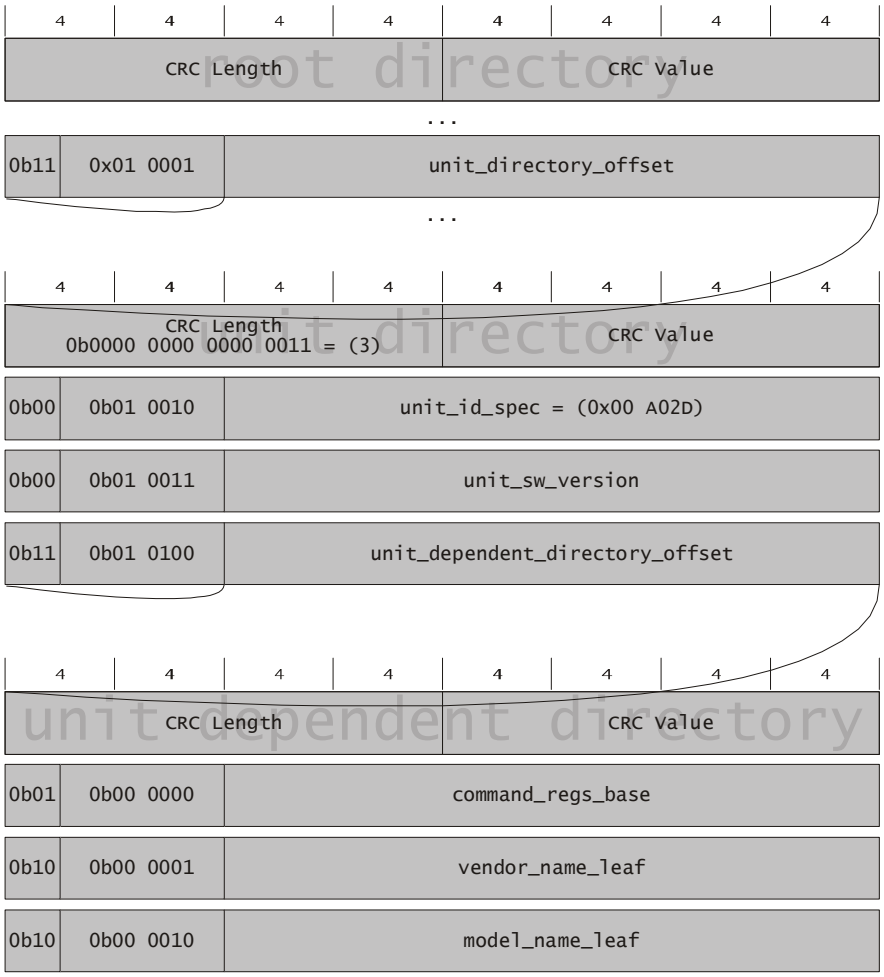

Abbildung 3-16: Verzeichnisorganisation bei FireWire Kameras

3.4.1.1 Identifizieren einer Kamera

Jede FireWire Kamera hat in ihrem „Root_directory" einen Eintrag mit dem Schlüsselwert/typ[1] 0xD1 – der „Unit_directory_offset". Der gefundene 24Bit Wert wird auf die Adresse des Verzeichniseintrags vierfach[2] aufgeschlagen (ähnlich dem Node_Unique_Id Leaf). Die gefundene Adresse ist die Basisadresse „Unit directory"s.

Nach der Kamera Spezifikation gibt es in diesem Verzeichnis drei Einträge, wobei der unit_id_spec Eintrag entscheidet, ob der Autofokus mit dem Gerät arbeiten kann. Wenn der Eintrag 0x00 A02D lautet, gehorcht das Gerät dem FireWire Kamera Standard 1.20 und die Node-Id kann als Ziel des Autofokus akzeptiert werden.

3.4.1.2 Adressraum der kameraspezifische Register

Alle Funktionen einer FireWire Kamera werden über ihre Kontroll- und Status Register gesteuert. Jede Kamera kann ihre Kontroll- und Status Register an eine beliebige Speicherstelle im Initial Unit

[1] Schlüsselwert/typ sei die Konkatenation der beiden Bitfelder interpretiert als 8 Bit Zahl.

[2] Alle Offsets werden in Quadlets gemessen, so dass eine Vervierfachung nötig ist.

Space einblenden. Das Auffinden des Bereichs sollte automatisch durch Auswertung des command_reg_base Eintrags geschehen. Dazu ist im „Unit directory“ (siehe 3.4.1.1) der „Unit dependent directory offset“ Eintrag zu finden, er ist durch einen Schlüsselwert/typ 0xD4 gekennzeichnet. Der Wert ist der Offset zum „unit dependent directory“. Dieses Verzeichnis enthält nun einen Eintrag mit Schlüsselwert/typ 0x40. Der Wert des Eintrags ist der Offset zur Basisadresse der Kamera Kontrollregister (siehe Abbildung 3-16).

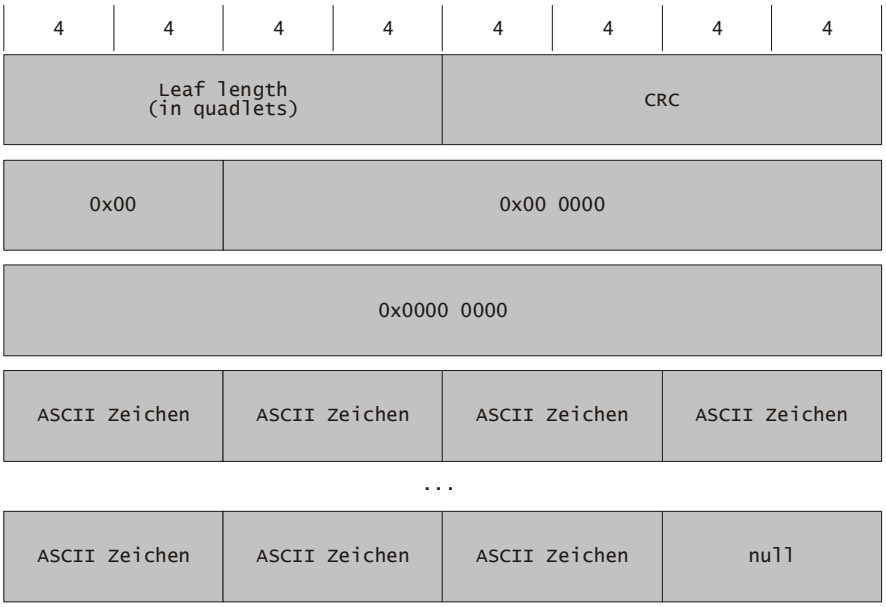

Abbildung 3-17: Vendor name/Model name Leaf

Die beiden anderen Einträge liefern den Hersteller und das Modell im Klartext:

(i) vendor_name_leaf

Schlüsselwert/typ 0x41. Der Eintrag ist ein Offset auf ein vendor name/model name leaf und enthält den Herstellernamen im Klartext (Abbildung 3-17).

(ii) model_name_leaf

Schlüsselwert/typ 0x42. Der Eintrag ist ein Offset auf ein vendor name/model name leaf und enthält den Modelnamen im Klartext (Abbildung 3-17).

3.4.1.3 Betriebsdaten abrufen

Die für den Autofokus wichtigen Betriebsdaten sind:

(i) Videomode, Videoformat und Kodierung

Die Einstellungen, die das Bildformat betreffen werden zweistufig gemacht, die erste Stufe – das Format – wählt eine Bildkategorie, die zweite Stufe – der Modus – wählt erst die Auflösung und Kodierung.

Zuerst muss nun überprüft werden, ob die gefundene Kamera im richtigen Format arbeitet, dazu ist das Cur_V_Format Register auszulesen. Tabelle 3-3 zeigt mögliche Formate, die DFW-VL 500 unterstützt ausschließlich Format_0.

Der nächste Schritt ist zu überprüfen, ob eine mit dem Autofokus kompatible Auflösung eingestellt ist. Dazu ist das Cur_V_Mode Register zu überprüfen, Tabelle 3-4, zeigt die Modi für den Fall, dass im Format_0 gearbeitet wird.

Der für den Autofokus gewählten Modus ist Format_0/Mode_3.

Video Format	Werte im Register
Format_0 (VGA non-compressed format)	0x0000 00000
Format_1 (Super VGA 1)	0x0000 00001
Format_2 (Super VGA 2)	0x0000 00002
Format_6 (Still Image Format)	0x0000 00006
Format_7 (Scalable Image Size Format)	0x0000 00007

Tabelle 3-3: Aktuelles Video Format (Cur_V_Format 0x608)

Video Mode	Wert im Register
0 (160x120) YUV (4:4:4)	0000 0000h
1 (320x240) YUV (4:2:2)	2000 0000h
2 (640x480) YUV (4:1:1)	4000 0000h
3 (640x480) YUV (4:2:2)	8000 0000h

Tabelle 3-4: Aktueller Video Modus bei Video Format 0(Cur_V_Mode 0x604)

(ii) Isochroner Kanal

Obwohl der FireWire Standard 64 isochrone Kanäle zur Verfügung stellt, können Kameras nur auf den ersten 16 Kanälen senden. Diese Einschränkung scheint willkürlich durch den FireWire Kamera Standard eingeführt worden zu sein.

Der eingestellte isochrone Kanal ist im ISO-Channel/ISO-Speed Register hinterlegt. Tabelle 3-5 zeigt die möglichen Werte des Registers; ‚N' ist der Kanal.

Übertragungsgeschwindigkeit	Wert im Register
0 (100 bps)	N000 0000h
1 (200 bps)	N100 0000h
2 (400 bps)	N200 0000h

Tabelle 3-5: Isochroner Kanal/Übertragungsgeschwindigkeit (ISO-Channel/Speed 0x60C)

3.4.2 Teilaufgabe 2: Bilddaten empfangen

Damit ein Bild in seiner Gesamtheit empfangen werden kann müssen zuerst alle Pakete die zum Bild gehören empfangen und dann zusammen gesetzt werden.

3.4.2.1 Isochrone Pakete empfangen

Wenn ein Kanal auf Port 1 empfangen werden soll, so muss im Isochron Port Register (18h) (Abschnitt 2.5.3.3) das Tag 1 und der IR Port 1 gesetzt werden, für Port 2 sind Tag 2 und IR Port 2 zu setzen.

Der Tag ist stets mit 0b0000 zu belegen, da die Kamera die isochronen Pakete mit diesem Tag markiert. IR Port ist mit dem aus dem ISO-Channel/Speed Register (Tabelle 3-5) der Kamera gelesenem Wert zu belegen.

Im letzten Schritt ist es nötig den Port an sich zu aktivieren, dazu ist das entsprechende IsoEn Bit (Siehe 2.5.3.1(viii) & (ix)) zu setzen.

3.4.2.2 Bilder empfangen

Da nun isochrone Pakete, der entsprechenden Kanäle 1 und/oder 2 im GRF empfangen werden, muss das empfangene Paket entweder Kanal 1 oder 2 zugeordnet werden. Das ChanNum Feld im Paket Kopf ermöglicht die Zuordnung. Das „sy“ Feld belegt mit 0b0001 im empfangenen isochronen Paket (siehe 2.5.4.10) markiert das erste Paket eines Bilds.

Da das Format und damit die Menge der in einem Paket übertragenen Pixel oder Bildzeilen konstant und bekannt sind, ist es durch Abzählen der Pakete ab dem Paket mit sy=0b0001 einfach möglich, die Position der übertragenen Bildpunkte festzustellen.

Im für den Autofokus gewählten Bildformat (Format_0/Mode_3 und 30fps) werden 2 Bildzeilen bzw. 1280 Pixel in 640 Quadlets übertragen.

Registerinhalt[0-2]	Bildrate	Zeilen/Paket	Pixel/Paket	Quadlets/Paket	Pakete/Bild
0b000 = 0	Nicht	unterstützt			
0b001 = 1	3,75 fps	¼	160	80	1920
0b010 = 2	7,5 fps	½	320	160	960
0b011 = 3	15 fps	1	640	320	480
0b100 = 4	30 fps	2	1280	640	240
0b101 = 5	Nicht	unterstützt			
0b110 = 6	Nicht	unterstützt			
0b111 = 7	Nicht	unterstützt			

Tabelle 3-6: Aktuelle Bildrate und Paketgrößen (Cur_V_Frm_Rate 0x600)

3.4.2.3 Bildformat

Das Format_0/Mode_3 hat eine Breite von 640 Bildpunkten und eine Höhe von 480 Bildpunkten bei 16 Bit/Bildpunkt. Die Kodierung ist eine YUV 4:2:2. Abbildung 3-18 zeigt ein in diesem Format empfangenes Paket und die Verteilung der Bytes auf die einzelnen Bildpunkte. Um diese YUV Helligkeit-Farbigkeit-Darstellung in eine gewöhnliche Farbigkeit-Darstellung wie RGB und vice versa umzurechnen können folgende Pseudofunktionen benutzt werden:

```
byte[] YUV2RGB (byte Y, U, V) {
 byte[3] rc;
 rc[0] = Y + 1.370 * (U - 128) // Rotkanal
 rc[1] = Y – 0.698 * (U - 128) – 0.336 * (V - 128) // Grünkanal
 rc[2] = Y + 1.730 * (V - 128) // Blaukanal
 return rc;
}
byte[] RGB2YUV (byte R, G, B) {
 byte[3] rc;
 rc[0] =  0.30 * R + 0.59 * G + 0.11 * B       // Luminanz
 rc[1] =  0.62 * R – 0.52 * G – 0.10 * B + 128 // Rot-Cyan Balance
 rc[2] = -0.15 * R – 0.29 * G + 0.44 * B + 128 // Gelb-Blau Balance
 return rc;
}
```

Die einzelnen Faktoren berücksichtigen die subjektive Farbwahrnehmung. Besonders bei der Berechnung der Luminanz ist deutlich zu erkennen, dass vor allem der grüne Farbton, gefolgt vom roten, großen Anteil an der wahrgenommenen Helligkeit hat.

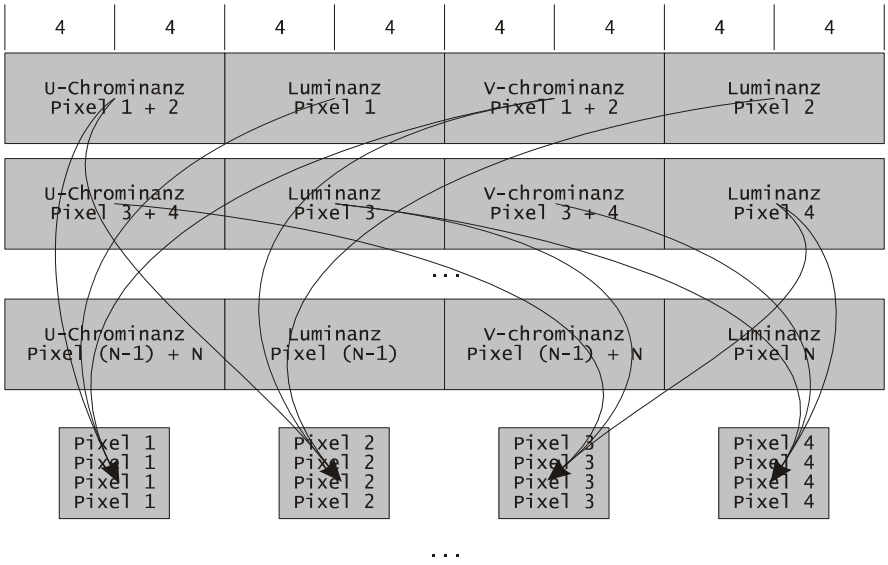

Abbildung 3-18: Struktur der Nutzlast eines isochronen Pakets mit YUV 4:2:2 Kodierung

3.4.3 Teilaufgabe 4: Steuerdaten senden

Der Fokus wird durch Schreiben auf das Focus CSR eingestellt. Die Basisadresse für dieses Register ist der Verzeichniseintrag „command_reg_base“ wie schon in Kapitel 3.4.1 beschrieben.

Das hier beschriebene Focus CSR ist das spezifische Focus CSR der Sony Kamera, Abbildung 3-19 zeigt die Struktur des Registers.

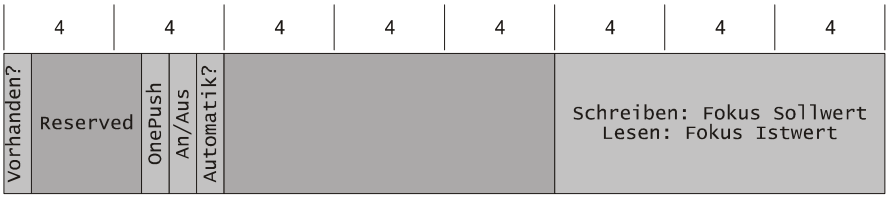

Abbildung 3-19: Fokus CSR (0x828)

Das Registerformat wird auch von anderen Features benutzt, so dass einige Felder nicht sinnvoll benutzt werden können.

Das „Vorhanden?“ Bit ist immer gesetzt und sollte auch bei jedem Schreibvorgang gesetzt werden, eine „1“ im Feld zeigt an, dass das Feature vorhanden ist – das ist bei der Sony Kamera natürlich der Fall.

„One Push“ veranlasst die Kamera den Fokus nur einmal und nicht beständig scharf zu stellen, unter der Voraussetzung, dass überhaupt ein Autofokus vorhanden ist. Für die Sony Kamera und den Autofokus ist dieses Feld irrelevant.

„An/Aus“ schaltet das Feature an und ab, ist für den Fokus nicht sinnvoll einsetzbar, sollte aber dennoch mit jedem Schreiben gesetzt werden.

Der numerische Wert des Fokus reicht von 000h (nah) bis 1BFh (fern) ist.[1] Der Inhalt des Register ist immer der Ist-Wert, dieser Ist-Wert nähert sich dem geschriebenen Soll-Wert mit Verzug (Mechanik).

[1] Der Wertebereich ist Sony spezifisch.

4 Ausblick

Mit den in Kapitel 3 präsentierten Ergebnissen wird es möglich sein ein Gesamtsystem, wie in Abschnitt 3.1 skizziert, in Betrieb zu nehmen und eine autonome Regelung des Autofokus zu implementieren.

Die praktische Umsetzung war allerdings bis zum jetzigen Zeitpunkt nicht möglich, da sich die Hardwarerealisierung der FireWire Tochterplatine des Nios Boards verzögert hat.

5 Anhang

5.1 Definitionen

5.1.1 CRC Berechnung[1]

```
#define MSB32 ((unsigned)1<<31)
#define ONES32 0XFFFFFFFF
#define CRC_COMPUTE ((Quadlet)0X04C11DB7)
typedef unsigned long Quadlet // “long” ist ein 32-bit Wert
// Erzeugt den CRC Wert für ein Paket mit “sizeInQuads” Länge
Quadlet GenerateCrc (Quadlet * inputs, int sizeInQuads)
{
  Quadlet crcSum;
  assert(sizeInQuads >= 1); // Größe des Pakets mit CRC
  crcSum = CalculateCrc(inputs, sizeInQuads - 1);  // CRC nur auf den
       Daten berechen
  return (~crcSum);
}
// Überprüft CRC eines Pakets mit “size” Quadlet Nutzlast
int ValidateCrc (Quadlet * inputs, int sizeInQuads)
{
  Quadlet crcSum;
  assert(sizeInQuads >= 1); // Größe des Pakets mit CRC
  crcSum = CalculateCrc(inputs, sizeInQuads);
  // compute CRC on the data *and* the received CRC
  return (crcSum != CRC_RESULTS);
}
// The GenerateCrc() function points to protected values,
// it checks these values and return a final 32-bit result
Quadlet CalculateCrc (Quadlet * inputs, int sizeInQuads)
{
  Quadlet inQuad, crcSum, newMask;
  int i, oldBit, newBit, sumBit;
  // Der Wert crcSum wird mit Einsen initialisiert
  crcSum = (Quadlet) 0XFFFFFFFF;
  // Jedes Quadlet wird einzeln behandelt
  for (i = 0; i < sizeInQuads; i += 1)
  {
    inQuad = inputs[i];
```

[1] Sinngemäß übernommen aus [9]

```
    // Bearbeite jedes Bit des Eingabe Quadlets
    for (newMask = MSB32; newMask != 0; newMask >>= 1)
    {
      newBit = ((inQuad & newMask) != 0); // Nächstes Bit
      oldBit = ((crcSum & MSB32) != 0); // MSB und crcSum verUNDen
      sumBit = oldBit ^ newBit; // Die Summe ist das EXOR
      crcSum = ((crcSum << 1) & ONES32) ^ (sumBit ? CRC_COMPUTE : 0);
    }
  }
  return (crcSum);
}
```

5.1.2 Konstanten

(i) Tsb12.h[1]

Die Definitionen wurden alphabetisch sortiert um eine einfachere Handhabung in der Papierversion zu erreichen. Konstanten im Pseudocode können so leicht nachgeschaut werden.

```
// Adresskonstanten
#define a_ATFBurstWrite       0xA0
#define a_ATFContinue         0x84
#define a_ATFFirst            0x80
#define a_ATFStatus           0x30
#define a_ATFUpdate           0x8C
#define a_Control             0x08
#define a_CycleTimer          0x14
#define a_Diagnostigs         0x20
#define a_FIFOControl         0x1C
#define a_GRFData             0xC0
#define a_GRFStatus           0x3C
#define a_HostControl         0x40
#define a_Interrupt           0x0C
#define a_InterruptMask       0x10
#define a_IsochPortNumber     0x18
#define a_ITFBurstWrite       0xB0
#define a_ITFContinue         0x94
#define a_ITFFirst            0x90
#define a_ITFStatus           0x34
#define a_ITFUpdate           0x9C
#define a_MuxControl          0x44
#define a_NodeAddress         0x04
#define a_PHYChipAccess       0x24
#define a_Version             0x00
```

[1] Übernommen aus dem dem C Prototypen von Andreas Mäder

5.2 Quellenverzeichnis

[1] Anderson, Don (1998):

FireWire System Architecture: IEEE1394a 2nd Edition

MindShare, Inc.

ISBN 0-201-48535-4

[2] Camera Working Group of the 1394 Trade Association (1998):

1394-based Digital Camera Specification (Version 1.20)

1394 Trade Association

http://www.1394ta.org/Technology/Specifications/specifications.htm

[3] Texas Instruments (2002):

TSB12LV01B IEEE 1394-1995 High Speed Serial-Bus Link-Layer Controller Data Manual.

http://www-s.ti.com/sc/ds/tsb12lv01b.pdf

[4] Texas Instruments (2003):

TSB41AB3 IEEE 1394a-2000 Three-Port Cable Transceiver/Arbiter

http://www-s.ti.com/sc/ds/tsb41ab3.pdf

[5] Sony (keine Angabe):

DFW-V500/DFW-VL500 Data Manual (Ver.1.0 English)

http://www.sony.net/Products/ISP/pdf/guide/GDFWV500E.pdf

[6] Texas Instruments (2001):

TSB12LV01B/TSB41AB3 Reference Schematic

http://focus.ti.com/lit/an/slla090a/slla090a.pdf

[7] IEEE (2001):

IEEE Standard for a Control and Status Registers (CSR) Architecture for Microcomputer Buses 2002

ISBN 0-7381-3100-8

http://shop.ieee.org/store/product.asp?prodno=SS94977

[8] IEEE (1994):

Information technology - Microprocessor systems - Control and Status Registers (CSR) Architecture for microcomputer buses 1994

ISBN 1-5593-7448-9

http://shop.ieee.org/store/product.asp?prodno=SH94220

[9] IEEE (1995):

P1394 - Standard for a High Performance Serial Bus Final Draft 22.Nov.1995

Nicht mehr erhältlich.