Rainer Bermbach, Jens Brocke

IEEE-1394-Software an der Schnittstelle zur Hardware

Erfahrungen bei der Portierung eines Embedded Hardware Abstraction Layers für IEEE 1394

Bei der Entwicklung von Embedded IEEE-1394-Anwendungen kommt man mangels einheitlicher Ansteuerung der verwendeten IEEE-1394-ICs um eine hardwarespezifische Anpassung der Protokollsoftware kaum herum. Diese Aufgabe übernimmt meist ein Hardware Abstraction Layer (HAL). In diesem Artikel werden am Beispiel eines HAL für den MPEG2Lynx Link Layer Controller von Texas Instruments Funktion und Aufbau einer HAL-Software erläutert. Außerdem wird die Portierung des MPEG2Lynx-HAL auf eine Embedded IEEE-1394-Plattform mit einem C167-Mikrocontroller beschrieben.

Das serielle Bussystem IEEE 1394, auch bekannt unter den Warenzeichen „FireWire™“ (Apple) und „i.LINK™“ (Sony), ist der aussichtsreichste Kandidat für die Integration von Computertechnik, Unterhaltungselektronik und Multimedia auf einem Bus. Für diese Anwendungen bietet sich IEEE 1394 durch Datenraten von derzeit bis zu 400 MBit/s, Dienste für asynchrone wie isochrone Übertragung, automatische Buskonfiguration mit „Hot Plugging“ und Punkt-zu-Punkt-Übertragung zwischen beliebigen Geräten geradezu an. Aber auch für industrielle Anwendungen wird IEEE 1394 zunehmend in Betracht gezogen, wie die „Automotive Working Group“ und die „Instrumentation and Industrial Working Group“ innerhalb der 1394 Trade Association zeigen.

Während für PC-Anwendungen durch den OHCI-Standard (Open Host Controller Interface) inzwischen eine einheitliche Schnittstelle zur Anbindung von IEEE 1394 an den PCI-Bus zur Verfügung steht, hat man bei Embedded Anwendungen ein grundsätzliches Problem. Es gilt, IEEE-1394-ICs verschiedener Hersteller mit IEEE-1394-Protokollsoftware anderer Hersteller zu verbinden. Dazu wird meist ein Hardware Abstraction Layer (HAL) als Softwareschicht zwischen dem Protokollstack und den ICs eingefügt, der die allgemeinen Funktionen des Stacks an die interne Struktur und Ansteuerung der IEEE-1394-Hardware anpaßt. Falls der Hersteller des Stacks keinen HAL für die gewünschte Hardware anbietet, muß man diesen HAL selbst entwickeln oder zumindest den HAL für eine vergleichbare Hardware überarbeiten.

Im vorliegenden Projekt soll ein IEEE-1394-Protokollstack eines Drittherstellers auf eine Embedded IEEE-1394-Plattform mit einem C167-Mikocontroller von Infineon und einem MPEG2Lynx Link Layer Controller (LLC) von Texas Instruments (TI) portiert werden. Da der Stackhersteller keinen Hardware Abstraction Layer für diesen LLC anbietet, ist eine Referenz bei der notwendigen Anpassung des mitgelieferten Beispiel-HAL an den MPEG2Lynx zumindest hilfreich. Hier bietet sich der HAL von Texas Instruments an, den TI den Kunden ihrer Embedded LLCs auf Anfrage kostenlos als C-Quellcode zur Verfügung stellt; allerdings übernimmt TI keine Garantie für die Funktion der Software und leistet auch keinen Support.

1 Funktion des HAL in einem IEEE-1394-Knoten

Anhand eines Blicks auf das Schichtenmodell des IEEE-1394-Standards in Bild 1 lassen sich die Aufgaben eines Hardware Abstraction Layers am besten erläutern [1]. Die unterste Schicht bildet der Physical Layer, der vor allem für die elektrische Codierung und Übertragung der Daten auf dem Bus zuständig ist. Darüber liegt der Link Layer, der vorwiegend die Pakete samt CRC zusammenstellt bzw. beim Empfang entschlüsselt und prüft. Physical Layer und Link Layer werden zur Zeit meist durch zwei getrennte ICs realisiert, den Physical Layer Controller (PHY) und den Link Layer Controller (LLC), ein Trend zum integrierten PHY/LLC-IC ist jedoch bereits absehbar. Das restliche IEEE-1394-Protokoll ist typischerweise in der Software realisiert. Der

Transaction Layer ist dabei für die asynchrone Übertragung zuständig (Read, Write und Lock Transactions). Er überwacht den Empfang der durch ihn gesendeten Request und Response Packets, wiederholt die Übertragung nicht bestätigter Pakete und erkennt auftretende Fehlerzustände (Response Timeout, Response Error, Retry Limit erreicht usw.). Das Serial Bus Management stellt keinen Layer im eigentlichen Sinn dar, weil es zu Konfigurations- und Kontrollzwecken mit allen anderen Layern kommuniziert. Zusätzlich realisiert es bei manchen Knoten die globalen Managementaufgaben des IEEE-1394-Busses (Bus Manager, Isochronous Resource Manager und Cycle Master). Über der IEEE-1394-Hard- und -Software liegt schließlich die Applikation, die über den IEEE-1394-Bus kommunizieren soll. Der Hardware Abstraction Layer greift in sämtliche Datenpfade ein, die zwischen der Hardware (PHY, LLC) und den Softwarekomponenten verlaufen. Er ist im IEEE-1394-Standard nicht definiert und daher in Bild 1 nur angedeutet.

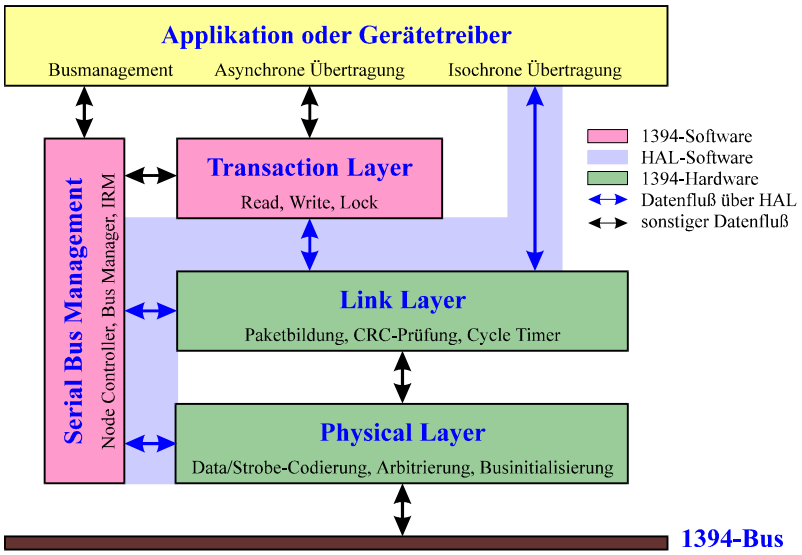

Bild 1: Schnittstellen des Hardware Abstraction Layers im IEEE-1394-Schichtenmodell

Für jeden der durch den Hardware Abstraction Layer beeinflußten Datenpfade definiert der IEEE-1394-Standard eine Reihe von allgemeinen Diensten [2]. Die reale Implementierung dieser Dienste hängt von der internen Struktur des Link Layer Controllers und dessen Ansteuerung über seinen Registersatz ab, die bei jedem Embedded LLC unterschiedlich sind. Dabei ist der Pfad zwischen Bus Management und Physical Layer nur logisch vorhanden, praktisch kann der Physical Layer Controller nur über den LLC mit der Software kommunizieren, da nur der LLC direkt vom Mikrocontroller gesteuert wird. Ein Dienst, der mit einem Datentransfer zum LLC verbunden ist, wird vom HAL in die erforderlichen Schreibzugriffe auf LLC-Register umgesetzt. Dienste zum Datentransfer an die Software werden vom HAL durch Auslesen der passenden LLC-Register und Übertragung der Daten an die vorgesehene Softwarekomponente realisiert. Letztere Dienste können durch die Software angefordert oder aufgrund von Busereignissen automatisch vom LLC geleistet werden. Busereignisse signalisiert der LLC dem HAL dabei über Interrupts, das auslösende Ereignis identifiziert der HAL über die Auswertung der Interruptregister des LLC.

Einer der Dienste zwischen Bus Management und Physical Layer ist z. B. das Auslösen eines Bus Resets. Hierzu greift der Hardware Abstraction Layer über das PHY-Zugriffsregister des Link Layer Controllers auf den Physical Layer Controller zu. Umgekehrt liest der HAL über den gleichen Mechanismus die Physical ID des Knotens aus dem Registersatz des PHY aus, wenn diese dem Bus Management mitgeteilt werden soll. Die Initialisierung des Link Layers für die Übertragung ist ein typischer Dienst zwischen Bus Management und Link Layer. Der Hardware Abstraction Layer löscht dabei alle in den FIFOs des Link Layer Controllers vorhandenen Daten und aktiviert die Sende- und Empfangslogik. Je nach LLC können zusätzlich verschiedene Eigenschaften konfiguriert werden, z. B. die Größe sowie die Verwendung der FIFOs, der Mechanismus für die Datenübertragung vom/zum LLC (µC-Schnittstelle, DMA, spezielle Schnittstellen), das Retry-Protokoll und die Paket-Header für die automatische Paketbildung. In umgekehrter Richtung signalisiert der Link Layer dem Bus

Management über den HAL verschiedene Fehlerzustände (Header CRC Error, Transaction Code Error usw.). Die meisten Dienste zwischen Transaction Layer und Link Layer umfassen die Übertragung asynchroner Pakete in beide Richtungen. Beim Transfer zum Link Layer setzt der HAL die Paketdaten in das vom Link Layer Controller geforderte Format um, schreibt die Daten in das passende FIFO des LLC und startet die Übertragung. Der Start erfolgt dabei je nach LLC entweder explizit über ein spezielles Register oder implizit durch das Beschreiben des FIFOs über verschiedene Register (First, Continue und Update Register). Empfangene Pakete werden vom HAL aus den FIFOs ausgelesen und in der vorgesehenen Form an den Transaction Layer übergeben. Die isochrone Datenübertragung findet hingegen direkt zwischen Link Layer und Applikation statt. Einer der Dienste seitens der Applikation ist z. B. die Auswahl der Empfangskanäle, die der HAL in die entsprechende Konfiguration der isochronen Empfangsfilter-Register des LLC umsetzt. Umgekehrt signalisiert der Link Layer den Cycle Synch Event (nomineller Beginn eines isochronen Zyklus) über den HAL an die Applikation. Werden die isochronen Daten samt Header vom/zum LLC übertragen, nimmt der HAL auch hier eine Anpassung des Datenformats zwischen der Applikation und den FIFOs im LLC vor.

2 Aufbau des MPEG2Lynx-HAL

Als Beispiel für die Implementierung eines Hardware Abstraction Layers dient in diesem Artikel der HAL von Texas Instruments für den MPEG2Lynx Link Layer Controller von TI. Da die Grundstruktur der verschiedenen Embedded Link Layer Controller recht ähnlich ist (µC-Schnittstelle, Registersatz, Sende- und Empfangs-FIFOs, Interruptsignal), ist der Aufbau des MPEG2Lynx-HAL aber auch auf andere LLCs übertragbar, die Ansteuerung der LLCs über deren Registersätze wäre jedoch unterschiedlich.

Der MPEG2Lynx-HAL besteht aus mehreren Funktionsgruppen, die untereinander sowie nach unten mit dem MPEG2Lynx Link Layer Controller und nach oben mit der IEEE-1394-Protokollsoftware kommunizieren (Bild 2). Der Hardware Abstraction Layer ist dabei seitens Texas Instruments für die Interaktion mit ihrem Embedded LynxSoft API (Application Programmer‘s Interface) und Bus Management vorgesehen [3]. Da weder diese Software noch ihre Dokumentation frei verfügbar sind, wird hier nur die Funktionalität erläutert, die aus der Interaktion mit dem HAL ersichtlich ist.

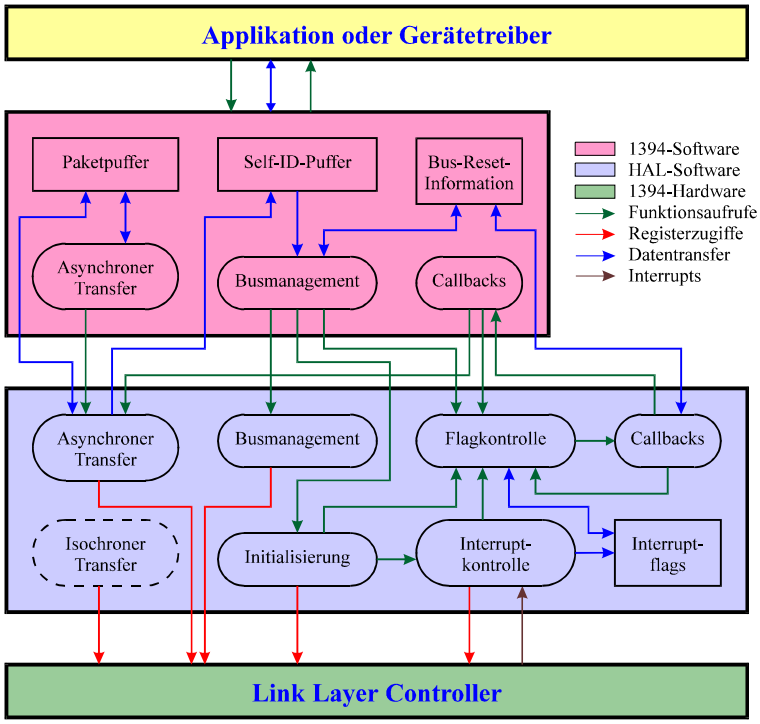

Bild 2: Interaktion des MPEG2Lynx-HAL mit IEEE-1394-Hard- und -Software

Die Protokollsoftware kann über Funktionsaufrufe im Hardware Abstraction Layer sowohl Daten an den Link Layer Controller übertragen als auch von diesem lesen. Der Datentransfer erfolgt dabei je nach Datenmenge über Funktionswerte und -parameter oder über Datenpuffer, auf die Protokollstack und Hardware Abstraction Layer über Zeiger zugreifen. Busereignisse meldet der Link Layer Controller über seinen Interrupt. Das auslösende Ereignis wird innerhalb der Interrupt Service Routine (ISR) des HAL ausgewertet und das entsprechende Interruptflag des HAL gesetzt. Die Flags werden ihrerseits durch die Flagkontrolle ausgewertet und die entsprechenden Callback-Funktionen des HAL aufgerufen. Callbacks sind spezielle Funktionen, die beim Eintreten von Ereignissen aufgerufen werden, um das Ereignis zu quittieren oder weiterzuverarbeiten, ihr Aufruf erfolgt oft über Zeiger. Die HAL-Callbacks bearbeiten die LLC-spezifischen Aspekte der aufgetretenen Ereignisse und rufen danach die jeweiligen Callbacks der Protokollsoftware auf, die für die Weiterverarbeitung der Ereignisse innerhalb des Protokollstacks sorgen. Die Flagkontrolle ist über eine Funktionswarteschlange implementiert, damit der HAL auch auf Plattformen ohne Betriebssystem funktioniert. Die Protokollsoftware muß daher in geeigneten Abständen Funktionen aus der Schlange aufrufen, um auf Busereignisse reagieren zu können.

Die genaue Interaktion zwischen Protokollsoftware, Hardware Abstraction Layer und Link Layer Controller wird im folgenden anhand typischer Vorgänge erläutert [4]. Die Zuordnung der dabei aufgeführten HAL-Funktionen zu den Funktionsgruppen aus Bild 2 ist in Tabelle 1 angegeben Die isochronen Funktionen des HAL werden nicht beschrieben, da sie teilweise nicht implementiert sind und Zusatzhardware voraussetzen (Datentransfer über Bulky Data Interface des MPEG2Lynx).

Funktion	Aufgaben
Initialisierung	
LynxHALInit	Initialisiert HAL-Software und MPEG2Lynx-LLC
BoardInit	Konfiguriert Schnittstelle zwischen Mikrocontroller und LLC
Flagkontrolle	
InitCallQ	Initialisiert Funktionswarteschlange
LynxHALCheckInterruptFlags	Prüft Interruptflags, ruft ggf. Callbacks auf
PutCallInQ	Stellt Funktion in die Warteschlange
YieldToCallQ	Ruft Funktion aus der Warteschlange auf
CauseBusEventIndication	Setzt Empfangs-Flag erneut, falls mehrere Pakete im FIFO sind
Interruptkontrolle	
LynxISRInitNodeISR	Initialisiert Interrupt Service Routine
LynxISREnableInterrupts	Gibt benötigte MPEG2Lynx-Interrupts frei
LynxHALNodeISR	Interrupt Service Routine, setzt Interruptflags gemäß Ereignissen
Callbacks	
LynxHALBusResetCallback	Prüft auf Self-ID-Fehler, aktualisiert Bus-Reset-Information
LynxHALIndicationCallback	Signalisiert Empfang von asynchronen Paketen
LynxHALIoCompleteCallback	Meldet empfangenen ACK-Code an Protokollsoftware
Asynchroner Transfer	
LynxHALSaveSelfIDPacket	Speichert Self-ID-Pakete im Self-ID-Puffer
LynxHALSaveAsyncPacket	Speichert empfangene Pakete im Paketpuffer
LynxHALSendAsyncPacket	Überträgt zu sendende Pakete zum MPEG2Lynx und sendet
Busmanagement	
LynxHALCauseBusReset	Erzeugt Reset auf dem IEEE-1394-Bus
LynxHALGetThisNodesID	Ermittelt das Self-ID-Paket des lokalen Knotens

Tabelle 1: Aufgaben einiger HAL-Funktionen

- **Initialisierung:** Die Protokollsoftware legt die Struktur für die Bus-Reset-Information an und speichert dort die Zeiger zu ihren Callback-Funktionen, die vom Hardware Abstraction Layer zur Signalisierung von Ereignissen benötigt werden. Danach initialisiert sie HAL-Software und MPEG2Lynx (LynxHALInit). Dies schließt die Initialisierung der Mikrocontroller-Schnittstelle (BoardInit), der Warteschlange (InitCallQ) und der Interrupt Service Routine (LynxISRInitNodeISR) sowie die Freigabe der LLC-Interrupts (LynxISREnableInterrupts) ein. Nun löst der Protokollstack einen Bus Reset aus (LynxHALCauseBusReset), um die aktuelle Struktur des IEEE-1394-Busses zu ermitteln.

- **Bus Reset:** Der Reset erzeugt einen Interrupt des Link Layer Controllers, durch den die Interrupt Service Routine (LynxHALNodeISR) des Hardware Abstraction Layers aufgerufen wird. Die ISR wertet die Interruptquelle aus, setzt das Reset-Flag des HAL und stellt die Flagkontrollfunktion (LynxHALCheckInterruptFlags) in die Warteschlange (PutCallInQ). Beim Aufruf der Kontrollfunktion aus der Warteschlange durch die Protokollsoftware (YieldToCallQ) prüft die Kontrollfunktion das Reset-Flag, setzt es zurück und ruft die Reset-Callback des HAL auf (LynxHALBusResetCallback). Die Callback wartet

auf das Ende der Self-ID-Phase des IEEE-1394-Busses. Bei zu langer Wartezeit bricht die Callback ab, um den Protokollstack nicht zu blockieren. Sie stellt sich dabei selbst erneut in die Warteschlange und wird daher später wieder aufgerufen. Nach der Self-ID-Phase aktualisiert die Callback die Bus-Reset-Information (Self-ID-Fehler, Reset-Zähler, lokale Knotennummer, Anzahl der Knoten, Knotennummer des Isochronous Resource Managers usw.) und ruft die Reset-Callback der Protokollsoftware auf. Diese reserviert Speicher für den Self-ID-Puffer und sichert die empfangenen Self-ID-Pakete im Puffer (LynxHALSaveSelfIDPacket). Aus diesen Daten und dem eigenen Self-ID-Paket, das der HAL aus den Registern des lokalen Physical Layer Controllers ermittelt (LynxHALGetThisNodesID), kann der Protokollstack bei Bedarf die Topology Map und die Speed Map generieren.

- **Asynchroner Empfang:** Der Ablauf gleicht dem beim Bus Reset, allerdings wird hier das Empfangs-Flag gesetzt und geprüft sowie die Empfangs-Callback des Hardware Abstraction Layers aufgerufen (LynxHALIndicationCallback). Diese ruft ihrerseits die Empfangs-Callback des Protokollstacks auf, die einen Paketpuffer reserviert und das empfangene Paket sichert (LynxHALSaveAsyncPacket). Sollten mehrere Pakete im Empfangs-FIFO sein, setzt die Protokollsoftware einen neuen Empfangszyklus in Gang (CauseBusEventIndication). Das gesicherte Paket wird von den asynchronen Funktionen des Protokollstacks ausgewertet.

- **Asynchrones Senden:** Die Protokollsoftware reserviert einen Paketpuffer, füllt ihn mit den Paketdaten und beauftragt den Hardware Abstraction Layer mit dem Versenden des Pakets (LynxHALSendAsyncPacket). Das für das gesendete Paket empfangene Acknowledge Packet (ACK) wird über Interrupt Service Routine und ACK-Flag durch die ACK-Callback des HAL (LynxHALIoCompleteCallback) ausgewertet und der ACK-Code per Zeiger an den Protokollstack übergeben. Danach wird die ACK-Callback der Protokollsoftware zur weiteren Verarbeitung des ACK-Codes aufgerufen.

3 Portierung des HAL auf C167-Plattform

Der Hardware Abstraction Layer von Texas Instruments für den MPEG2Lynx Link Layer Controller von TI soll im vorliegenden Projekt als Referenz für die Anpassung des zugekauften IEEE-1394-Protokollstacks an den MPEG2Lynx dienen. Weil die von TI erhaltene Version des MPEG2Lynx-HAL auf eine DSP-Plattform (TMS320Cxx von TI) zugeschnitten ist, muß zu diesem Zweck eine Anpassung an den C167-Mikrocontroller erfolgen, der auf der im Projekt eingesetzten Embedded IEEE-1394-Plattform verwendet wird. Als Compiler dient die µVision2-Programmierumgebung von Keil, die Testläufe des portierten MPEG2Lynx-HAL direkt auf der Zielhardware ermöglicht. Die Kontrolle der Embedded IEEE-1394-Plattform erfolgt dabei über das Monitorprogramm des Keil-Debuggers. Zusätzlich überträgt die HAL-Testapplikation Statusinformationen in Textform seriell an den Debugger, die dieser im seriellen Ausgabefenster darstellt.

Der C-Quellcode des MPEG2Lynx-HAL ist in mehrere Softwaremodule mit unterschiedlichen Aufgaben unterteilt (Tabelle 2). Der eigentliche Hardware Abstraction Layer besteht aus den vier Modulen „LynxHAL“, „LynxISR“, „LynxReg“ und „LynxStr“. Das Modul „BIOS“ ist für die hardwarespezifischen Anpassungen an die jeweilige Mikrocontroller-Plattform zuständig. Das Modul „CallQ“ verwaltet die Funktionswarteschlange (s. Kap. 2). Es wird nur in Umgebungen ohne Betriebssystem benötigt, in denen die Flagkontrolle nicht über eine separate Task erfolgen kann.

Modul	Aufgaben
LynxHAL	HAL-Initialisierung, Callbacks, Asynchroner Transfer, Busmanagement
LynxISR	Interrupt Service Routine, MPEG2Lynx-Interruptfreigabe, Interruptflag-Kontrolle
LynxReg	Lesen und Schreiben von MPEG2Lynx-Registern durch Mikrocontroller
LynxStr	Isochroner Transfer und asynchroner Stream-Transfer, nur teilweise implementiert
BIOS	µC-Startup-Code, Schnittstellen- und Interruptinitialisierung, µC-Interruptfreigabe
CallQ	Funktionen in Warteschlage stellen und aufrufen

Tabelle 2: Softwaremodule des MPEG2Lynx-HAL

Bei der Portierung des MPEG2Lynx-HAL auf die verwendete IEEE-1394-Plattform wurden sowohl Änderungen aufgrund des C167-Mikrocontrollers als auch wegen der verwendeten µVision2-Programmierumgebung durchgeführt. Dies betraf vor allem die hardwarenahen Module „BIOS“ und „LynxReg“, die direkt auf Mikrocontroller und Link Layer Controller zugreifen. Im Modul „BIOS“ konnte der Startup-Code entfallen, da dieser vom Keil-Compiler erzeugt wird [5]. Gleiches gilt für die Initialisierung der Vektoradresse der Interrupt Service Routine (ISR), die der Compiler durch die Deklaration der ISR mit dem Zusatz „interrupt 0x19 using Reg_Bank_ILVL1“ automatisch setzt. Außerdem verwendet der Compiler dadurch das Register Bank Switching des C167, um nicht alle Register einzeln sichern zu müssen. Die Low-Level-Funktionen zur seriellen Kommunikation konnten ebenfalls entfallen, da der Keil-Compiler entsprechende Bibliotheken anbietet. Die Makros zur Maskierung der Mikrocontroller-Interrupts wurden an den C167 angepaßt. Im Modul „LynxReg“ wurden die LLC-Registerzugriffe auf 32-Bit-Zeiger umgestellt, da die Basisadresse des LLC außerhalb des Programmspeichers liegt (Speichermodell „SMALL“: 64 KByte Code, Speichertyp „near“ für Variablen). Dabei wurden Zeiger des Speichertyps „huge“ anstatt des standardisierten „far“ verwendet, weil die Adreßberechnung im C167 über diesen Typ schneller abläuft. Außerdem erhielten Variablen, die durch die ISR beeinflußt werden, in allen Modulen das Attribut „volatile“ (flüchtig), damit das Programm immer direkt auf die Speicherwerte und nicht auf evtl. Kopien in CPU-Registern zugreift.

Die wichtigste Änderung im Modul „BIOS“ betraf die Initialisierung der Schnittstelle zwischen dem C167-Mikrocontroller und dem MPEG2Lynx Link Layer Controller. Weil der MPEG2Lynx nur Betriebsarten für TI-DSPs sowie 68xxx- und 8051-kompatible Controller aufweist, emuliert die IEEE-1394-Plattform die DSP-Schnittstelle [6]. Der Busmodus und das Timing werden per Software im C167 eingestellt, das RD/WR#-Signal des LLC per Hardware aus RD# und WR# des C167 generiert [7]. Im DSP-Modus erzeugt der LLC aber standardmäßig kein Ready-Signal (sog. Blind Access Mode), das der C167 für einen optimalen Buszugriff benötigt. Das Ready-Signal muß daher erst aktiviert werden. Zusätzlich ist eine Umkehrung seiner Polarität erforderlich, da der RDY-Pin des LLC auf der IEEE-1394-Plattform mit dem READY#-Pin des C167 verbunden ist. Die Polarität des LLC-Interrupts sowie die Treiberart für Ready und Interrupt (Push/Pull oder Tristate) können ebenfalls eingestellt werden. Diese Konfigurationen steuert das I/O-Kontrollregister des MPEG2Lynx. Der erste Zugriff auf das Register muß noch mit vielen Waitstates erfolgen, erst danach kann der C167 auf Ready-Betrieb umschalten. Anschließend wird der vom LLC verwendete Interrupt im C167 konfiguriert und freigegeben.

Die LLC-Registerzugriffe im Modul „LynxReg“ sind von der Busbreite und der Speicherkonvention der verwendeten ICs abhängig und mußten somit an den C167-Mikrocontroller angepaßt werden. Weil die LLC-Register 32 Bit (ein Quadlet) breit sind, der Bus zwischen C167 und MPEG2Lynx Link Layer Controller aber nur 16 Bit pro Zyklus transportiert, werden die Register in zwei 16-Bit-Werten (Doublets) gelesen oder geschrieben. Der MPEG2Lynx speichert die Quadlets dabei IEEE-1394-typisch gemäß Big Endian, der C167-Mikrocontroller hingegen gemäß Little Endian. Aus diesem Grund müssen Upper Doublet und Lower Doublet bei der Adressierung jeweils vertauscht werden, damit beide ICs die Daten gleich interpretieren. Das Upper Doublet wird dabei ab Offset 0 zur LLC-Registeradresse gelesen oder geschrieben, das Lower Doublet ab Offset 2 (Bild 3).

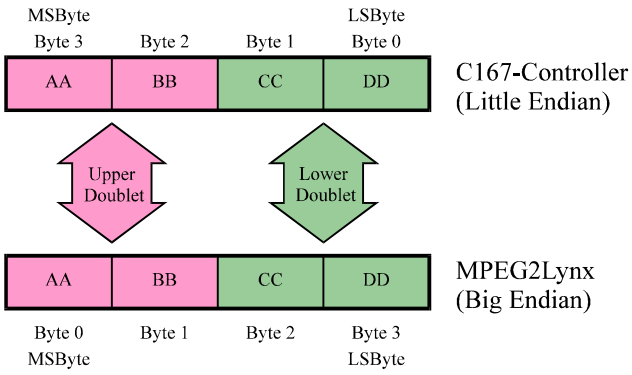

Bild 3: Adressierung der Doublets zwischen C167 und MPEG2Lynx

Die Registerzugriffsfunktionen können die vertauschte Adressierung auf verschiedene Weise realisieren:

1. Zugriff auf die einzelnen Doublets des LLC-Registers über Zeiger auf 16-Bit-Werte und nachträgliches Kombinieren zum Quadlet (Lesen) bzw. vorangehende Zerlegung des Quadlets (Schreiben). Diese Methode ist nötig, wenn zwischen den beiden Zugriffen noch andere Aktionen durchgeführt werden sollen. Bei einem Prototyp der verwendeten IEEE-1394-Plattform war z. B. eine spezielle Adreßberechnung für jedes Doublet erforderlich, um einen Hardwarefehler zu kompensieren.

2. Zugriff auf das gesamte Register über einen Zeiger auf 32-Bit-Werte und nachträgliches (Lesen) bzw. vorangehendes Vertauschen der Doublets im Quadlet (Schreiben). Die Vertauschung ist nötig, weil der C167-Mikrocontroller bei der Zerlegung des 32-Bit-Zugriffs in zwei Buszyklen das Doublet an der niedrigeren Adresse gemäß Little Endian als Lower Doublet und das an höheren Adresse als Upper Doublet interpretiert.

3. Zugriff auf das gesamte Register über einen Zeiger auf 32-Bit-Werte im Data Invariant Little Endian Mode des MPEG2Lynx. In diesem Modus vertauscht der MPEG2Lynx die Doublets automatisch, so daß er sich nach außen wie ein Little-Endian-System verhält. Diese Methode ist am elegantesten, da die Quadlets ohne Manipulation durch die Software übertragen werden können. Der Modus wird ebenfalls über das I/O-Kontrollregister des MPEG2Lynx gewählt, typischerweise zusammen mit den Ready- und Interrupteinstellungen. Da der MPEG2Lynx aber nach dem Einschalten oder einem Reset im Big-Endian-Modus arbeitet, muß bei diesem ersten Schreiben des Kontrollregisters die Vertauschung noch beachtet werden.

Bei der Portierung des Hardware Abstraction Layers wurden außerdem einige Fehler im HAL-Quellcode beseitigt. So übergibt die Error-Callback-Funktion des HAL einen Fehlercode an die IEEE-1394-Protokollsoftware, für den ein Parameter in der Funktionszeigerdeklaration fehlte. Die Zeitdifferenz zwischen IEEE-1394-Cycle-Time-Werten wurde durch die Funktion „GetElapsedCycleTimer“ falsch berechnet, da diese den Übertrag zwischen den Feldern nicht korrekt berücksichtigte. Außerdem waren einige Masken für LLC-Register sowie die Größeneinstellungen für die FIFOs falsch angegeben.

Grundlegende Fehler enthält zusätzlich die Fehlerbehandlung des Hardware Abstraction Layers. So lösen manche Busfehler den LLC-Interrupt aus, die aber in der Interrupt Service Routine gar nicht behandelt werden. Andere Fehlertypen werden zwar behandelt, es ist jedoch kein Fehlercode für die Callback-Funktion angegeben. Beim Verwerfen von FIFO-Daten aufgrund von Fehlern werden wichtige Konfigurationsdaten in den beeinflußten LLC-Registern überschrieben. Außerdem reagiert der HAL grundsätzlich nicht auf Broadcast Packets. Schließlich müßte das isochrone Modul „LynxStr“, sofern verwendet, fertig implementiert und ggf. auf die verwendete Zusatzhardware am Bulky Data Interface des MPEG2Lynx angepaßt werden. Da diese Fehler für die vorgesehenen, einfachen Testprogramme aber unkritisch sind, wurde im vorliegenden Fall auf ihre Korrektur zunächst verzichtet.

4 HAL-Testapplikation

Zur Überprüfung der auf die verwendete IEEE-1394-Plattform portierten Version des MPEG2Lynx Hardware Abstraction Layers wurde anstelle des nicht verfügbaren Embedded LynxSoft API von Texas Instruments eine eigene Testapplikation direkt auf den HAL aufgesetzt. Diese Testapplikation implementiert zur Vereinfachung keine vollständige IEEE-1394-Protokollsoftware, sondern nur die Teile, die zur Interaktion mit dem HAL erforderlich sind (s. Kap. 2). Sie wird zusammen mit dem HAL vom Keil-Compiler seriell auf die Embedded IEEE-1394-Plattform übertragen und über das Monitorprogramm des Keil-Debuggers direkt auf der Zielhardware betrieben. Der Benutzer erhält im seriellen Ausgabefenster des Debuggers Informationen zum Zustand des Embedded Knotens (Bild 4). Als Gegenstelle im PC dient eine PCI-Einsteckkarte „TSBKPCI“ von TI mit der zugehörigen IEEE-1394-Testsoftware „TI 1394 PCILynx Eval“. Mit dieser Software kann der Benutzer den Zustand des PC-Knotens überwachen sowie Pakete an den Embedded Knoten schicken und von diesem empfangen.

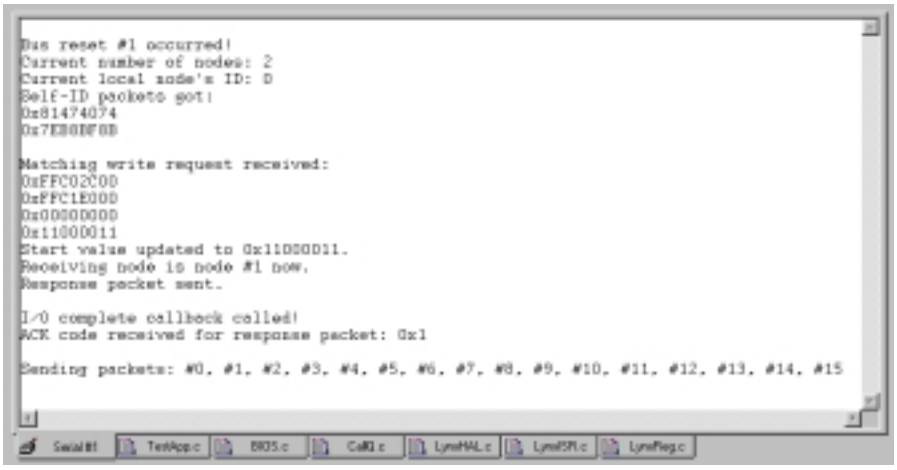

Bild 4: Serielle Ausgabe der Testapplikation im Debugger

Die Testapplikation erkennt Resets auf dem Bus und zeigt deren Anzahl, die Anzahl der Knoten, die lokale Knotennummer und die empfangenen Self-ID Packets im seriellen Fenster des Keil-Debuggers an. Weiter wartet sie auf den Empfang eines Quadlet Write Request Packets mit der Adresse 0xE000 0000 0000. Alle anderen Pakete werden ignoriert (kein Response Packet gesendet), wodurch beim Sender ein Split Transaction Timeout auftritt, den „PCILynx Eval“ entsprechend anzeigt. Beim Empfang des Quadlet Write Request Packets wird dieses angezeigt, das entsprechende Response Packet gesendet und der daraufhin empfangene ACK-Code ausgegeben. Danach sendet die Testapplikation ihrerseits 16 Quadlet Write Request Packets an den Knoten, von dem die empfangene Write Request stammt, und zwar an aufsteigende Adressen ab 0xE000 0000 0000. Das Datenquadlet der empfangenen Write Request multipliziert mit der Paketnummer bildet jeweils die Nutzdaten dieser Pakete. ACK-Codes und Response Packets für die 16 Pakete werden erneut ignoriert, da die Kontrolle der Übertragung leicht in „PCILynx Eval“ erfolgen kann. Dazu müssen nur 64 Byte ab Adresse 0xE000 0000 0000 im PC-Knoten zum Schreiben freigegeben und ihr Inhalt überwacht werden (Bild 5).

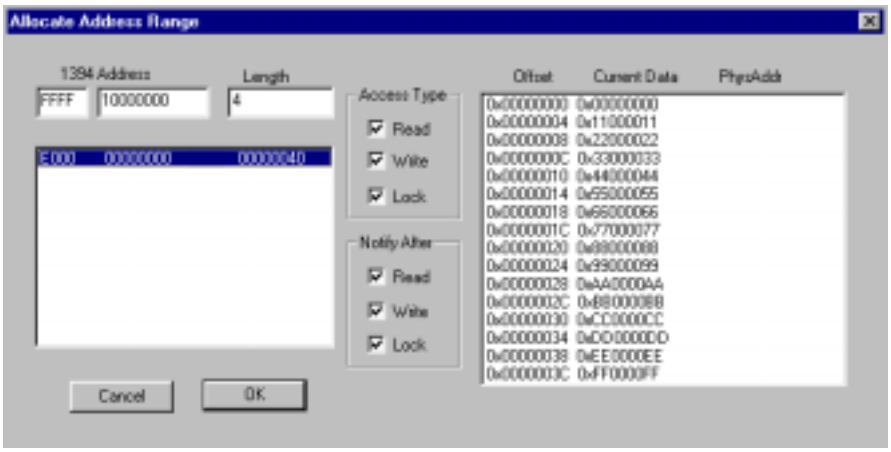

Bild 5: Kontrolle der Übertragung in „PCILynx Eval“

In dem in Bild 4 und Bild 5 dargestellten Beispiel wurde beim Start der Testapplikation auf dem Embedded Knoten (Knotennummer 0) ein Reset ausgelöst. Anschließend sandte der PC-Knoten (Knotennummer 1) das erwartete Write Request Packet. Aus diesem Paket wurden das Datenquadlet 0x1100 0011 als neuer Startwert und der Knoten 1 als neuer Empfänger decodiert. Schließlich übertrug der Embedded Knoten die 16 Pakete mit den Daten 0x0000 0000 bis 0xFF00 00FF korrekt an den PC-Knoten. Sollte hingegen während der Übertragung dieser Pakete ein Bus Reset auftreten, bricht die Testapplikation die Übertragung ab und geht wieder in den Wartezustand über, da sich die Knotennummer des Empfangsknotens geändert haben kann. Beim Empfang einer neuen, passenden Write Request während der Übertragung der 16 Pakete wird die Übertragung ebenfalls abge-

brochen. Die Testapplikation startet in diesem Fall aber sofort eine neue Übertragungssequenz mit den neu decodierten Werten.

Wie die Testapplikation zeigt, reicht die Funktionalität des MPEG2Lynx-HAL auch ohne zusätzliche IEEE-1394-Protokollsoftware für einfache Laborlösungen, die nicht voll standardkonform sein müssen, bereits aus. Um den Hardware Abstraction Layer zu einem funktionsfähigen IEEE-1394-Protokollstack zu erweitern, ist allerdings ein höherer Programmieraufwand erforderlich. Der genaue Aufwand hängt z. B. davon ab, ob der Knoten mehrere Transaktionen gleichzeitig bearbeiten soll, ob ein minimales oder ein allgemeines Configuration ROM verwendet wird und ob die optionalen, globalen Managementaufgaben implementiert werden. Auf jeden Fall müssen aber die erwähnten, grundlegenden Fehler des HAL beseitigt werden (s. Kap. 3).

5 Bewertung

Im Rahmen des vorliegenden Projekts sollte der Hardware Abstraction Layer von Texas Instruments für den MPEG2Lynx Link Layer Controller von TI als Referenz für die Anpassung des zugekauften IEEE-1394-Protokollstacks an den MPEG2Lynx dienen. Bei der zu diesem Zweck erforderlichen Portierung des MPEG2Lynx-HAL auf die verwendete Embedded IEEE-1394-Plattform mit dem C167-Mikrocontroller wurden wichtige Erfahrungen mit den kritischen Punkten dieser Anpassung gesammelt (Bus- und Interruptanbindung, Registerzugriff, Einsatz des Keil-Compilers). Zusätzlich erwies sich der HAL zur Einarbeitung in die hardwarenahen Aspekte des IEEE-1394-Protokolls und die Ansteuerung des Link Layer Controllers als durchaus geeignet.

Wegen dieser Vorarbeiten und Erfahrungen fällt es bei der laufenden Portierung des Protokollstacks nicht schwer, die Funktionen des mit dem Stack gelieferten Beispiel-HAL auf den MPEG2Lynx abzubilden. Ob sich eine Funktion des Beispiel-HAL bei Verwendung anderer Register und Masken auch für den MPEG2Lynx verwenden läßt oder ob die Struktur des MPEG2Lynx eine grundsätzlich andere Implementierung erfordert, ist durch Vergleich des MPEG2Lynx-HAL mit dem Beispiel-HAL leicht festzustellen. Im Zweifelsfall können einfach Routinen des Beispiel-HAL mit dem portierten MPEG2Lynx-HAL kombiniert und direkt auf der Zielhardware getestet werden, um eine geeignete Variante zu entwickeln.

Zusammenfassend läßt sich festzustellen, daß die Analyse des MPEG2Lynx-HAL-Quellcodes allein bereits eine deutliche Hilfe bei der Portierung des Protokollstacks im vorliegenden Projekt bietet. Somit sollte die Analyse eines HAL-Quellcodes für die jeweils verwendete IEEE-1394-Hardware bei vergleichbaren Projekten auf keinen Fall unterbleiben, sofern sich ein geeigneter Code beschaffen läßt. Die lauffähige Anpassung dieser HAL-Referenzsoftware an die jeweilige Plattform stellt wegen der daraus resultierenden Testmöglichkeiten eine noch wertvollere Hilfe bei der Portierung eines zugekauften Protokollstacks dar. Ob dies den Aufwand für die zusätzliche HAL-Anpassung rechtfertigt, ist von der Erfahrung mit der verwendeten IEEE-1394-Hard- und -Software abhängig und kann daher nicht allgemein beantwortet werden.

Die in diesem Artikel beschriebenen Arbeiten werden teilweise mit Mitteln des Landes Niedersachsen im Rahmen eines AGIP-Projektes gefördert.

Literatur

[1] Anderson, D.: *FireWire System Architecture: IEEE 1394a.* 2nd ed. Reading, Massachusetts u. a.: Addison Wesley, 1999. – ISBN 0-201-48535-4

[2] IEEE 1394-1995: *Standard for a High Performance Serial Bus.* New York: 1995. – ISBN 1-5593-7583-3

[3] Solomon, R.: *Embedded 1394 LynxSoft Hardware Abstraction Layer (HAL) for MPEG2Lynx and GPLynx.* Texas Instruments Inc., 15.10.1997. – Applikationsbericht, Publikation Nr. SLLA022. http://www.ti.com/sc/docs/psheets/abstract/apps/slla022.htm

[4] Solomon, R.: *Design of the GPLynx HAL*. Texas Instruments Inc., 06/1998. – Applikationsbericht, Publikation Nr. SLLA026. http://www.ti.com/sc/docs/psheets/abstract/apps/slla026.htm

[5] Keil Software Inc.: *Getting Started and Create Applications with µVision2 and 166/ST10 Microcontroller Development Tools*. 05/1999. – Benutzerhandbuch. Evaluations-Software und Dokumentation verfügbar unter http://www.keil.com/demo/default.htm

[6] Texas Instruments Inc.: *TSB12LV41 (MPEG2Lynx): IEEE 1394-1995 Link-Layer-Controller for Consumer Applications*. 10/1998. – Datenblatt, Publikation Nr. SLLS276A. http://www.ti.com/sc/docs/products/analog/tsb12lv41.html

[7] Siemens AG: *C167 Derivatives: 16-Bit CMOS Single-Chip Microcontrollers*. Version 2.0, 03/1996. – Benutzerhandbuch. http://www.infineon.com/products/micro/prod_lib/c166/c167.htm

Prof. Dr.-Ing. Rainer Bermbach studierte und promovierte an der TU Darmstadt. Nach längerer Industrietätigkeit nahm er 1994 einen Ruf an die FH Braunschweig/Wolfenbüttel an, wo er im Fachbereich Elektrotechnik die datentechnischen Fachgebiete vertritt. Das hier erwähnte Forschungsprojekt ist Teil eines FuE-Schwerpunktes, der sich mit dem Einsatz von Netzwerken in Embedded Anwendungen befaßt. Weitere Schwerpunkte seiner Arbeit liegen u. a. bei Embedded Controllern, Rechnerarchitekturen sowie Hardwarebeschreibungssprachen.
E-Mail: r.bermbach@fh-wolfenbuettel.de

Dipl.-Ing. (FH) Jens Brocke studierte Nachrichtentechnik mit dem Schwerpunkt Mikrocomputertechnik an der FH Braunschweig/Wolfenbüttel. Seit 1999 betreut er ein Drittmittel-Forschungsprojekt an der FH Braunschweig/Wolfenbüttel, in dem in Kooperation mit einem Industriepartner der Einsatz von IEEE 1394 für Multimediaanwendungen im Kfz evaluiert wird.
E-Mail: j.brocke@fh-wolfenbuettel.de