Universität Kaiserslautern
Lehrstuhl für Hochspannungstechnik und EMV

Diplomarbeit

Implementation eines Kernel-Treibers und einer Software-Schnittstelle zum Zugriff auf eine DSP-Karte über TCP/IP unter Linux

Klaus Knopper

Kaiserslautern, April 1998

Betreuer: Prof. Dr.-Ing P. Weiß
Dipl.-Ing. M. Renner

Erklärung

Hiermit versichere ich, daß ich die vorliegende Arbeit selbständig angefertigt und mich keiner anderen Quellen und Hilfsmittel als der angegebenen bedient habe.

Kaiserslautern, den 23.4.1998

Inhaltsverzeichnis

1 Einleitung

Digitale Signalprozessoren (DSPs) werden häufig zur Echtzeitverarbeitung von Meßsignalen und in multimedialen Anwendungen wie Video- und Audiokarten eingesetzt. Ein anderer Anwendungsbereich sind integrierte DSP-Systeme, die selbständig Aufgaben wie das Verarbeiten von Meßdaten nebst Auswertung und Ausgabe der Resultate verrichten. Wünschenswert für ein solches System ist die Möglichkeit, über eine Netzwerkstrecke mit TCP/IP als Netzwerkprotokoll auf den DSP und seine Peripherie zugreifen zu können, da die maximale Leitungslänge und die Fehlertoleranz einer Netzwerkstrecke deutlich größer sind als die einer direkten galvanischen Verbindung mit den Ein- und Ausgabeports des DSP. Außerdem können durch Netzwerkkomponenten wie Bridges oder Router DSP-Daten über lokale oder globale Datennetzwerke transportiert werden.

Ein denkbarer Ansatz ist es, mit dem DSP einen Netzwerkadapter auf einer Platine unterzubringen und einen Microkernel mit TCP/IP Stack für diese Kombination zu implementieren. Dies bedingt allerdings einen nicht unerheblichen Aufwand an Hard- und Softwaredesign und bringt eine zusätzliche Belastung des DSP mit Aufgaben mit sich, für die dieser nicht konstruiert ist, schränkt also dessen Arbeitsgeschwindigkeit für die eigentliche Applikation ein, da er sich auch um Generierung und Beantwortung von TCP-Paketen kümmern muß.

Praktikabler erscheint eine Lösung, in der der DSP über seine Ein- und Ausgabeschnittstellen mit einem Host-Rechner und dem dazugehörenden Host-Betriebssystem kommuniziert, und dieses Host-Betriebssystem die Aufgabe des Datentransports über eine TCP/IP Schnittstelle übernimmt. Ein großer Vorteil dieses Ansatzes ist die Tatsache, daß Implementationen von TCP/IP bereits für die meisten Rechnerarchitekturen vorhanden sind und übernommen werden können.

Da Motorola DSP-Systeme auf Karten mit ISA und PCI-Schnittstelle vertreibt, bietet sich ein Hostsystem bestehend aus CPU-Board, DSP-Karte und einer Standard-Netzwerkkarte als Lösung an. Da der bei einer solchen Kompaktlösung zur Verfügung stehende Raum für das Hostsystem begrenzt ist, muß das Host-Betriebssystem mit Rücksicht auf Minimierung der benötigten Ressourcen an Festplattenkapazität, auf Stabilität und Leistungsfähigkeit gewählt werden. Linux erfüllt diese Anforderungen und ist aufgrund seiner offenen Architektur für diesen Zweck geeignet.

Ziel dieser Arbeit ist es, unter Linux als Host-Betriebssystem einen Treiber für die Motorola DSP56301ADM Entwicklerkarte zu implementieren, mit dessen Hilfe die Karte vom Host-Betriebssystem aus angesprochen werden kann. Die Steuerung der Karte, Programmierung des DSP und Übertragung von Daten vom DSP soll dabei über ein TCP/IP-basiertes Netzwerk möglich sein. Eine Software-Schnittstelle für Applikationsentwickler ist zu entwickeln und zu dokumentieren.

2 Analyse der Aufgabenstellung

Eine wichtige Voraussetzung für eine zufriedenstellende Arbeitsgeschwindigkeit des DSP in einem eingebetteten Hostrechner ist ein flüssiges Zusammenspiel zwischen DSP, Software und Ein-/Ausgabekanälen. Die Eingangsdaten (beispielsweise digitalisierte Analogmeßwerte eines Meßwertaufnehmers) müssen abgetastet werden, die vom DSP berechneten Funktionswerte[1] müssen vom DSP bzw. dessen Datenregistern an ein Anwendungsprogramm oder eine Ausgabeeinheit transportiert werden.

Für den Datentransport vom Meßwertaufnehmer zum DSP nach erfolgter Analog-Digital-Wandlung sind Hardware-Lösungen in der Anwendung. Dies sind A/D-Wandlereinheiten, die digitalisierte Abtastwerte eines analogen Signals mit einem mit dem DSP synchronisierten Takt in einem vom DSP abrufbaren Datenregister oder I/O-Bereich liefern. Die hierbei auftretenden Problemstellungen wie Digitalisierrauschen, Fehlertoleranzen und Grenzwertbetrachtungen sind in der Meßtechnik bekannt und sollen kein Gegenstand dieser Arbeit sein.

Hier ist der Datentransport vom DSP an eine geeignete Ausgabeeinheit, an ein Speichermedium oder ein Anwenderprogramm, das auf einem mit dem DSP-Subsystem gekoppelten Hostrechner läuft, zu betrachten. Handshaking-Mechanismen sind erforderlich, die gewährleisten, daß die Anforderungen, die an eine Übertragung von digitalisierten Daten im allgemeinen und von Meßdaten im speziellen gestellt werden, erfüllt sind, nämlich:

1. Gültigkeit der vom DSP gelieferten Daten,
2. Fehlerfreiheit oder Fehlererkennung während der Übertragung,
3. Aktualität der Daten oder definierte Verzögerungsintervalle.

Punkt 1 erfordert eine Erkennung der Bereitschaft zur Datenübergabe, was allgemein durch das Auslösen eines Interrupts oder das Setzen eines Flags in einem Statusregister, welches vom Kommunikationspartner gelesen wird, geschehen kann.

Punkt 2 ist bei den meisten Bussystemen bereits durch die Parity-Überprüfung, die von der Hardware vorgenommen wird, erfüllt. Abgesehen davon, daß eine fehlerhafte Datenübertragung bei modernen Peripheriegeräten nur unter sehr extremen Arbeitsbedingungen vorkommt, ist durch einfache, vom Bus-Controller des Hostsystems berechnete Prüfsummen die Erkennung eines fehlerhaften Datentransfers mit vom Treiber und Anwenderprogramm unabhängiger Korrektur möglich. Eine weitere Plausibi-

[1] Ergebnisse einer Frequenzanalyse mit FFT u.ä.

litätsprüfung der vom DSP gelieferten Werte durch das Anwenderprogramm ist daher selten erforderlich.[2]

Aus Punkt 3 ergibt sich die Forderung nach einer ausreichenden Geschwindigkeit des Datenstroms bzw. einer ausreichenden Bandbreite des Transportkanals in Verbindung mit dem Transportprotokoll. Bei einer TCP-Datenübertragung über eine 100 Mbit-Verbindung sind Werte von 3-5 MB/s typisch. Der Treiber sollte daher einen entsprechenden Durchsatz bei der Datenübertragung vom und zum DSP gewährleisten.

[2] Dies betrifft nicht fehlerhafte Meßwerte, die bereits vom A/D-Wandler an den DSP geliefert werden können.

3 Die Entwicklerkarte DSP56301ADM

Das von Motorola hergestellte Application Development Module Board DSP56301ADM [1], im folgenden als „Entwicklerkarte DSP56301“ oder einfach als „DSP-Karte“ bezeichnet, beinhaltet die folgenden Hardware-Komponenten:

- Motorola DSP 56301 (24-Bit Digital Signal Processor),
- 32 Kilobyte statisches RAM,
- 512 Kilobyte dynamisches RAM,
- 64 Kilobyte flash PROM,
- ISA- und PCI-Steckerleiste (Anschluß wahlweise),
- Steckkontakt-Leisten für Erweiterungen wie Logic Analyzer, SSI (Syncronous Serial Interface), SCI (Serial Communication Interface) oder JTAG/OnCE[3] Modul.

Die Karte besitzt eine Reihe von Jumpern, mit denen die Konfigurationsparameter gesetzt werden können, unter anderem die Arbeitsweise als ISA- oder PCI-Karte. Als ISA-Karte wird ein wählbarer Interrupt des Hostsystems belegt und ein DMA-Kanal des Hosts benutzt. Für den DMA-Betrieb im ISA-Modus sind 5 gesockelte Puffer-Bausteine auf der Rückseite des Boards angebracht, die beim Betrieb der Karte als PCI-Board entfernt werden müssen.

Mit Hilfe eines DIP-Schalters kann die Art der Initialisierung des DSP eingestellt werden. Die DPSs der Serie 56300 besitzen ein 192 24-Bit Worte großes Urlader-ROM, um ein DSP-Programm über dessen Schnittstellen in den Programmspeicher zu laden und auszuführen. Als Varianten stehen Bootstrap vom on-Board flash PROM, von der SCI-Schnittstelle, vom PCI-Bus oder per Universal/ISA-Bus zur Verfügung. Das DSP56301 Bootstrap-Programm für den PCI-Modus ist im Anhang A auf Seite 60 aufgelistet.

Im PCI-Modus sind die Pins des HI32-Port[4] des DSP 56301 direkt mit den Steckkontakten der PCI-Schnittstelle verbunden. Es werden keine Interrupts und keine DMA-Kanäle des Hosts beansprucht. Mit Hinsicht auf zeitgemäße Hardware und den höheren Busdurchsatz wird diese Betriebsart gewählt.

[3] On Chip Emulator

[4] Host Interface 32 Bit

Als Hostsystem wird ein Rechner mit folgender Ausstattung eingesetzt:

- Intel Pentium MMX 166MHz,
- 32 MB RAM,
- 4 PCI, 4 ISA Slots,
- 4 MB PCI Gafikkarte,
- NE2000-kompatible ISA-Netzwerkkarte
- Betriebssysteme: Linux (RedHat 4.2 Distribution), DOS und Windows NT[5].

Als Entwicklungssystem steht der GNU C Compiler Version 2.7.2.1, die Linux-Kernel Quelltexte Version 2.0.32 und ein DSP56300-Assembler (unter DOS) sowie ein DSP56300-Simulationsprogramm (unter Windows NT), beide von Motorola, zur Verfügung.

[5]Windows NT ist ein Warenzeichen von Microsoft.

4 Linux als Host-Betriebssystem

Im folgenden wird die Wahl von Linux als Host-Betriebssystem für die Steuerung der DSP56301-Karte begründet. Die Betrachtungen gelten größtenteils für alle Unix[6]-Betriebssysteme, zu denen auch Solaris[7], BSD, Aix[8], HP-UX[9] und andere gehören. Im Detail wird jedoch nur auf die Linux-spezifischen Eigenschaften des Host-Betriebssystems eingegangen.

4.1 Linux – Überblick

Linux ist ein weitgehend POSIX-konformes [4], multitasking- und multiuserfähiges Unix-artiges Betriebssystem, das sich seit vielen Jahren aufgrund seiner hohen Betriebssicherheit und guten Unterstützung verschiedenster Prozessorarchitekturen großer Beliebtheit erfreut.

Linux wurde 1991 von dem finnischen Informatikstudenten Linus Torvalds zunächst als experimentelles Betriebssystem für Intel-basierte Rechnerachitekturen entwickelt und im Internet unter der *GNU General Public License* (Anhang D) veröffentlicht. Die Bedingungen der GPL gestatten es allen Nutzern von Linux, die Quelltexte des Betriebssystems, die größtenteils in der Programmiersprache C verfaßt sind, zu kopieren, zu modifizieren und die Modifikationen ihrerseits wieder unter der GPL zu veröffentlichen. Das erklärte Ziel der GPL ist es hierbei, die selbständige Fort- und Weiterentwicklung von Software ohne Begrenzung der Rechte am Quelltext zu garantieren. Daher wird der in dieser Arbeit entwickelte Treiber ebenfalls unter den Konditionen der GPL weitergegeben und veröffentlicht. Dies hat den Vorteil, daß den Nutzern des Treibers auch dessen Quelltext zur Modifikation für zukünftige Erweiterungen oder Unterstützung neuer Motorola-DSP-basierter Karten zur Verfügung steht.

Unix im allgemeinen und Linux im speziellen werden vor allem im Server-Bereich eingesetzt, wenn Stabilität und Sicherheit in heterogenem Umfeld ein Entscheidungskriterium für die Wahl des Betriebssystems sind. Da die Netzwerkunterstützung auf Basis von TCP/IP bereits integraler Bestandteil des Betriebssystems ist und die Anforderungen von Linux an die Hardwareausstattung sehr gering sind, eignet sich Linux hrevorragend als Host-Betriebssystem zur Implementation eines Treibers für eine Karte, die über ein IP-basiertes Netzwerk-Interface ansprechbar sein soll. Linux stellt für alle gängigen Netzwerkkarten Treiber zur Verfügung, so daß für die reine Datenüber-

[6]UNIX ist ein Warenzeichen von AT&T.
[7]SOLARIS ist ein Warenzeichen von Sun Microsystems.
[8]AIX ist ein Warenzeichen von IBM.
[9]HP-UX ist ein Warenzeichen von Hewlett Packard.

tragung per TCP/IP von und zur DSP-Karte keine weitere Entwicklungsarbeit notwendig ist.

Obwohl DSPs häufig für Echtzeitanwendungen (beispielsweise Verarbeitung von Meßdaten) eingesetzt werden, ist zur Entgegennahme der vom DSP berechneten Resultate nur selten ein echtzeitfähiges Betriebssystem erforderlich. Linux unterstützt zwar Echtzeitapplikationen[10] und die Vergabe von Prioritäten für Einzelprozesse, in der Praxis wird hiervon jedoch kaum Gebrauch gemacht. Auch der zu erstellende Treiber soll mit Rücksicht auf parallel ablaufende Prozesse des Hostrechnersystems implementiert werden, wobei jedoch die Datenübertragung vom DSP zum Anwendungsprogramm und umgekehrt Vorrang vor anderen Tasks haben soll.

4.2 Peripheriegeräte und Gerätedateien

Schnittstellen, Peripheriegeräte, Hauptspeicher und Speichermedien werden unter Unix als sogenannte „special device files“, Gerätedateien im Dateisystem, zumeist im Verzeichnis `/dev`, abgebildet. Es handelt sich hierbei um virtuelle Dateien, die den Zugriff auf ein Gerät mit standardisierten Systemaufrufen ermöglichen. Der Systemkern erkennt anhand einer der Datei zugeordneten Identifikation, um welches Gerät es sich handelt, und aktiviert den zum Zugriff erforderlichen Gerätetreiber.

Eine Gerätedatei identifiziert das zugehörige Gerät mit den in der folgenden Beschreibung angegebenen Parametern. Der Dateiname der Gerätedatei ist prinzipiell beliebig wählbar, jedoch ist ein Hinweis auf die Art des Gerätes durch den gewählten Namen sinnvoll.

Typ `c` oder `b`: Identifikation eines **character** oder **block**devices. Character devices (zeichenorientierte Geräte) ermöglichen einen meist nur sequentiellen, ungepufferten Zugriff, block devices (blockorientierte Geräte) einen dynamisch gepufferten, wahlfreien Zugriff auf Segmente innerhalb des Adreßraums oder Speichers des zugehörigen Gerätes.

major number: Die major number identifiziert den zur Datei gehörenden Treiber und die Art des Gerätes. In der Dokumentation des Kernel-Quelltextes unter `/usr/src/linux/Documentation` sind die Kennungen der am häufigsten verwendeten Gerätedateien in der Datei `devices.txt` abgelegt.

minor number: Die minor number kann zur Unterscheidung von mehreren, mit dem gleichen Treiber arbeitenden, virtuellen Geräten dienen. Ein Beispiel hierfür ist

[10] mit einer Kernelerweiterung

der Soundkartentreiber, bei dem über die minor number das Subsystem „Aus-/Eingabe“, „Mixer“ oder “Midi-Schnittstelle“ oder eine zweite Soundkarte ausgewählt werden kann.

Gerätedateien im Verzeichnis `/dev` werden mit dem Unix-Kommando `mknod Name typ major minor` erzeugt. Die Berechtigungen zum Zugriff auf die Gerätedatei und damit die Zugriffskontrolle für Benutzer und Prozesse können wie unter Unix üblich mit dem Kommando `chmod Name Berechtigung` erteilt werden.

Tabelle 1 zeigt eine Auswahl der unter Linux üblichen Gerätedateien und ihrer Identifikationen aus `devices.txt`. Die major number 0 dient zur dynamischen Zuweisung einer freien major number an Geräte, die keine feste Gerätekennung benutzen.

0		Unnamed devices (e.g. non-device mounts)
1	char	Memory devices
1	block	RAM disk
2	char	Pseudo-TTY masters
2	block	Floppy disks
3	char	Pseudo-TTY slaves
3	block	First MFM, RLL and IDE hard disk/CD-ROM interface
4	char	TTY devices
5	char	Alternate TTY devices
6	char	Parallel printer devices
7	char	Virtual console access devices
7	block	Loopback devices
8	block	SCSI disk devices
9	char	SCSI tape devices
9	block	Metadisk (RAID) devices
10	char	Non-serial mice, misc features
11	char	Raw keyboard device
11	block	SCSI CD-ROM devices
12	char	QIC-02 tape
12	block	MSCDEX CD-ROM callback support
13	char	PC speaker
13	block	8-bit MFM/RLL/IDE controller
14	char	Sound card
		...

Tabelle 1: Einige der unter Linux verwendeten major device numbers.

Für die Device-Datei der DSP-Karte wurde aus praktischen Überlegungen ein zeichenorientierter Zugriff gewählt. Hierdurch ist ein direktes, ungepuffertes Lesen vom und

Schreiben zum Transferregister der Karte möglich, wobei die Caching-Mechanismen des virtuellen Unix-Dateisystems nicht in Anspruch genommen werden. Um Kollisionen mit major device numbers neuer, zukünftiger Gerätetreiber zu vermeiden, wurde die major number **101** für den zu implementierenden Treiber bei der in den Kernel-Quelltexten mitgelieferten Datei `devices.txt` angegebenen Kontaktadresse offiziell registriert.

Mit dem Treiber für die DSP56301-Karte sollen bis zu maximal 16 gleichzeitig arbeitende DSP-Karten bedient werden können. Daher entschied sich der Autor für die in Tabelle 2 angegebene Einteilung der minor device numbers. Die minor number 0 bezeichnet eine Gerätedatei mit dem Namen `/dev/mdspstat`, aus der Statusinformationen über alle vom Treiber erkannten und initialisierten Karten abgerufen werden können. Dies entspricht in der Funktionsweise der Datei `/dev/sndstat`, die vom Soundkarten-Treiber unterstützt wird.

major 101		char device	Motorola DSP 56xxx Board
	minor 0	`/dev/mdspstat`	Statusinformation
	minor 1	`/dev/mdsp1`	1. DSP Board, I/O und Steuerung
	minor 2	`/dev/mdsp2`	2. DSP Board, I/O und Steuerung
	minor 3	`/dev/mdsp3`	3. DSP Board, I/O und Steuerung
		...	
	minor 16	`/dev/mdsp16`	16. DSP Board, I/O und Steuerung

Tabelle 2: Major und minor device numbers der DSP56xxx-Serie.

4.3 Modularisierung des Betriebssystemkerns

Mit der Version 2.0 von Linux wurde eine weitestgehende Modularisierung des Linux-Kernels erreicht. Dies ermöglicht es, beim Zugriff auf ein Gerät den hierfür benötigten Treiber dynamisch nachzuladen und ihn gegebenenfalls wieder aus dem System zu entfernen und benötigte Ressourcen freizugeben, wenn seine Dienste längere Zeit nicht mehr in Anspruch genommen werden.

Diese Vorteile der Modultechnik wurden zur Implementation des Treibers für die DSP-Karte genutzt, daher ist der Treiber als Modul separat nachladbar und muß nicht statisch in den Linux-Kernel eincompiliert werden. Ein weiterer Vorteil ist, daß bei Änderungen am Treiber kein System-Neustart erforderlich ist: Der alte Treiber kann zur Laufzeit entfernt[11] und der neue eingesetzt[12] werden.

[11] Kommando **rmmod**.
[12] Kommando **insmod**.

Der Quellcode des Kernel-Moduls für die DSP-Karte beinhaltet die Datei `mdsp.h`, welche Definitionen und Konstanten sowohl für den Treiber selbst als auch für Applikationen, die den Treiber benutzen, enthält, sowie die Datei `mdsp.c`, die die Variablen und Prozeduren des Treibers beinhaltet.

Anhang B beschreibt die Übersetzung der Quelltexte und die Installationsprozedur, die zur Einbindung des Treibers in den laufenden Betrieb erforderlich ist. Eine notwendige Vorbedingung ist, wie in diesem Abschnitt erläutert, daß der verwendete Linux-Kernel des Hostsystems mit Modulunterstützung übersetzt wurde.

5 Implementation des Kernel-Moduls

In den folgenden Abschnitten wird auf den Aufbau des Treibers für die DSP-Karte, die konkrete Implementation der vom Treiber zur Verfügung gestellten Systemroutinen und auf den Zusammenhang mit dem virtuellen Dateisystem (VFS) unter Unix eingegangen.

5.1 Prozeduren des Virtual File System

Auf character-device-Dateien wird i.d.R. mit den gleichen Operationen zugegriffen wie auf „normale“ Dateien, die Daten enthalten oder Daten aufnehmen können. Einige der Standard-Operationen aus der C-Standardbibliothek unter Unix, die auf Dateien angewendet werden können, zeigt Tabelle 3.

C-Funktion	Beschreibung
`open()`	Öffnen einer Datei,
`read()`	Lesen aus einer Datei,
`write()`	Schreiben in eine Datei,
`seek()`	Positionierung innerhalb einer Datei
`select()`	Überprüfung, ob innerhalb einer Zeitspanne Daten aus einer Datei gelesen oder Daten geschrieben werden können.

Tabelle 3: Dateioperationen der Standard C-Bibliothek unter Unix

Ein Zugriff auf eine vom Treiber unterstützte DSP-Karte soll also wie das Öffnen, Lesen, Schreiben und Schließen einer Datei aus dem Anwenderprogramm heraus erfolgen. Quelltext 1 zeigt ein einfaches Beispiel.

```
#include<fcntl.h>
main()
{
 int fd;
 int number=10;
 fd=open("/dev/mdsp1",O_WRONLY);
 if(fd<0) exit(-1);
 write(fd,(char *)&number,sizeof(int));
 close(fd);
}
```

Quelltext 1: Ein einfaches Beispielprogramm

Es wird versucht, die Gerätedatei `/dev/mdsp1` zum Schreiben zu öffnen. Bei Erfolg wird der Wert 10 in die Datei geschrieben bzw. muß vom Treiber an den DSP übertragen werden.

Der Treiber wird also über die vom C-Programmierer gewohnte Softwareschnittstelle angesprochen und muß die zur Bearbeitung einer Aufgabe erforderliche Funktionalität bieten.

5.2 Prozeduren des Kernel-Moduls

Zur Implementation eines Kernel-Moduls unter Linux gibt es eine definierte Programmierschnittstelle [5]. In einer Struktur vom Typ `file_operations` (Quelltext 2) werden Zeiger auf alle Funktionen abgelegt, die das Modul dem virtuellen Dateisystem für diese Gerätedatei zur Verfügung stellt. Diese Struktur, die in `/usr/include/linux/fs.h` definiert ist, wird beim Laden des Moduls durch den `kerneld` oder mit dem Kommando `modprobe` in der Prozedur `init_module()` an den Kernel übergeben und das Gerät mit der zur Gerätedatei gehörenden major number wird im System angemeldet.

Der Name dieser Initialisierungsroutine in der Modul-Objektdatei ist festgelegt und nicht frei wählbar. Sie wird automatisch beim Laden des Moduls aufgerufen. Ebenso muß eine Prozedur mit dem Namen `cleanup_module()` existieren, die beim Entfernen des Moduls die Freigabe von Ressourcen, die durch den Treiber belegt wurden, vornimmt, und das Gerät mit der entsprechenden major number vom System abmeldet.

```
struct file_operations {
  int (*lseek) (struct inode *, struct file *, off_t, int);
  int (*read) (struct inode *, struct file *, char *, int);
  int (*write) (struct inode *, struct file *, const char *, int);
  int (*readdir) (struct inode *, struct file *, void *, filldir_t);
  int (*select) (struct inode *, struct file *, int, select_table *);
  int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
  int (*mmap) (struct inode *, struct file *, struct vm_area_struct *);
  int (*open) (struct inode *, struct file *);
  void (*release) (struct inode *, struct file *);
  int (*fsync) (struct inode *, struct file *);
  int (*fasync) (struct inode *, struct file *, int);
  int (*check_media_change) (kdev_t dev);
  int (*revalidate) (kdev_t dev);
};
```

Quelltext 2: Die `file_operations`-Struktur

Die Namen der anderen Prozeduren, die die eigentlichen Aufgaben des Treibers übernehmen, sind hingegen frei wählbar. Dem Kernel sind die Einsprungadressen und Aufrufparameter nach der Registrierung der `file_operations`-Struktur mittels der Systemfunktion `int register_chrdev(int major, char *name, struct file_operations *fops);` bekannt.

Es ist sinnvoll, alle Variablen und Prozeduren des Moduls (mit Ausnahme der beiden genannten `init_module()` und `cleanup_module()`) als `static` zu deklarieren. So werden nur wenige Symbolinformationen in der Objekt-Datei angelegt und Kollisionen von Prozedurnamen und Variablen im Zusammenspiel von Modulen vermieden.

Auf Wunsch des Fachbereichs wurden im Quelltext alle Systemroutinen, die vom Linux-Kernel zur Verfügung gestellt werden, per `#define` mit dem Präfix `kernel_` versehen. Vom Linux-Kernel exportierte Variablen werden entsprechend mit `kernel_var_` gekennzeichnet. Mit Rücksicht auf die Distribution des Moduls unter der GPL (Abschnitt D) wurden alle Funktionsnamen und Kommentare im Quelltext in Englisch gehalten.

Im folgenden werden die im Treiber implementierten Prozeduren und Funktionen unter Berücksichtigung der Benutzung durch ein Anwenderprogramm beschrieben.

5.2.1 `init_module()`

Der Kernel kann auf zwei Arten veranlaßt werden, das `mdsp`-Modul zu laden.

1. Mit dem Kommando `insmod mdsp` wird ein Laden des Moduls erzwungen.
2. Beim Zugriff auf eine Gerätedatei mit der major device number 101 lädt der `kerneld` automatisch das `mdsp`-Modul.

Im Fall 2 muß in der Konfigurationsdatei des `kerneld`, `/etc/conf.modules`, ein Eintrag der Form

```
alias char-major-101 mdsp
```

vorhanden sein, damit der `kerneld` die Zugehörigkeit des `mdsp`-Moduls zu zeichenbasierten Geräten mit der major device number 101 erkennen und das Modul beim Zugriff auf das Gerät in den Kernel einbinden kann.

Wird der Zugriff auf das Gerät beendet (z.B. durch Schließen der Gerätedatei durch den aufrufenden Prozeß), kann der `kerneld` das Modul nach einer durch Kommandozeilen-Parameter spezifizierbaren Wartezeit wieder deinitialisieren und den Speicher, der für das Modul verwendet wurde, freigeben. Dies bedingt, daß die Routinen, die beim Öffnen und Schließen der Gerätedateien aufgerufen werden, einen Zähler verwalten, der dem Benutzungsgrad des zugehörigen Treibers entspricht. Erst, wenn alle Zugriffe beendet sind, darf der Treiber aus dem Kernel entfernt werden, da sonst die von Benutzerprozessen angeforderten Dienste nicht mehr verfügbar sind.

Quelltext 3 zeigt die Initialisierungsroutine, die beim Laden des Moduls ausgeführt wird.

```
/* Module initialization at loading time of driver */
int init_module(void)
{
 int result;
 if(init_mdsp())
  {
   PRINTK("No DSP boards found.\n");
   return -EIO;
  }
 /* Register file operations for mdsp device */
 if((result=kernel_register_chrdev(mdsp_major, mdsp_name, &fops_mdsp))==-EBUSY)
  {
   PRINTK("Unable to get major %d for MDSP device, errno=%d.\n",mdsp_major,
          result);
   return -EIO;
  }
 PRINTK("Registered Device %s, major %d.\n", mdsp_name, mdsp_major);
 return 0;
}
```

Quelltext 3: `init_module()`

Zunächst wird in die Routine `init_mdsp()` verzweigt, die alle vom Treiber unterstützten PCI DSP-Karten identifiziert und die Adressen ihrer Register in die Liste `mdsp_board[]` einträgt. Diese für die Hardware-Erkennung sehr wichtige Prozedur soll etwas ausführlicher betrachtet werden.

```
/* Board initialization */
static int init_mdsp()
{
 int i,result;
 unsigned int base;

 if(!kernel_pcibios_present())
  {
   PRINTK("No PCI-BIOS present, exiting.\n");
   return -1;
  }
```

Quelltext 4: Überprüfung auf PCI-System

Zunächst wird mit einer Funktion aus dem Linux-Kernel, `pcibios_present()`, überprüft, ob der Kernel ein PCI-System unterstützt und ob sich das zum PCI-Bus gehörende PCI-Bios zurückmeldet. Falls dies nicht der Fall ist, kehrt `init_mdsp()` sofort mit einem Fehlerwert zur aufrufenden Prozedur `init_module()` zurück.[13]

Bevor die Initialisierungsroutine weiter betrachtet wird, muß auf eine wichtige Strukturdefinition zu Beginn der Datei `mdsp.c` verwiesen werden.

[13]Fast alle Kernel-Funktionen liefern im Fehlerfall einen negativen Wert zurück, der der mit negativem Vorzeichen versehenen Systemvariablen `errno` entspricht, und von der aufrufenden Funktion ausgewertet werden kann. Ausnahmen sind Funktionen, die beispielsweise Adressen zurückliefern. Bei diesen wird im Fehlerfall 0 zurückgegeben.

```
/* Board parameters */
typedef struct _mdsp_info
{
 unsigned char busnr,devnr,irq,irqtype;
 unsigned short class,ready;
 unsigned int boardtype;
 unsigned int base;        /* Base address MDSP registers                    */
 unsigned int basetype;    /* Memory-mapped for DSP56301                     */
 unsigned int hctr,        /* Control Register Address                       */
              hstr,        /* Status Register Address                        */
              hcvr,        /* Host Control Vector Register                   */
              htxr,        /* Transmission Register Address (Input to DSP) */
              hrxr;        /* Receive Register Address   (Output from DSP) */
 struct wait_queue *wait;
} mdsp_info;

static mdsp_info mdsp_board[MDSP_MAX]; /* structs for all boards found */
```

Quelltext 5: Die `mdsp_info`-Struktur

`MDSP_MAX` ist in `mdsp.h` mit dem Wert 16 definiert. Stehen normalerweise auch in herkömmlichen Rechnern keine 16 PCI-Slots zur Verfügung, so soll doch die Möglichkeit, 16 DSP-Karten gleichzeitig zu bedienen, für größere Systeme mit kaskadierten PCI-Bussen bestehen.

Im Datenfeld `mdsp_board[]` werden die beim Reboot autokonfigurierten Parameter der einzelnen Karten gespeichert, nachdem sie wie folgt in der Initialisierungsroutine des `mdsp`-Treibers ausgelesen wurden.

```
 for(i=0;i<MDSP_MAX;i++)
  {
   mdsp_board[i].ready=0;   /* Flag non-initialized    */
   mdsp_board[i].wait=NULL; /* Zero waitqueue          */
  }
 for(i=0;i<MDSP_MAX;i++)
  {
   int j,found=0;
   /* Find all supported devices */
   for(j=0;j<sizeof(mdsp_type)/sizeof(unsigned short);j++)
    {
     found=(!kernel_pcibios_find_device(PCI_VENDOR_ID_MOTOROLA,mdsp_type[j],
                                      i,&mdsp_board[i].busnr,
                                      &mdsp_board[i].devnr));
     if(found)
      {
       mdsp_board[i].boardtype=mdsp_type[j];
       break;
      }
    }
   if(!found) break;
```

Quelltext 6: Finden aller unterstützten Motorola DSP-Karten im PCI-System

Zunächst werden alle 16 Einträge der Feldes `mdsp_board[]` als inaktiv (`ready=0`) markiert. `wait` ist ein Zeiger auf eine Linux-spezifische `wait_queue`-Struktur, die für blockierende Systemaufrufe, die das Multitasking jedoch nicht behindern sollen, benötigt wird. Sie wird ausschließlich in `select_mdsp()` verwendet und vom System initialisiert.

Die aus dem Linux-Kernel mit PCI-Unterstützung stammende Funktion `pcibios_find_device()` durchsucht den PCI-Bus Slot für Slot nach Boards, die auf die angegebenen Herstellerdaten passen. Motorola PCI Karten besitzen immer den Wert `0x1057` als Identifikation im Vendor-ID Register. `mdsp_type[]` ist ein in `mdsp.c` deklariertes Feld, das die boardspezifischen Identifikationen aller DSP-Karten enthält, die vom Treiber unterstützt werden sollen. Derzeit enthält dieses Feld ausschließlich die Identifikation `0x1801` für die DSP56301ADM. Weitere Kartentypen können problemlos in das Feld eingefügt werden, da sich die `for()`-Schleife dynamisch an die Feldgröße von `mdsp_type[]` anpaßt.

Wurden keine Motorola DSP-Karten gefunden, wird die äußere `for()`-Schleife verlassen und es erfolgt eine entsprechende Kernel-Meldung. Der aufrufende Prozeß bekommt als Rückgabewert eine Fehlermeldung vom Treiber zurück.

Für die mit dieser Kombination aus Herstellerfeld und Kartentyp-Feld gefundenen PCI-Karten werden die Busnummer (üblicherweise 0 bei Rechnern ohne kaskadierte PCI-Bussysteme) und die Devicenummer im `mdsp_board[]`-Strukturfeld vermerkt. Die weiteren Aufrufe zum Auslesen der PCI-Konfiguration bzw. zur Konfiguration der Karten bekommen diese beiden Identifikationsnummern übergeben.

Alle `pcibios_()`-Aufrufe werden unter Linux mittels der architekturabhängigen Header-Dateien unter `/usr/include/asm` in entsprechende Assemblermakros umgewandelt, die die für die jeweilige Architektur spezifischen Interruptroutinen ausführen. Für die Programmierung eines Kernel-Treibers kann generell unter Linux fast vollständig auf inline-Assembler verzichtet werden.

Die in Quelltext 7 ausgeführten PCI-Bios-Funktionen tragen die vom Board gelieferten Informationen *PCI-Klasse*[14], *physikalische Adresse* (Basisadresse der dualported Register der DSP56301) und Interrupt (immer 0 bei der DSP56301, es werden keine Interrupts auf der Host-Seite erzeugt oder erwartet) in das `mdsp_board`-Strukturfeld ein.

Bevor auf die Karte vom Kernel bzw. vom aufrufenden Benutzerprozeß zugegriffen werden kann, muß in der PCI-Konfiguration der Karte der Zugriff auf die Register der Karte über direkte Adressierung ermöglicht werden[15] (Quelltext 8). Hier wird auch die Fähigkeit der Karte, Parity-Fehler zu erkennen und entsprechende Statusflags zu setzen, eingeschaltet.

[14] zu rein informativem Zweck

[15] *memory space access.*

```
    if((result=kernel_pcibios_read_config_word(mdsp_board[i].busnr,
                                               mdsp_board[i].devnr,
                                               PCI_CLASS_DEVICE,
                                               &mdsp_board[i].class)))
     {
      PRINTK("Failed to read class type from PCI board, errno=%d\n",result);
      continue;
     }
    if((result=kernel_pcibios_read_config_dword(mdsp_board[i].busnr,
                                                mdsp_board[i].devnr,
                                                PCI_BASE_ADDRESS_0,
                                                &base)))
     {
      PRINTK("Failed to read base from PCI board. errno=%d\n",result);
      continue;
     }
    if((result=kernel_pcibios_read_config_byte(mdsp_board[i].busnr,
                                               mdsp_board[i].devnr,
                                               PCI_INTERRUPT_LINE,
                                               &mdsp_board[i].irq)))
     {
      PRINTK("Failed to read interrupt line from PCI board, errno=%d\n",result);
      continue;
     }
```

Quelltext 7: Auslesen der PCI-Konfiguration

```
    /* Enable Memory space access and parity error handling */
    kernel_pcibios_write_config_word(mdsp_board[i].busnr,
                                     mdsp_board[i].devnr,
                                     PCI_COMMAND,
                                     PCI_COMMAND_MEMORY|
                                     PCI_COMMAND_PARITY);
    /* Memory-mapping if necessary, otherwise register IO-space */
    if(base&PCI_BASE_ADDRESS_SPACE) /* Normal IO-Port */
     {
      DPRINTK("ioaddress is already io-mapped.\n");
      if(kernel_check_region(base,MDSP_MEM_SIZE))
       {
        PRINTK("Cannot access IO memory at 0x%x, skipping board.\n",
               (unsigned int)base);
        continue;
       }
      /* Register IO region  to /proc/ioports */
      base&=PCI_BASE_ADDRESS_IO_MASK;
      kernel_request_region(base,MDSP_MEM_SIZE,mdsp_name);
      mdsp_board[i].base=base;
      mdsp_board[i].basetype=PCI_BASE_ADDRESS_SPACE_IO;
     }
    else /* Memory-mapped, must be mapped into Kernel-Space */
     {
      DPRINTK("ioaddress is memory-mapped, mapping into kernel-space.\n");
      base &= PCI_BASE_ADDRESS_MEM_MASK;
      mdsp_board[i].base = (unsigned int) kernel_ioremap(base,MDSP_MEM_SIZE);
      mdsp_board[i].basetype=PCI_BASE_ADDRESS_SPACE_MEMORY;
     }
```

Quelltext 8: Einschalten des memory space access

Obwohl durch das PCI-Kommando `PCI_COMMAND_MEMORY` bereits die Art des Zugriffs festgelegt wurde, wurde aus Kompatibilitätsgründen mit möglichen zukünftigen

Karten der Serie 56000 noch einmal die von der Karte ausgelesene Basisadresse daraufhin überprüft, ob es sich um eine IO-Adresse oder eine Memorymapped-Adresse handelt. Dies wird durch die zwei höchsten Bits dieser Adresse signalisiert, wofür das Makro `PCI_BASE_ADDRESS_SPACE` als Maske dient. Für den Fall einer IO-Adresse wird auf die Register der Karte mit den Linux-Makros `inl()` und `outl()` (siehe auch 16 auf Seite 27) zugegriffen, andernfalls mit `readl()` und `writel()` über das gleiche Adreßschema, das auch bei der Adressierung von RAM verwendet wird.

Ein Problem ist noch zu überwinden: Die physikalischen Adressen („Hardware-Adressen") unterscheiden sich vom Adreßschema, das im Kernel-Space verwendet wird.[16] Hier bietet die Funktion `vremap()` Abhilfe, die den physikalischen Adreßbereich, der von der Karte verwendet wird, in den vom Kernel benutzten Adreßraum mit der angegebenen Größe einblendet.[17]

Für IO-adressierbare Karten entfällt dieses Remapping. Stattdessen wird hier der verwendete IO-Bereich durch die Prozedur `request_region()` als vom System beansprucht markiert und in `/proc/ioports` eingetragen.

```
/* Precalculate addresses for easier access in functions. */
mdsp_board[i].hctr=mdsp_board[i].base+HCTR;
mdsp_board[i].hstr=mdsp_board[i].base+HSTR;
mdsp_board[i].hcvr=mdsp_board[i].base+HCVR;
mdsp_board[i].htxr=mdsp_board[i].base+HTXR;
mdsp_board[i].hrxr=mdsp_board[i].base+HRXR;
mdsp_board[i].ready=1; /* Everything ok, flag board ready */
```

Quelltext 9: Eintrag der vorberechneten Registeradressen in `mdsp_info`

Zu der von der Karte gelieferten und in den Kernel-Space eingeblendeten Basisadresse müssen Offsets addiert werden, um die Adressen der von der Karte verwendeten Register zu erhalten. Um dies nicht in jeder Prozedur tun zu müssen, die auf die Register zugreift, werden die Werte vorberechnet in das `mdsp_board[]`-Strukturfeld eingetragen. Nachdem die Karte korrekt erkannt und konfiguriert wurde, wird sie in das `mdsp_board`-Strukturfeld als vom Treiber ansprechbar vermerkt, das `ready`-Flag wird gesetzt.

Die von der HI32-Schnittstelle des DSP56301 zur Verfügung gestellten Register sind ausführlich in [2] beschrieben. Hier sollen sie nur kurz angegeben werden (Tabelle 4).

Schließlich gibt das Modul eine Console-Meldung aus, in der die PCI-Informationen über jede erkannte DSP-Karte und ihre in den Kernel-Bereich eingeblendete Basis-

[16] Gleiches gilt für den User-Space, da jeder Prozeß auf einem Unix-System durch die Speicherverwaltung in einem eigenen virtuellen Adreßraum abläuft.

[17] Laut [2] können die von der HI32-Schnittstelle des DSP56301 benutzten Register in einem Adreßbereich von 64KB angesprochen werden, daher dieser Wert für `MDSP_MEM_SIZE`. Vom Treiber selbst werden nur die ersten 4 32-Bit Worte dieses Bereichs verwendet.

HCTR	Host side Control Register (Steuern der Datenübertragung zwischen DSP und der HI32-Schnittstelle)
HSTR	Host side Status Register (u.a. Information über den aktuellen Zustand des Transferregisters)
HCVR	Host side Command Vector Register (Register für Interruptmasken und Adressen von Interruptroutinen des DSP)
HTXR	Host side Transmit Register (6 32-Bit-Wort FIFO Host→DSP)
HRXR	Host side Receive Register (6 32-Bit-Wort FIFO DSP→Host, gleiche Adresse wie HTXR)

Tabelle 4: Register der HI32-Schnittstelle des DSP56301

adresse angezeigt werden. Nach Ablauf der äußeren Schleife wird zusätzlich die Anzahl aller gefundenen DSP-Boards ausgegeben.

```
    /* Print status of card to console */
    PRINTK("BUS: %d, DEVICE: 0x%x, CLASS: 0x%x, ADDRESS: 0x%x, IRQ: %d\n",
           (int)mdsp_board[i].busnr,(int)mdsp_board[i].devnr,
           (int)mdsp_board[i].class, (int)mdsp_board[i].base,
           (int)mdsp_board[i].irq);
  } /* for i < MDSP_MAX */
 PRINTK("Found %d Motorola DSP56xxx board%s.\n",i,i!=1?"s":"");
 return 0;
}
```

Quelltext 10: Kernel-Meldungen nach Abarbeiten vom `init_mdsp()`

Statt auf die Console umzuschalten, um die Kernel-Meldungen zu sehen, kann der Benutzer unter der X-Window Oberfläche das Programm `xconsole` starten. Die Kernel-Meldungen werden in dieses Fenster umgeleitet. Über den `syslogd` besteht außerdem die Möglichkeit, Kernel-Meldungen in beliebige Dateien, via E-Mail an Benutzer oder über das Netzwerk an einen auf einem Remote-System laufenden `syslogd` weiterzuleiten.

5.2.2 `cleanup_module()`

Die Prozedur `cleanup_module()` wird im modularen System des Linux-Kernels innerhalb eines Moduls aufgerufen, wenn das Modul aus dem Kernel entfernt und die vom Modul in Anspruch genommenen Ressourcen freigegeben werden sollen. Dies geschieht bei der Methode des automatischen Ladens und Entladens von Modulen (siehe Abschnitt 2 auf Seite 16), wenn kein Prozeß mehr auf die dem Modul zugehörige Devicedatei zugreift, wenn also der Referenz-Counter des Moduls auf 0 steht und der beim Aufruf des `kerneld` angegebene Timout verstrichen ist.

Wurde das Modul hingegen mit dem Kommando `modprobe` oder `insmod` in den Kernel eingebunden, so kann es nur manuell mit dem Kommando `rmmod Modulname` entfernt werden.

Mit dem Kommando `lsmod` kann untersucht werden, welche Module mit welcher Methode geladen wurden und aktiv sind.
Beispiel:

```
root@lisa #  lsmod
  Module:         #pages:  Used by:
  isofs               5           0
  mdsp                2           0  (autoclean)
  psaux               1           1  (autoclean)
  ne                  2           1  (autoclean)
  8390                2    [ne]   0  (autoclean)
```

Das `mdsp`-Modul wurde also durch den Kerneld automatisch durch einen Zugriff geladen, ist aber derzeit inaktiv. `psaux`, der Treiber für die PS2-Maus wurde automatisch geladen und ist in Benutzung, ebenso wie der Treiber für eine NE2000-kompatible Netzwerkkarte `ne`, der das Hilfsmodul `8390` nachgeladen hat. Das Modul `isofs`, das den Zugriff auf ISO9660-Dateisysteme (zumeist CD-Roms) ermöglicht, wurde jedoch manuell oder durch ein Script beim Systemstart geladen und kann nur mit `rmmod` wieder entfernt werden.

Die Prozedur aus `cleanup_module()` aus `mdsp.c` ist sehr einfach gehalten, da die PCI-Karte nicht explizit deaktiviert werden muß. Es müssen lediglich die als vom Treiber benutzt markierten Speicherbereiche, die in `init_module()` und der Prozedur `init_mdsp()` festgestellt und initialisiert wurden, freigegeben werden.

Das Device mit der major number 101 und dem Namen `mdsp` wird mit der Kernel-Funktion `unregister_chrdev()` abgemeldet.

Das Modul wird aus dem Speicher entfernt. Erst beim nächsten Zugriff oder Aufruf von `insmod mdsp` wird das Modul wieder geladen und vollständig neu initialisiert (siehe 5.2.1 auf Seite 16).

5.2.3 open_mdsp()

Die Funktion `open_mdsp()` wird vom `mdsp`-Modul ausgeführt, wenn ein Prozeß mittels der Funktion `open()` aus der Standard C-Bibliothek auf eine Gerätedatei mit der major number 101 zugreift. Ist das `mdsp`-Modul noch nicht geladen, so werden zuerst die Initialisierungsprozeduren aus Abschnitt 5.2.1 ausgeführt. Erst bei deren Erfolg wird in die eigentliche Funktion `open_mdsp()` verzweigt, deren Adresse in

```
/* Cleanup before unloading module, release io or unmap memory space */
static void cleanup_mdsp(void)
{
 int i;
 for(i=0;i<MDSP_MAX;i++)
  {
   if(mdsp_board[i].ready)
    {
     if(mdsp_board[i].basetype==PCI_BASE_ADDRESS_SPACE_IO)
       kernel_release_region((unsigned int)mdsp_board[i].base,MDSP_MEM_SIZE);
     else
       kernel_iounmap((void *)mdsp_board[i].base);
    }
   mdsp_board[i].ready=0;
  }
}

void cleanup_module(void)
{
 kernel_unregister_chrdev(mdsp_major,mdsp_name);
 cleanup_mdsp();
 PRINTK("Unregistered Device %s, major %d.\n", mdsp_name, mdsp_major);
}
```

Quelltext 11: `cleanup_module()`

den Initialisierungsroutinen in der Systemstruktur `file_operations` an den Kernel übergeben wurde.

Wie in `/usr/include/linux/fs.h` deklariert, werden der Funktion vom Kernel als Parameter ein Zeiger auf eine Struktur vom Typ `inode` und ein Zeiger auf eine Struktur vom Typ `file` übergeben, also bereits die Ergebnisse der Unix-Prozedur `stat()` aus der C-Bibliothek und nicht die Parameter der aufrufenden Funktion `open()`, nämlich der Dateiname und die Art des Zugriffs.

Wenn der aufrufende Prozeß versucht, eine nicht-existente Datei zu öffnen, oder eine Datei, die keine durch das `mdsp`-Modul verwaltete Gerätedatei ist, wird überhaupt nicht in das Modul verzweigt, sondern der Benutzerprozeß erhält sofort vom Kernel einen Fehlerwert als Rückgabe. Nur wenn die Datei die für den Benutzerprozeß passend gesetzten Berechtigungen besitzt, wird die Funktion `open_mdsp()` überhaupt erreicht.

In der Funktion selbst muß, da der Treiber mehrere gleichzeitig arbeitende Geräte unterstützt, über die minor device number das zu der Datei passende Gerät gefunden werden.[18]

`i_rdev` aus der `inode`-Struktur[19] enthält die minor number der Gerätedatei in den unteren 8 Bit. Das Makro `MINOR()` aus `/usr/include/linux/kdev_t.h` wertet diese aus. Der Index der zur minor number gehörenden DSP-Karte im Feld

[18] Siehe auch Tabelle 2 auf Seite 12.

[19] Definition in `/usr/include/linux/fs.h`.

```
static int open_mdsp(struct inode *inode, struct file *file)
{
 int minor=MINOR(inode->i_rdev),boardnr=minor-1;
 if((minor==0)||(minor>0&&minor<=MDSP_MAX&&mdsp_board[boardnr].ready))
  {
   MOD_INC_USE_COUNT; /* Increment in-use counter */
```

Quelltext 12: `open_mdsp()`

`mdsp_board[]` ergibt sich aus der minor number minus 1. Das Gerät mit der minor number 0 dient der Ausgabe von Statusinformationen und wird in Quelltext 13 behandelt.

Existiert eine zur minor number passende, arbeitende DSP-Karte, so wird der In-Use-Zähler des Moduls mit dem in `/usr/include/linux/module.h` definierten Makro `MOD_INC_USE_COUNT` um eins erhöht, um ein Entladen des Moduls während laufender Zugriffe zu verhindern.

```
   if(minor==0)
    {
     if(status_buf||(status_buf=(char *)vmalloc(MAX_STATUS))==NULL) return -EIO;
     status_len=status_ptr=0;
     init_status();
    }
```

Quelltext 13: Sonderbehandlung der Status-Datei `/dev/mdspstat`

Da die DSP-Karten schon in der Prozedur `init_module()` initialisiert und in `mdsp_board[]` katalogisiert wurden, ist nur im Falle der Gerätedatei mit der minor number 0, `/dev/mdspstat`, eine spezielle Behandlung innerhalb von `open_mdsp()` erforderlich. Für die aus dieser Datei lesbaren Informationen muß ein Puffer angefordert werden. Die Statusinformationen über alle Karten werden in der Prozedur `init_status()`, die größtenteils aus dem Kernel-Modul für Soundkarten übernommen wurde, in den Statuspuffer übertragen und können vom Anwenderprogramm ausgelesen werden.

```
   DPRINTK("Minor %d opened.\n",minor);
   return 0;
  }
 return -ENODEV;
}
```

Quelltext 14: Rückgabewerte der Funktion `open_mdsp`

Wenn das Modul mit der Compiler-Option `-DDEBUG` übersetzt wurde, wird als Console-Meldung die minor number der durch `open()` geöffneten Gerätedatei ausgegeben.

Auch bei dieser Funktion bedeutet ein Rückgabewert von 0 Erfolg, ein Wert ungleich 0 einen aufgetretenen Fehler. Wird 0 an die aufrufende Prozedur (das virtuelle Filesystem des Kernels) zurückgegeben, gibt diese wiederum einen gültigen Dateideskriptor an das Anwenderprogramm zurück.

Ein explizites Locking wird nicht ausgeführt, d.h. es können mehrere Anwenderprozesse gleichzeitig sowohl lesend als auch schreibend auf die Gerätedatei zugreifen. Dies bringt den Vorteil mit sich, daß ein Duplex-Betrieb möglich ist, daß also ein Prozeß Daten zur DSP-Karte übertragen kann, während ein anderer Prozeß Daten von der Karte liest. Außerdem können mehrere Prozesse quasi gleichzeitig die Kontroll- und Statusregister einer Karte verändern, oder abwechselnd Daten aus dem HI32-Transferregister auslesen.[20] Der Nachteil dieser Methode ist, daß sich der Anwender darüber im klaren sein muß, daß sich seine Prozesse durch gegenseitiges Überschreiben von Registern der DSP-Karte beeinträchtigen können. Er muß sich also selbst darum kümmern, daß sich seine Prozesse kooperativ verhalten.[21]

5.2.4 `release_mdsp()`

`release_mdsp()` ist das Gegenstück zu `open_mdsp()`. `release_mdsp()` wird im `mdsp`-Modul aufgerufen, wenn der Anwenderprozeß eine zuvor mit `open()` geöffnete Gerätedatei mit der major device number 101 wieder mit `close()` schließt.

Auch hier muß zunächst mit Hilfe der minor device number festgestellt werden, ob es sich um eine gültige DSP-Karte handelt, von der sich der Anwenderprozeß abmelden möchte. Ist dies der Fall, wird der In-Use Zähler um eins erniedrigt.

Wieder erhält das Gerät mit der minor device number 0 eine Sonderbehandlung: Der zuvor bei `open_mdsp()` angeforderte Pufferspeicher für Statusinformationen wird freigegeben.

[20] Ein exakt gleichzeitiger Zugriff zweier Prozesse auf ein Register der DSP-Karte kann nicht stattfinden, da der PCI-Controller Zugriffe auf das Bussystem nur sequentiell erlaubt, auch bei Multiprozessorsystemen.

[21] Ein Beispiel – und gleichzeitig Gegenbeispiel – hierfür ist das Utility `mdsptest` (Abschnitt 6.2 auf Seite 45), das auch problemlos mehrmals gleichzeitig gestartet werden kann. Der Grund hierfür ist, daß anhand der abgefragten Status-Flags überprüft wird, ob gültige Daten vorliegen oder Daten übertragen werden können. Außerdem werden die nur zum Zweck eines Benchmark übertragenen Daten bei diesem Programm nicht auf Konsiztenz überprüft oder weiterverarbeitet. Der gemessene Durchsatz bei der Datenübertragung verhält sich natürlich entsprechend der Anzahl der gleichzeitig gestarteten Programme. Da bei `mdsptest` allerdings kein eigener Locking-Mechanismus und auch keine Sicherheits-Abfrage verwendet wird, kann das Programm andere Prozesse, die die gleiche DSP-Karte benutzen, störend beeinflussen.

```
static void release_mdsp(struct inode *inode, struct file *file)
{
 int minor=MINOR(inode->i_rdev),boardnr=minor-1;
 if((minor==0)||(minor>0&&minor<=MDSP_MAX&&mdsp_board[boardnr].ready))
  {
   if(minor==0 && status_buf) {vfree(status_buf); status_buf=NULL; }
   MOD_DEC_USE_COUNT; /* Increment in-use counter */
   DPRINTK("Minor %d closed.\n",minor);
  }
}
```

Quelltext 15: `release_mdsp()`

5.2.5 `read_mdsp()`

Um die vom Treiber dem Host-Betriebssystem zur Verfügung gestellten Routinen zum Lesen und Schreiben von Registern der DSP-Karte realisieren zu können, werden zunächst zwei elementare Funktionen zum Lesen und Schreiben von Registern benötigt (Quelltexte 16 und 17).

```
/* Basic register read/write functions. Though the DSP56301 currently      */
/* uses memory-mapped access, other cards might be mapped into io-space,   */
/* so both is supported.                                                   */
static __inline void mdsp_write_reg(unsigned int addr,__u32 val,unsigned int type)
{
 if(type==PCI_BASE_ADDRESS_SPACE_IO) kernel_outl(val,addr);
 else writel(val,addr);
}
```

Quelltext 16: `mdsp_write_reg()`

```
static __inline __u32 mdsp_read_reg(unsigned int addr,unsigned int type)
{
 return (type==PCI_BASE_ADDRESS_SPACE_IO)?kernel_inl(addr)
                                         :readl(addr);
}
```

Quelltext 17: `mdsp_read_reg()`

Bei beiden Prozeduren wird die in `init_mdsp()` festgestellte Art des Zugriffs auf Register der DSP-Karte unterschieden, was durch den Parameter `type` angegeben wird. Für den Fall IO-basierter Register wird die Kernel-Funktion `inl(unsigned int addresspointer)` bzw. `outl(__u32 value, unsigned int addresspointer)` verwendet, für den Fall von speicheradressierbaren Registern hingegen die Makros `readl(unsigned int addresspointer)` und `writel(__u32 value, unsigned int addresspointer)`, die genausogut durch einfache Zeigerdereferenzierung ersetzt werden können.

`__u32` ist ein architekturunabhängier *32-Bit unsigned* Datentyp, auf Intel-Systemen gleichbedeutend mit `unsigned int`.

Das Datentransferregister-FIFO HTXR bzw. HRXR muß gesondert behandelt werden, da ein lesender Zugriff auf dieses Register ohne Verfügbarkeit von Daten im FIFO sowie ein schreibender Zugriff bei vollen FIFO einen Blockadezustand im PCI-Bus auslöst.[22] Ein solches Verhalten ist in einem Multitasking-System nicht akzeptabel, insofern dieser Fehler von einem Anwenderprogramm bei direktem Zugriff auf die HI32-Schnittstelle jederzeit unbeabsichtig herbeigeführt werden könnte. Daher muß vor jedem Datentransfer über das Transferregister eine Abfrage des Statusregisters HSTR erfolgen. Ein Lesezugriff auf das HRXR-Register darf nur erfolgen, wenn Daten zum Lesen bereit stehen, was durch das Flag HRRQ (Bit 2 gesetzt) im Statusregister signalisiert wird. Ein Schreibzugriff auf das HTXR-Register darf gleichfalls nur erfolgen, wenn das Flag HTRQ (Bit 1) im Statusregister gesetzt ist. Das `mdsp`-Modul muß also bei Lese- oder Schreibzugriffen auf das Transferregister immer zunächst das Statusregister auf die entsprechenden Flags überprüfen und ggf. bis zum Eintreffen des gewünschten Zustands blockieren oder eine entsprechende Fehlermeldung zurückliefern.

Da die DSP-Karte im PCI-Modus keine Interrupts des Hostsystems verwendet,[23] muß auf die Verfügbarkeit des Transferregister mittels *Polling* gewartet werden. Dies ist grundsätzlich kein Geschwindigkeitsnachteil gegenüber einer Interruptroutine, da in beiden Fällen eine Änderung im Statusregister sofort entdeckt wird. Der Linux-Treiber für die parallele Schnittstelle arbeitet ebenfalls nach dem Polling-Verfahren. Interruptroutinen werden jedoch in multitaskingfähigen Betriebssystemen bevorzugt eingesetzt, um keine Prozessorleistung für das Warten auf ein Ereignis einsetzen zu müssen.

```
/*************************************************************************\
* poll() waits for the given flag to be set in the given register.        *
* returns 0 on success. There is no timeout limit, though the polling may be  *
* terminated on reception of any non-blocked signal.                      *
\*************************************************************************/
static int poll(unsigned int address,__u32 flag,unsigned int type)
{
 for(;;)
  {
   __u32 status=mdsp_read_reg(address,type);
   unsigned int sig;
   if(status&flag) break;
```

Quelltext 18: `poll()`

Für das blockierende Warten auf ein gesetztes Status-Flag wurde die Prozedur `poll()` implementiert, die konform mit dem Multitasking unter Linux während des Wartens den Scheduler aufruft und so gewährleistet, daß andere Prozesse in der Zwischenzeit abgearbeitet werden können.

[22]Dieses Verhalten wird in [2] angesprochen und konnte durch Tests nachvollzogen werden.

[23]Eine Ausnahme sind Interrupts des Typs `INTA#`, die von einem auf dem DSP ablaufenden Programm explizit generiert werden müssen.

In einer Endlosschleife wird das unter der Adresse `address` liegende Register ausgelesen und auf das gesetzte Flag `flag` hin überprüft. Ist es gesetzt, so kehrt `poll()` sofort mit dem Wert 0 als Zeichen für Erfolg zurück.

```
   /* Did we receive any non-blocked signals? (^C etc.) */
   sig=kernel_var_current->signal&~kernel_var_current->blocked;
   if(sig)
    { /* Handle SIGSTOP etc. */
     DPRINTK("Command aborted by signal 0x%x\n",sig);
     return -ERESTARTSYS;
    }
```

Quelltext 19: `poll()` – Reagieren auf Signale

Ist in der Signalmaske ein nicht vom Prozeß als blockiert vermerktes Signalbit gesetzt, so wird der Fehlerwert `-ERESTARTSYS` zurückgegeben. Die aufrufende Prozedur kann entscheiden, wie in solche einem Fall weiter zu verfahren ist.

Die Signalmaske eines Prozesses wird durch an den Prozeß mit dem Systemaufruf `kill()` verschickte Signale verändert. Beispielsweise wird solch ein Signal ausgelöst, wenn der Vaterprozeß (die Benutzer-Shell) nach einem Tastaturinterrupt eine Kommandofolge wie *Control-C* erkennt und dem Kindprozeß daraufhin das Signal `SIGINT` schickt. Wird dieses Signal nicht ausgewertet und behandelt, während sich der Prozeß im Kernel-Space (`mdsp`-Modul) aufhält, so ist er nicht unterbrechbar. Dies kann dazu führen, daß Prozesse bei blockierenden Systemaufrufen nicht mehr beendet werden können und angeforderte Ressourcen nicht mehr freigeben.[24] Da das Überprüfen der Signalmaske des aktuellen Prozesses kaum Rechenzeit in Anspruch nimmt und „Programmhänger" vermieden werden können, wurde der Erkennung und Bearbeitung von Signalen Rechnung getragen. Außerdem besteht so die Möglichkeit, den Warteprozeß mit dem Signal `SIGSTOP` anzuhalten und später mit dem Signal `SIGCONT` weiterlaufen zu lassen. Die Kontrolle über den Prozeß liegt also beim Benutzer.

```
#ifndef SLOW
   /* Be nice to other processes */
   if(kernel_var_need_resched) kernel_schedule();
#endif
  }
 return 0;
}
```

Quelltext 20: Aufruf des Schedulers in `poll()`

Schließlich muß dafür gesorgt werden, daß das unerwünschte „Busy Waiting" vermieden wird. Nach jeder erfolglosen Überprüfung des Registers auf das angegebene Flag

[24]Daher gilt es unter Linux als guter Programmierstil, bei Implementation von Systemaufrufen, die längere Zeit blockieren können, Signale abzufangen, eine Signalbehandlungsroutine auszuführen und ggf. zum aufrufenden Prozeß zurückzukehren.

wird – nur bei Bedarf, der durch die Kernel-Variable `need_resched` signalisiert wird – der Prozeß-Scheduler aufgerufen und das Abarbeiten wichtiger Systemtasks ermöglicht.

Bei hoher Systemlast mag diese explizite Abgabe der CPU an andere Prozesse unerwünscht sein, vor allem, wenn das gewünschte Ziel aus einem Standalone-System besteht, dessen einzige Aufgabe die Einbettung einer DSP-Karte ist.[25] Für diesen Sonderfall kann mit der Übersetzungs-Option `-DSLOW` der Scheduler-Aufruf unterbunden werden. Dies ist jedoch nur in wenigen Ausnahmefällen sinnvoll, da hierdurch auch wichtige Systemtasks in ihrer Funktion gestört werden, beispielsweise der TCP/IP Stack oder die Benutzer-Shell, in die der blockierende Prozeß eingebettet ist. Dies kann dazu führen, daß beim Remote-Betrieb das Hostsystem keine Antwort auf Komandos mehr liefert, sich also auch der blockierende Prozeß nicht mehr abbrechen läßt. Die `SLOW`-Option sollte also generell vermieden werden.[26]

Nachdem die für die Kommunikation zwischen der HI32-Schnittstelle des DSP56301 und dem Hostsystem elementaren Funktionen betrachtet wurden, kann nun auf die Funktion `read_mdsp()` eingegangen werden.

```
static int read_mdsp(struct inode *inode, struct file *file, char *buffer,
                     int count)
{
 int minor=MINOR(inode->i_rdev),boardnr=minor-1;
 if((minor==0)||(minor>0&&minor<=MDSP_MAX&&mdsp_board[boardnr].ready))
  {
   int i,result;
```

Quelltext 21: `read_mdsp()` (1)

Zunächst wird, wie in `open_mdsp()`, auf das Vorhandensein einer zur minor number der vom Anwenderprogramm angesprochenen Gerätedatei passenden DSP-Karte überprüft. `read_mdsp()` erhält als Übergabeparameter zusätzlich zu Zeigern auf die zur Gerätedatei gehörende `inode` und `file`-Struktur die in der C-Funktion `read(int fd, void *buffer, size_t count)` übergebene Adresse eines Puffers im Nutzeradreßraum und die Anzahl der zu lesenden Bytes.

```
   if(count<=0) return -EINVAL;
   if((result=kernel_verify_area(VERIFY_WRITE,buffer,count))!=0) return result;
```

Quelltext 22: `read_mdsp()` (2)

Ist die Anzahl der zu lesenden Bytes vom Anwenderprogramm fehlerhaft als ein Wert kleiner oder gleich 0 angegeben worden, so kehrt `read_mdsp()` sofort

[25] Allerdings ist auf einem solchen System kaum eine hohe CPU-Last durch andere Prozesse als diejenigen, die auf die DSP-Karte zugreifen, zu erwarten.

[26] Tests haben gezeigt, daß bis zu einer System-Load von $4,0$ keinerlei Performanceeinbußen für die Datenübertragung zwischen DSP und Host zu verzeichnen sind, obwohl der Scheduler aufgerufen wird.

mit dem Fehlerwert `-EINVAL` zurück. Andernfalls wird mit der Kernel-Funktion `verify_area()` überprüft, ob der vom aufrufenden Prozeß spezifizierte Speicherbereich im Nutzeradreßraum existiert und für den Kernel schreibbar ist.

```
    if(minor==0)
     { /* Write status information for all cards */
      return read_status(buffer,count);
     }
    else
     {
```

Quelltext 23: `read_mdsp()` (3)

Das Gerät mit der minor number 0 erhält – wie in den Abschnitten 5.2.3 und 5.2.4 erwähnt – eine Sonderbehandlung. Hier werden die in `open_mdsp()` in den Statuspuffer geschriebenen Informationen mit maximal `count` Bytes in den vom Anwenderprogramm zur Verfügung gestellten Puffer `buffer` geschrieben.

```
      if(!mdsp_board[boardnr].ready) return -ENODEV;
      /* Count must be a multiple of sizeof(__u32) */
      count &= ~(MDSP_INT_LEN-1);
      DPRINTK("Reading %d 32-bit words from DSP.\n",count/sizeof(int));
      for(i=0; i<count; i+=MDSP_INT_LEN) /* 32-bit values */
       {
        __u32 val;
        /* Wait for status flag, then read. HSTR_HRRQ is set by the DSP       */
        /* automatically after write operations to DTXR.                      */
        result=poll(mdsp_board[boardnr].hstr,HSTR_HRRQ,
                    mdsp_board[boardnr].basetype);
        if(result) return i?i:result;    /* return bytes read so far or error */
        val=mdsp_read_reg(mdsp_board[boardnr].hrxr,
                          mdsp_board[boardnr].basetype);
```

Quelltext 24: `read_mdsp()` (4)

Quelltext 24 zeigt den Ausschnitt aus `read_mdsp()`, der für das Auslesen des Transferregister-FIFO HRXR verantwortlich ist.

Der vom aufrufenden Prozeß übergebene Counter `count` in Bytes muß auf einen durch die Größe eines 32-Bit Wortes teilbaren Wert reduziert werden, da sich aus dem Transferregister nur vollständige 32-Bit Werte lesen lassen.

In einer Schleife werden so viele Worte aus dem Transferregister gelesen, bis die geforderte Bytezahl erreicht ist.

Wurde `poll()` durch ein Signal unterbrochen, wird die bis zu diesem Zeitpunkt gelesene Bytezahl zurückgegeben, ansonsten der Fehlerstatus `-ERESTARTSYS`, den `poll()` liefert. Dadurch wird verhindert, daß die bis zum Eintreffen des Signals gelesenen Daten vom aufrufenden Prozeß ignoriert werden und quasi „verloren gehen“.

Schließlich wird der aus dem Transferregister gelesene 32-Bit Wert mit Hilfe des Makros `put_user()` in den Benutzeradreßraum des aufrufenden Prozesses ko-

```
        kernel_put_user(val,(__u32 *)&buffer[i]);
       }
     return count;
    }
  }
 return -ENODEV;
}
```

Quelltext 25: `read_mdsp()` (5)

piert. Dieses Makro aus `/usr/include/asm/segment.h` sorgt sowohl für die korrekte Behandlung der Adreß-Datentypen der Übergabeparameter *long*, *short* und *char* als auch für die Übertragung aus dem Kernel- in den User-Space. Alternativ kann unter der Angabe der Größe des zu kopierenden Datums in Bytes die Kernel-Prozedur `memcpy_tofs()` verwendet werden, die allerdings für das Kopieren ganzer Speicherbereiche vorgesehen und damit für die Übergabe eines einzigen 32-Bit Wertes ineffizient ist.

Wurde beim Aufruf von `mdsp_read()` die Inode einer Gerätedatei übergeben, der keine DSP-Karte zugeordnet ist, so kehrt die Funktion mit dem Fehler `-ENODEV` zurück, andernfalls wird die tatsächlich aus dem Transferregister gelesene Anzahl von Bytes an den aufrufenden Prozeß übergeben.[27]

5.2.6 `write_mdsp()`

`write_mdsp()` stellt das Gegenstück zu `read_mdsp()` dar. Hier wird die vom aufrufenden Prozeß angegebene Anzahl von 32-Bit Werten[28] in das HTXR Register, das im Falle des DSP56301 unter der gleichen virtuellen Adresse wie das Leseregister HRXR angesprochen wird, geschrieben. `/dev/mdspstat` mit der minor number 0 erfährt hier nicht die aus Abschnitt 5.2.5 bekannte Sonderbehandlung, da die Generierung der Statusinformationen dem Kernel-Modul vorbehalten bleibt und der „Inhalt“ dieser virtuellen Datei nicht vom Benutzer überschreibbar sein soll.

Mit der aus Abschnitt 5.2.5 auf Seite 28 beschriebenen Hilfsfunktion `poll()` wird über das Flag `HSTR_HTRQ` im Status-Register der DSP-Karte auf die Verfügbarkeit einer Schreiboperation auf das `HTXR`-Register gewartet. Dieses Register der HI32-Schnittstelle des DSP56301 ist, wie das `HRXR`-Register, ein 6 32-Bit Worte großes FIFO. Das gesetzte `HSTR_HTRQ`-Flag signalisiert, daß das FIFO nicht vollständig gefüllt ist, daß also für mindestens einen 32-Bit Wert Platz ist.[29]

[27] `read_mdsp()` stellt also, genau wie `write_mdsp()` im nächsten Abschnitt eine Ausnahme dar, da die meisten Kernelprozeduren und -funktionen nur im Fehlerfall einen Wert ungleich 0 zurückgeben

[28] `count` wieder als Bytezahl

[29] Mit dem Fetch/Prefetch-Flag `SFT` im Kontrollregister `HCTR` kann dieses Verhalten beeinflußt werden, siehe [2]. Um Software-Polling zu ermöglichen, wie es vom `mdsp`-Modul verwendet wird, muß

```
static int write_mdsp(struct inode *inode, struct file *file,
                      const char *buffer, int count)
{
 int minor=MINOR(inode->i_rdev),boardnr=minor-1;
 if(minor>0&&minor<=MDSP_MAX&&mdsp_board[boardnr].ready)
  {
   int i,result;
   if((result=kernel_verify_area(VERIFY_READ, buffer, count))!=0) return result;
   if(count<sizeof(long)) return -EINVAL;
   /* Count must be a multiple of sizeof(__u32). */
   count &= ~(MDSP_INT_LEN-1);
   DPRINTK("Writing %d 32-bit words to DSP.\n",count/MDSP_INT_LEN);
   for(i=0; i<count; i+=MDSP_INT_LEN) /* 32-bit words */
    {
     __u32 val;
     val=kernel_get_user((__u32 *)&buffer[i]);
     /* Wait for status flag in HSTR, then write. HTRQ is set by the DSP */
     /* automatically if there is space in FIFO for another 32-bit word. */
     result=poll(mdsp_board[boardnr].hstr,HSTR_HTRQ,
                 mdsp_board[boardnr].basetype);
     if(result) return i?i:result;  /* return bytes read so far or error */
     mdsp_write_reg(mdsp_board[boardnr].htxr,val,
                    mdsp_board[boardnr].basetype);
    }
   return count;
  }
 return -ENODEV;
}
```

Quelltext 26: `write_mdsp()`

`write_mdsp()` kehrt mit der Anzahl der geschriebenen Bytes zur aufrufenden Prozedur zurück. Wurde `poll()` durch ein Signal unterbrochen, so wird die Anzahl der bis zu diesem Zeitpunkt geschriebenen Bytes, bzw. wenn noch keine Transaktion stattgefunden hat, der Fehlerwert `-ERESTARTSYS` zurückgegeben.

5.2.7 `ioctl_mdsp()`

Mit den Standard-Operationen `open()`, `close()`, `read()` und `write()`, deren Unterstützung im `mdsp`-Modul in den vorigen Abschnitten diskutiert wurde, läßt sich ein Datentransfer zwischen der DSP-Karte und dem Host-System realisieren. Zur Steuerung des DSP oder zur direkten Abfrage von Registern der Karte sind diese Funktionen und Prozeduren jedoch nicht ausreichend.

Der Systemaufruf `ioctl()`[30], der in `/usr/include/sys/ioctl.h` deklariert wird, ermöglicht es auf einem Unix-System, weiter gehende Operationen auf Dateien, insbesondere Gerätedateien, auszuführen. So können beispielsweise Mixer-Einstellungen einer Soundkarte oder Änderungen in der Konfiguration einer Netzwerkkarte über `ioctl()`-Aufrufe ausgeführt werden.

das `SFT`-Flag, wie per Default vorgegeben, auf 0 gesetzt sein.

[30] „IO-Control“

Um einem Anwenderprogramm den direkten Zugriff auf die HI32-Schnittstelle des DSP56301 zu ermöglichen, wurde eine `ioctl()`-Schnittstelle für alle Register dieser Karte und das PCI-Kontrollregister implementiert.

`int ioctl(int d, int request, ...)` ist unter Linux ein Systemaufruf mit variabler Parameterzahl. Als erster Parameter wird der von `open()` zurückgelieferte Dateideskriptor der Gerätedatei übergeben.[31]

Der Parameter `request` bezeichnet ein an den Kernel-Treiber übergebenes Kommando, das sich, wie in [5] beschrieben, aus der Art des Zugriffs, der Parametergröße, einer Treiberkennung und einer Befehlsnummer zusammensetzt.

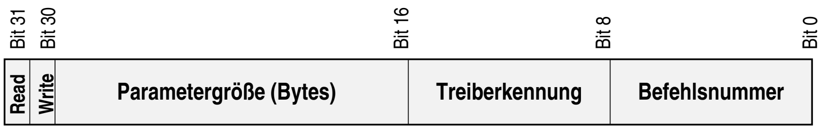

Abbildung 1: Aufbau eines `ioctl()`-Kommandos

Zur Generierung dieses Kommandos sind in `/usr/include/asm/ioctl.h` Makros definiert, die in `mdsp.h` verwendet werden.

```
/*************************************************************************\
*                       mdsp driver IOCTL Commands                         *
* param 3 is _address_ of unsigned long value to be read to or to be written. *
\*************************************************************************/
/* Read/Write Status Register        */
#define MDSPCTL_READ_HSTR  _IOR(MDSP_MAJOR,'s',unsigned long)
#define MDSPCTL_WRITE_HSTR _IOW(MDSP_MAJOR,'s',unsigned long)
/* Read/Write Control Register       */
#define MDSPCTL_READ_HCTR  _IOR(MDSP_MAJOR,'c',unsigned long)
#define MDSPCTL_WRITE_HCTR _IOW(MDSP_MAJOR,'c',unsigned long)
/* Read/Write Receive FIFO Register  */
#define MDSPCTL_READ_HRXR  _IOR(MDSP_MAJOR,'r',unsigned long)
#define MDSPCTL_WRITE_HRXR _IOW(MDSP_MAJOR,'r',unsigned long)
/* Read/Write Transmit FIFO Register */
#define MDSPCTL_READ_HTXR  _IOR(MDSP_MAJOR,'t',unsigned long)
#define MDSPCTL_WRITE_HTXR _IOW(MDSP_MAJOR,'t',unsigned long)
/* Read/Write Host Vector Register   */
#define MDSPCTL_READ_HCVR  _IOR(MDSP_MAJOR,'i',unsigned long)
#define MDSPCTL_WRITE_HCVR _IOW(MDSP_MAJOR,'i',unsigned long)
/* Read/Write PCI Command and Status Register */
#define MDSPCTL_READ_CCMR  _IOR(MDSP_MAJOR,'p',unsigned long)
#define MDSPCTL_WRITE_CCMR _IOW(MDSP_MAJOR,'p',unsigned long)
```

Quelltext 27: `ioctl()`-Kommandodefinitionen aus `mdsp.h`

Der dritte Parameter von `ioctl()` wird vom `mdsp`-Modul als Adresse eines Datums vom Typ `unsigned long` interpretiert. Bei schreibenden `ioctl()`-Kommandos

[31] Alternativ kann bei der ebenfalls in der Standard C-Bibliothek verwendeten gepufferten Methode mit `fileno()` der Dateideskriptor aus der `FILE`-Struktur, deren Adresse von `fopen()` zurückgegeben wird, ermittelt werden.

befindet sich dort der Wert, der in ein Register der DSP-Karte übernommen werden soll. Bei lesenden Kommandos wird hier der ausgelesene Wert abgelegt.

Ein Programmfragment aus dem unter Abschnitt 6.2 auf Seite 45 beschriebenen Utility `mdsptest` soll als Beispiel für die Benutzung eines `ioctl()`-Aufrufes dienen.

```
unsigned long val;
if(ioctl(fd,MDSPCTL_READ_HSTR,&val))
 {
  fprintf(stderr,"ioctl() failed.");
  close(fd);
  return 4;
 }
printf("Status register: 0x%lx\n",val);
```

Quelltext 28: Beispiel für den Aufruf eines `mdsp-ioctl()`

Hier wird der Inhalt des Statusregisters einer DSP-Karte ausgelesen und als Hexadezimalwert ausgegeben. `fd` ist der Dateideskriptor der zuvor mit `open()` geöffneten Gerätedatei, die mit der DSP-Karte assoziiert ist. `ioctl()` gibt bei erfolgeicher Ausführung 0 zurück, bei einem auftretenden Fehler den vom zugehörigen Treiber übergebenen Fehlerwert.

Quelltext 29 zeigt den Beginn des im `mdsp`-Modul implementierten `ioctl()`-Interface.

```
static int ioctl_mdsp(struct inode *inode, struct file *file, unsigned int cmd,
                      unsigned long param)
{
 int minor=MINOR(inode->i_rdev),boardnr=minor-1,result;
 if(minor==0||_IOC_TYPE(cmd)!=MDSP_MAJOR) return -EINVAL;
 if(minor>0&&minor<=MDSP_MAX&&mdsp_board[boardnr].ready)
  {
```

Quelltext 29: `ioctl_mdsp()` (1)

Nach der aus den vorigen Abschnitten bekannten Überprüfung auf Operabilität der angesprochenen DSP-Karte[32] und Überprüfung auf die `mdsp` major number 101 im Treiberkennungsfeld wird das mit dem `ioctl()`-Aufruf übergebene Kommando ausgewertet und dementsprechend die Adresse des zu lesenden/schreibenden Registers der DSP-Karte festgelegt.

Wird das Datentransferregister `HRXR` bzw. `HTXR` angesprochen, muß zunächst das FIFO auf Lesbarkeit bzw. Schreibbarkeit eines Wertes untersucht werden. Ist im Statusregister das entsprechende Flag nicht gesetzt, so blockiert der Versuch, auf dieses Register der DSP56301ADM zuzugreifen, den PCI-Bus bis zum System-Reset. Daher

[32] Im Falle von `/dev/mdspstat` ist die Ausführung von `ioctl()` nicht sinnvoll, deshalb wird für die minor number 0 ein entsprechender Fehlercode zurückgegeben.

```
    __u32 val,oldval;
    __u32 status;
    unsigned int address=0;
    unsigned int type=mdsp_board[boardnr].basetype;
    int pcicommand=0;
    switch(_IOC_NR(cmd))
     {
      case 'c': address=mdsp_board[boardnr].hctr; break;
      case 's': address=mdsp_board[boardnr].hstr; break;
      case 'r': status=mdsp_read_reg(mdsp_board[boardnr].hstr,type);
                if(!(status&HSTR_HRRQ)) return -EAGAIN;
                address=mdsp_board[boardnr].hrxr;
                break;
      case 't': status=mdsp_read_reg(mdsp_board[boardnr].hstr,type);
                if(!(status&HSTR_HTRQ)) return -EAGAIN;
                address=mdsp_board[boardnr].htxr;
                break;
      case 'i': address=mdsp_board[boardnr].hcvr; break;
               break;
      case 'i': address=mdsp_board[boardnr].hcvr; break;
      case 'p': pcicommand=1; break;
      default:  return -EINVAL;
     }
```

Quelltext 30: `ioctl_mdsp()` (2)

wird hier sofort der Fehlerwert `-EAGAIN` zurückgeliefert, welcher unter Unix bei einem nichtblockierenden Systemaufruf signalisiert, daß die Ausführung zu diesem Zeitpunkt nicht möglich ist. Mit Hilfe der in `mdsp.h` definierten `ioctl()`-Kommandos `MDSPCTL_READ_HRXR` und `MDSPCTL_WRITE_HTXR` kann also das nichtblockierende Äquivalent der `read()` und `write()`-Funktion im Anwenderprogramm realisiert werden.[33]

Je nach gesetztem Lese- oder Schreibflag im übergebenen Kommando muß das zugehörige HI32-Register oder das PCI-Kommando-/Statusregister der DSP-Karte gelesen oder geschrieben werden.

Die mit dem dritten Parameter des `ioctl()`-Aufrufes übergebene Adresse wird vor dem Auslesen aus dem Benutzeradreßraum bzw. Schreiben in den Benutzeradreßraum mit der aus Abschnitt 5.2.5 und 5.2.6 bekannten Kernel-Funktion `verify_area()` auf die Verfügbarkeit dieses Zugriffs untersucht.

Quelltext 31 macht deutlich, daß auch `ioctl()`-Kommandos, die sowohl lesend als auch schreibend auf ein Register der DSP-Karte zugreifen, erlaubt sind. In diesem Fall wird zuerst ein Wert aus dem Register gelesen und in der 32-Bit Variablen `oldval` gespeichert, danach wird ein Wert aus dem Benutzeradreßraum in das Register ge-

[33] Die sonst unter Unix übliche Methode, die insbesondere Bei Netzwerk-Sockets angewandt wird, mit einem `fcntl()`-Aufruf eine Datei oder einen Socket in den nichtblockierenden Modus zu schalten, wurde deswegen nicht im Treiber vorgesehen. Stattdessen können die nichtblockierenden `ioctl()`-Kommandos oder eine mit `select()` arbeitende Abfrage (siehe Abschnitt 5.2.8 auf Seite 37) verwendet werden.

```
    if(_IOC_DIR(cmd)&_IOC_READ)
     {
      if((result=kernel_verify_area(VERIFY_WRITE, (unsigned long *)param,
                                    sizeof(unsigned long)))!=0) return result;
      if(pcicommand) /* Read PCI CCMR/CSTR */
       {
        int result;
        if((result=kernel_pcibios_read_config_dword(mdsp_board[boardnr].busnr,
                                                    mdsp_board[boardnr].devnr,
                /* dword -> COMMAND and STATUS */   PCI_COMMAND,
                                                    &oldval))) return result;
       }
      else /* Read MDSP register */
        oldval=mdsp_read_reg(address,type);
     }
    if(_IOC_DIR(cmd)&_IOC_WRITE)
     {
      if((result=kernel_verify_area(VERIFY_READ, (unsigned long *)param,
                                    sizeof(unsigned long)))!=0) return result;
      val=(__u32)kernel_get_user((unsigned long *)param);
      if(pcicommand) /* Read PCI CCMR/CSTR */
       {
        int result;
        if((result=kernel_pcibios_write_config_dword(mdsp_board[boardnr].busnr,
                                                     mdsp_board[boardnr].devnr,
                 /* dword -> COMMAND and STATUS */   PCI_COMMAND,
                                                     val))) return result;
       }
      else /* Write MDSP register */
        mdsp_write_reg(address,val,type);
     }
    if(_IOC_DIR(cmd)&_IOC_READ) /* write remembered value */
      kernel_put_user((unsigned long)oldval,(unsigned long *)param);
    return 0;
   } /* end if minor>0&&minor<=MDSP_MAX) */
  return -ENODEV;
}
```

Quelltext 31: `ioctl_mdsp()` (3)

schrieben und dieser Wert im Benutzeradreßraum abschließend durch den in `oldval` gespeicherten Wert ersetzt.

Die `ioctl_mdsp()`-Funktion übergibt bei Erfolg 0 und bei einem aufgetretenen Fehler den zugehörigen Fehlerwert an den `ioctl()`-Aufruf des Benutzerprozesses.

5.2.8 `select_mdsp()`

Die Funktion `select()` dient dazu, virtuelle Dateien auf die Verfügbarkeit von Daten zum Lesen mit `read()`, auf die Annahme von Daten durch `write()` oder Ausnahmebehandlungen innerhalb eines Zeitraumes zu überprüfen. Dies ist beispielsweise sinnvoll, wenn eine bidirektionale Datenübertragung realisiert werden soll, ohne daß beim Aufruf von `read()` ohne verfügbare Daten oder `write()` bei Nichtbereitschaft der Datei zur Übernahme blockiert wird.

Die `select()`-Funktion kann mit Hilfe der bereits im Treiber implementierten `ioctl()`-Kommandos und Aufrufen von `usleep()` oder `sleep()` im Anwenderprogramm prinzipiell ersetzt werden. Jedoch ist `select()` besser dazu geeignet, auf das Eintreffen der Lese- oder Schreibereitschaft des Datentransferregisters zu warten, als dies mit aufwenigeren Sequenzen eigener Statusabfragen und Warteintervallen im Anwenderprogramm zu realisieren ist.

```
static int select_mdsp(struct inode *inode, struct file *file, int type,
                       select_table *wait)
{
 int minor=MINOR(inode->i_rdev),boardnr=minor-1;
 if(minor>0&&minor<=MDSP_MAX&&mdsp_board[boardnr].ready)
  {
   int flag;
   for(flag=0;flag<2;flag++)
    { /* try two times, pause in between */
     __u32 status=mdsp_read_reg(mdsp_board[boardnr].hstr,
                                mdsp_board[boardnr].basetype);
     switch(type)
      {
       case SEL_IN:  if(status&HSTR_HRRQ) return 1; break;
       case SEL_OUT: if(status&HSTR_HTRQ) return 1; break;
       case SEL_EX:  if(status&HSTR_TRDY) return 1; break;
       default:      return -EINVAL;
      }
     if(flag==0) kernel_select_wait(&mdsp_board[boardnr].wait,wait);
    }
  }
 else return -ENODEV;
 return 0;
}
```

Quelltext 32: `select_mdsp()`

In `select_mdsp()` wird innerhalb von `switch()` nach der Art der Verfügbarkeit unterschieden, auf die gewartet werden soll. Wurde in `select()` der fragliche Dateideskriptor in der Liste der auf Lesebereitschaft zu prüfenden Dateien abgelegt, so wird das Statusregister der DSP-Karte auf das gesetzte Flag `HSTR_HRRQ` überprüft. Soll auf Schreibbereitschaft überprüft werden, so ist das Statusregister auf `HSTR_HTRQ` hin zu untersuchen. Wird der Dateideskriptor in der Liste der auf Ausnahmebehandlungen zu untersuchenden Dateiliste übergeben, so wird das `HSTR_TRDY`-Flag im Statusregister gestestet, welches signalisiert, daß sowohl das Transmit- als auch das Receive-FIFO leer ist, daß also weder auf der Host-Seite noch auf der DSP-Seite Daten in der Wartenschlange sind. Dies eignet sich beispielsweise zur Überprüfung darauf, ob alle im FIFO abgelegten Daten bearbeitet wurden.

Ist das abgefragte Flag gesetzt, kehrt `select_mdsp()` sofort mit dem Erfolgswert 1 zurück.[34] Ist es nicht gesetzt, also die angeforderte Verfügbarkeitsbedingung nicht

[34] Es handelt sich hier wieder um eine Ausnahme, da die anderen Funktionen, wie in den vorigen Abschnitten erwähnt, vorzugsweise mit 0 als Rückgabewert bei Erfolg zurückkehren.

erfüllt, so wird der aufrufende Prozeß mit der Kernel-Prozedur `select_wait()` in den Wartezustand versetzt, bis der in der `select_table *wait` verstrichene Timeout erreicht ist. Danach wird das Statusregister noch einmal geprüft und bei gesetztem Flag 1, bei nicht gesetztem Flag 0 ohne weitere Wartezeit an die aufrufende `select()`-Funktion des Anwenderprozesses übergeben.

Als Beispiel soll hier wieder ein Programmfragment aus dem Quelltext von `mdsptest` (Abschnitt 6.2, Seite 45) dienen.

```
const struct timeval tc={10,0};
struct timeval tv=tc;
fd_set fds;

...

printf("Select()ing DSP for read (10s)..."); fflush(stdout);
FD_ZERO(&fds);
FD_SET(fd,&fds);
result=select(FD_SETSIZE,&fds,NULL,NULL,&tv);
printf("returned after %d sec, %d usec, status=%d\n",tc.tv_sec-tv.tv_sec,
                                                      tc.tv_usec-tv.tv_usec,
                                                      result);
```

Quelltext 33: Beispiel für den Aufruf von `select()`

Durch das Makro `FD_ZERO()` aus `/usr/include/linux/time.h` werden zunächst alle Einträge in der `fd_set`-Struktur `fds` zurückgesetzt. `FD_SET(fd,&fds)` trägt den Dateideskriptor einer mit `open()` geöffneten `mdsp`-Gerätedatei in die zu untersuchende Liste ein. Der Aufruf von

```
result=select(FD_SETSIZE,&fds,NULL,NULL,&tv);
```

benutzt schließlich die vom `mdsp`-Modul zur Verfügung gestellte Funktion `select_mdsp()`, um innerhalb eines Zeitintervalls `tv`, hier mit 10 Sekunden plus 0 Mikrosekunden initialisiert, die Gerätedatei auf vom DSP ins Transferregister geschriebene Daten hin zu untersuchen. Der Rückgabewert von `select()` gibt die Anzahl der Dateien an, für die innerhalb der gewählten Zeitspanne die Verfügbarkeitsbedingung erfüllt ist. Mit dem Makro `FD_ISSET()` kann der Dateideskriptor aus `fds` ermittelt werden, für den die Bedingung zutrifft. Da nur ein einziger Deskriptor übergeben wurde, ist dies hier jedoch nicht notwendig.

Unter Linux werden die Variablen `tv_sec` und `tv.tv_usec` in der an `select()` übergebenen `timeval`-Struktur verändert. Es wird die bis zum Eintreffen der angeforderten Verfügbarkeit verstrichene Zeit von den Anfangswerten abgezogen. So ist auch die Angabe der verstrichenen Zeit möglich, wie im abschließenden `printf()` in Quelltext 33 gezeigt.

5.2.9 **seek_mdsp()**

Wie in Abschnitt 2 auf Seite 15 erwähnt, übergibt das Kernel-Modul `mdsp.o` eine Liste von Prozeduren an das virtuelle Dateisystem, und stellt so die im Modul definierten Funktionen Anwenderprozessen zur Verfügung.

Wird in dieser `file_operations`-Struktur eine Prozedur oder Funktion mit `NULL` initialisiert, so kehrt der Versuch, diese Prozedur vom Anwenderprozeß aus aufzurufen, sofort mit dem Fehlerwert `EINVAL` zurück. Eine Ausnahme hiervon bildet das Element `lseek`. Wird es mit `NULL` initialisiert, so wird eine Defaultroutine im Kernel aufgerufen, die die Position in der virtuellen Datei zurückgibt oder verändert.

Dies kann in Ausnahmefällen in einem Anwenderprogramm zu Fehlern führen, da eine Positionierung innerhalb der zum `mdsp`-Modul gehörenden Gerätedateien keinen Sinn ergibt. Als Abhilfe wurde eine Dummy-Funktion `lseek_mdsp()` implementiert, die dem aufrufenden `lseek()` eine Fehlermeldung zurückliefert.

```
/* Per default, Linux returns a successful lseek if set to NULL in         */
/* file_operations. The purpose of this function is to return a correct    */
/* error message to the caller.                                             */
static int lseek_mdsp(struct inode *inode, struct file *file, off_t offset,
                      int origin)
{
 return -ESPIPE;
}
```

Quelltext 34: `lseek_mdsp()`

Die restlichen Elemente der `file_operations`-Struktur werden nicht implementiert, da sie für die Unterstützung der DSP-Karte keine Funktionalität bieten. Dies sind `readdir()`,[35] `mmap()`,[36], `fsync()`,[37] `fasync()`,[38] `check_media_change()`[39] und `revalidate()`.[40]

[35] Auslesen eines Verzeichnisindexes
[36] Abbildung des Dateinhaltes in den Benutzeradreßraum
[37] Zurückschreiben von Dateipuffern
[38] Asynchrone Benachrichtigung eines Prozesses beim Eintreffen eines `SIGIO`-Signals
[39] Überprüfung bei blockorientierten Geräten, ob ein Wechsel des Mediums stattgefunden hat
[40] Aktualisierung des virtuellen Dateisystems nach einem Wechsel des Mediums

5.3 Ablaufdiagramm

Abbildung 2 zeigt schematisch die Funktionsweise des `mdsp`-Kernel-Moduls, wobei zwischen Abläufen im Bereich des Anwenderprogramms und Abläufen im Kernel-Bereich unterschieden wird.

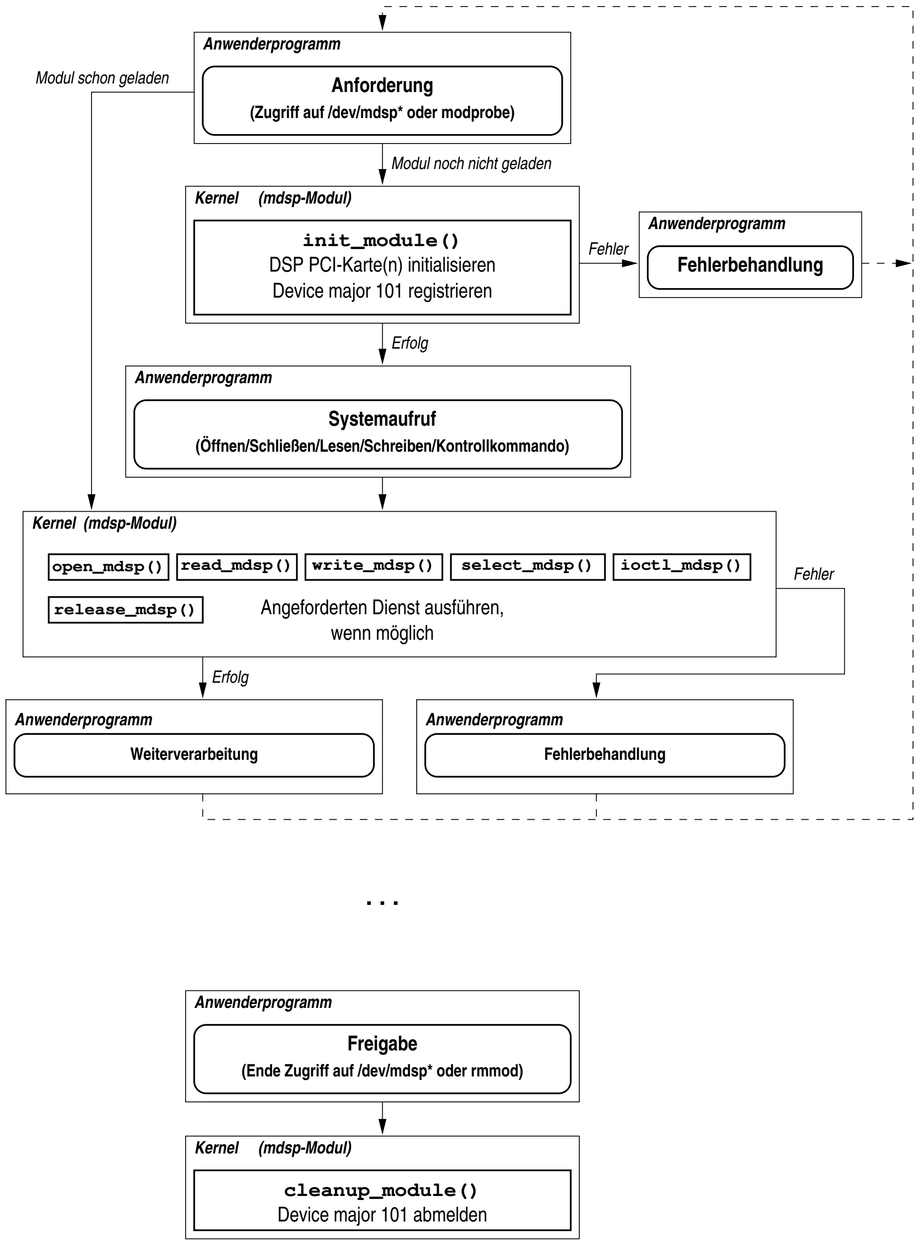

Abbildung 2: Interaktion zwischen Anwenderprogramm und `mdsp`-Modul

Beim Zugriff auf eine Gerätedatei mit der major number 101 wird ein Dienst des `mdsp`-Modul angefordert. Der `kerneld`-Dämon lädt das Modul und führt die Initialisierungsroutine `init_module()` innerhalb des Moduls aus, wenn nicht bereits geschehen.

Tritt hierbei ein Fehler auf, so wird dem Anwenderprogramm durch einen geeigneten Rückgabewert an die aufrufende Funktion mitgeteilt, daß der angeforderte Dienst des virtuellen Dateisystems nicht zur Verfügung steht. Andernfalls wird in die Funktion des Kernel-Module verzweigt, deren Dienst angefordert wurde (`open_mdsp()` beim Aufruf von `open()`, `read_mdsp()` beim Aufruf von `read()`, `write_mdsp()` beim Aufruf von `write()`, `select_mdsp()` beim Aufruf von `select()`, `ioctl_mdsp()` beim Aufruf von `ioctl()` oder `release_mdsp()` beim Aufruf von `close()`, siehe Abschnitt 5.2.3 bis 5.2.9).

Anhand des von der aufgerufenen Funktion des Kernel-Moduls gelieferten Rückgabewertes kann im Anwenderprogramm entschieden werden, ob ein Abbruch oder eine Weiterverarbeitung evtl. zurückgegebener Daten erfolgen soll.

6 Anwendungen

Zur Demonstration und Anwendung der Programmierschnittstelle wurden einige Test-Programme implementiert, um die vom Treiber zur Verfügung gestellten Ressourcen in ihrer korrekten Funktion zu überprüfen und für den Anwender einen einfachen Zugriff auf die DSP56301-Karte ermöglichen. Vier dieser Programme werden im folgenden beschrieben.

6.1 lod2pci

Da der Assembler[41] für den Motorola DSP 56301 lediglich Objektdateien im .CLD und mittels eines mitgelieferten Konvertierungsprogramms im .LOD-Format erzeugen kann, war es notwendig, ein Konvertierungsprogramm zu schreiben, welches das DSP-Programm in ein möglichst unkompliziertes und zum Upload per PCI-Bootstrap verwendbares Datenformat konvertiert.

Das folgende DSP-Programm liest in einer Endlosschleife bei Verfügbarkeit Daten aus dem Transferregister-FIFO, und schreibt den letzten gelesenen Wert um eins inkrementiert ins Transferregister zurück, sofern das Ausgabe-FIFO nicht voll ist. Das Programm wurde als Test für die korrekte Funktion von DSP-Karte und Treiber verwendet und ist für die Durchführung von Benchmarks zum Lesen und Schreiben des Transferregisters geeignet.

```
M_DSR  equ $FFFFC9
M_DTXS equ $FFFFCD
M_DRXR equ $FFFFCB
M_STRQ equ 1
M_SRRQ equ 2

START equ $100
        org p:START                 ; Startadresse $100

        clr a                       ; Akkumulatorregister a löschen
READ
        brclr #M_SRRQ,x:M_DSR,WRITE  ; Wenn FIFO nicht leer,
        movep x:M_DRXR,a            ; lese von FIFO nach a
WRITE
        brclr #M_STRQ,x:M_DSR,READ   ; Wenn FIFO nicht voll,
        movep a,x:M_DTXS            ; schreibe a nach FIFO
        add #1,a                    ; Inkrementiere a
        jmp READ                    ; Endlosschleife
```

Quelltext 35: `beispiel.asm`, ein Testprogramm für den DSP 56301.

Nach Assemblierung und Konvertierung mit

[41] Der Assembler `asm56300` ist ein DOS-Programm, daher kommt hierbei der **dosemu** unter Linux zum Einsatz.

```
asm56300 -a -b -l beispiel.asm
cldlod beispiel.cld > beispiel.lod
```

besitzt die Datei `beispiel.lod` folgendes Format:

```
_START BEISPIEL 0000 0000 0000 DSP56300 6.1.0

_DATA P 000100
200013 0CC902 000003 084E0B 0CC901 FFFFFD 08CE0D 014180
0C0101
_SYMBOL P
READ                 I 000101
WRITE                I 000104
_SYMBOL N
M_DSR                I FFFFC9
M_DTXS               I FFFFCD
M_DRXR               I FFFFCB
M_STRQ               I 000001
M_SRRQ               I 000002
START                I 000100

_END 000100
```

Für den PCI-Bootstrap sind hier lediglich die Werte nach `_DATA P` und vor `_SYMBOL` interessant, nämlich die Startadresse des Programms, auf die sich alle absoluten Sprungadressen beziehen und die 24-Bit-DSP-Opcodes in hexadezimaler Schreibweise. Weiterhin wird für den Bootstrap die Länge des Programme als Anzahl der 24-Bit Opcodes benötigt, auch dies wird von `lod2pci` erledigt.

Als Ausgabeformat zum einfachen Upload über den PCI-Bootstrap erschien dem Autor das folgende Format wünschenswert, welches mit dem Kommando

```
lod2pci beispiel.lod beispiel.pci
```

erzeugt wird. Jede Zeile der Datei enthält einen hexadezimalen Wert, wobei der erste Wert die Länge des Programms in 24-Bit Worten ist, der zweite Wert bezeichnet die Startadresse im Adreßraum der DSP-Karte. Die folgenden Werte sind die DSP-Opcodes.

```
9
100
00200013
000cc902
00000003
00084e0b
```

```
000cc901
00fffffd
0008ce0d
00014180
000c0101
```

Diese Schreibweise in der Darstellung als Text-Hexdump hat den Vorteil, daß beim Einlesen der Werte eine automatische Konvertierung der Byte-Order in die des Host-Systems erfolgt. Die meisten Motorola-Prozessoren verwenden `BIG_ENDIAN` (Anordnung der Bytes innerhalb eines Langwortes 4321), Intel hingegen verwendet `LITTLE_ENDIAN` (Anordnung 1234). Würden die Werte in der Ausgabedatei binär gespeichert werden, so ist die korrekte Anordnung der Bytes in einem 24-Bit Wort nicht auf allen Host-Prozessorarchitekturen gewährleistet.

Ein in dieses Format konvertiertes DSP-Programm kann mit dem Utility `mdsptool` mittels der `-f`-Option (siehe Abschnitt 6.3) auf die DSP-Karte geladen und gestartet werden.

6.2 mdsptest

`mdsptest` ist ein textorientiertes, sehr einfach gehaltenes C-Programm, das alle mit dem implementierten Treiber nutzbaren Systemaufrufe ausführt und speziell für `read()` und `write()` einen Benchmark-Durchlauf ausführt, um den Datendurchsatz von und zur Karte festzustellen.

`mdsptest` wird mit der Device-Datei der zu prüfenden DSP-Karte als Parameter aufgerufen. Ein beispielhafter Aufruf folgt.

```
[knopper@lisa mdsp-1.0]$ mdsptest /dev/mdsp1
  Device file '/dev/mdsp1': Major 101, Minor 1.
  Status register: 0x3
  Control register: 0x900
  Trying to set control register for upload: success.
  Uploading program...success.
  Program running.
  Select()ing DSP for read (10s)...returned after 0 sec, 0 usec, status=1
  Select()ing DSP for write (10s)...returned after 0 sec, 0 usec, status=1
  Benchmarking read()...Read 66846720 bytes in 12.739 secs.
  Throughput was 5570560 Bytes/sec.
  Benchmarking write()...Wrote 66846720 bytes in 9.455 secs.
  Throughput was 7427413 Bytes/sec.
```

Zunächst teilt `mdsptest` dem Anwender mit, daß die Device-Datei `/dev/mdsp1` geöffnet werden konnte, daß die major device number 101 und die minor device number 1 ist. Dies entspricht (siehe 4.2) der ersten vom Treiber gefundenen DSP-Karte.

Anschließend wird die Datei mit `open()` geöffnet und mittels `ioctl()`-Aufrufen werden die Werte des Status- und Kontrollregisters der Karte ausgelesen.

Anschließend wird das aus dem Kontrollregister gelesene Langwort auf das gesetzte Flag `HF0` hin überprüft, welches signalisiert, daß der Bootstrap-Modus bereits beendet wurde. Ist dies nicht der Fall, wird mittels `write()` ein kurzes Testprogramm (siehe Abschnitt 35 auf Seite 43) im Bootstrap-Modus übertragen, welches den DSP veranlaßt, in einer Endlosschleife Daten vom Transferregister entgegenzunehmen und den letzten gelesenen Wert bei einem Lesevorgang des Hosts jeweils um 1 erhöht zurückzuschreiben. Dieses assemblierte Programm ist als DSP-Maschinencode in `mdsptest.c` als Feld von 32-Bit-Werten abgelegt und wird mit `write()` übertragen.

Mit der `select()`-Funktion wird überprüft, ob Daten vom Transferregister gelesen werden können (was darauf schließen läßt, daß das übertragene Programm tatsächlich abgearbeitet wird) und ob der DSP weitere Daten vom Transferregister entgegennehmen kann.

Ist dies der Fall, so wird der Durchsatz beim Lesen vom und Schreiben zum Transferregister mittels `read()` und `write` gemessen, indem insgesamt 168 Megabyte Daten vom DSP gelesen und an den DSP übertragen werden, wobei die hierfür benötigte Zeit mit der BSD-Funktion `gettimeofday()` gemessen wird.

Der auf dem verwendeten Rechner (Intel Pentium, 166 MHz, PCI Bustaktfrequenz 33 MHz) gemessene Durchsatz beträgt beim Lesen ca. 5,5 Megabytes/s, beim Schreiben 7,4 Megabytes/s. Dies gilt für das Lesen und Schreiben mit Polling-Handshake. Wird die Karte als Busmaster verwendet, wobei der DSP entsprechende Routinen abarbeiten muß und auf der Host-Seite von der Karte adressierbarer Speicher zur Verfügung stehen muß, sind noch wesentlich höhere Werte zu erwarten. Dies ist im Treiber jedoch

```
/* Simple DSP program, reads from and writes to HI32 in a loop */
   unsigned long prog[]={
   0x00000009, /* Length */
   0x00000100, /* Address */
   0x00200013, /* DSP prog */
   0x000cc902,
   0x00000003,
   0x00084e0b,
   0x000cc901,
   0x00fffffd,
   0x0008ce0d,
   0x00014180,
   0x000c0101};

      ...

   write(fd,(char *)prog,sizeof(prog));
```

Quelltext 36: Das Testprogramm als `unsigned int`-Feld nicht vorgesehen, da hierfür ein permanent auf der DSP-Karte ablaufendes Programm erforderlich wäre, welches den Datenfluß vom und zum Host selbständig steuert. Der Autor dieser Arbeit hielt den hierfür erforderlichen Aufwand nicht nicht gerechtfertigt, insofern dies auch hohe, wenn nicht unlösbare Anforderungen an die Entwickler von DSP-Programmen stellen würde, die sich neben der korrekten Funktion ihrer eigentlichen DSP-Programme auch noch um den Datentransfer und das Handshake zwischen Karte und Host kümmern müßten.

War die Ausführung von `mdsptest` erfolgreich, so sind die Systemaufrufe `open()`, `ioctl()`, `select()`, `write()`, `read()` und `close()` auf die gewählte Gerätedatei anwendbar und der DSP arbeitet das per PCI-Bootstrap übertragene Testprogramm korrekt ab.

6.3 mdsptool

`mdsptool` ist ein für die Benutzung mit dem `mdsp`-Modul konzipiertes Anwenderprogramm, das die vom Treiber zur Verfügung gestellten Systemaufrufe über eine einfache Schnittstelle zum Anwender nutzt.

Das Programm arbeitet textorientiert und wird über Kommandozeilenparameter gesteuert. `mdsptool -help` gibt eine kurze Hilfestellung aus.

```
MDSPTOOL (C) Klaus Knopper, March 1998
Usage: ../Tools/mdsptool options
List of options:
  -#           Apply following command to DSP-Card number # (default 1)
  -a           Print information for all MDSP boards from /dev/mdspstat
  -c           Print contents of host control register
  -C 0xnnnnnnnn Write hex number nnnnnnnn to host control register
  -d           Dump data from DSP (forever loop)
  -D 0xnnnnnnnn Read 0xn data words from DSP
  -f file      Upload hexdump file in bootstrap mode (first thing to do
               after a reset, see documentation for file format.
               Use - for stdin and make sure that HCTR_HF0 is cleared.)
  -i           Print contents of host vector control register
  -I 0xnnnnnnnn Write hex number nnnnnnnn to host vector control register
  -p           Print contents of pci control register
  -P 0xnnnnnnnn Write hex number nnnnnnnn to pci control register
  -t           Print contents of data transfer register immediate
  -T 0xnnnnnnnn Write hex number nnnnnnnn to data transfer register immediate
  -s           Print contents of host statusregister
  -S 0xnnnnnnnn Write hex number nnnnnnnn to host status register
  -v           Be verbose (print decoded values of registers)
```

Die vom Benutzer angegebenen Kommandozeilenparameter werden sequentiell in der gegebenen Reihenfolge abgearbeitet. Mit der Option `-#`, wobei `#` einer in Dezimalschreibweise angegebenen Zahl zwischen 1 und 16 entspricht, wird die Nummer der DSP-Karte angegeben, auf die die folgenden Optionen angewandt werden sollen. So entspricht `-1` der der Gerätedatei `/dev/mdsp1` zugeordneten DSP-Karte.

Parameter von Optionen, die ein Schreiben von Registern der DSP-Karte bewirken, müssen zur eindeutigen Unterscheidung von Dezimalwerten in der in C üblichen Hexadezimalschreibweise mit `0x` als Präfix angegeben werden.

Tabelle 5 zeigt eine ausführliche Optionsübersicht.

-#	Die zur DSP-Karte mit der angegebenen Nummer gehörende Gerätedatei wird geöffnet. Die nachfolgenden Optionen werden auf diese Datei angewandt. Wird diese Option weggelassen, so wird `/dev/mdsp1` benutzt.
-a	Der Inhalt von `/dev/mdspstat` wird ausgegeben, die Statusinformationen aller vom `mdsp`-Treiber erkannten DSP-Karten werden angezeigt.
-c	Der Inhalt des Kontrollregisters `HCTR` wird ausgegeben.
-C 0xnnnnnnnn	Der in hexadezimaler Schreibweise angegebene Wert wird in das Kontrollregister `HCTR` geschrieben.
-d	Es wird der Inhalt des Datentransferregisters `HRXR` in einer Endlosschleife als Hexadezimalwert mit Präfix `0x` ausgegeben. Mit `Control-C` kann abgebrochen werden.
-D 0xnnnnnnnn	Es wird die als Hexadezimalwert angegebene Anzahl von 32-Bit Werten aus dem Datentransferregister `HRXR` ausgelesen.
-f file	Der Inhalt der angegebenen Datei wird als Folge von 32-Bit Werten an das Transferregister übertragen. Die Datei kann ein DSP-Programm im unter Abschnitt 6.1 auf Seite 43 angegebenen Format (32-Bit Worte in hexadezimaler Schreibweise, Text) sein. Wird – als Dateiname angegeben, so erwartet `mdsptool` die Eingabe der Daten von `stdin`. Damit das Laden eines DSP-Programmes nicht frühzeitig abgebrochen wird, darf das `HCTR_HF0`-Bit im Kontrollregister nicht gesetzt sein. Da dies durch einen Hardware-Reset (Schalter auf der DSP56301ADM-Karte) nicht automatisch geschieht, sollte es explizit mit der `-C`-Option von `mdsptool` auf 0 gesetzt werden.
-i	Der Inhalt des hostseitigen Interruptregisters des DSP wird ausgegeben.

-I 0xnnnnnnnn	Der in hexadezimaler Schreibweise angegebene 32-Bit Wert wird ins hostseitige Interruptvektorregister des DSP geschrieben. Dabei ist der Wert 0 zu vermeiden, da hierdurch ein ungültiger Interrupt auf dem DSP ausgelöst wird, welcher zu einem unvorhersehbarem Verhalten der HI32-Schnittstelle führen kann (siehe [2] und [3]).
-p	Der Inhalt des PCI-Konfigurationsregisters und des PCI-Statusregisters wird ausgegeben.
-P 0xnnnnnnnn	Der in hexadezimaler Schreibweise angegebene 32-Bit Wert wird in das PCI-Kontroll- und Statusregister übernommen. Die unteren 16 Bit werden ins PCI-Kontrollregister, die oberen 16 Bit ins PCI-Statusregister der gewählten DSP-Karte geschrieben. Wird dabei Bit 2, `PCI_COMMAND_MEMORY`, auf 0 gesetzt, so ist die HI32-Schnittstelle nicht mehr vom `mdsp`-Kernel-Modul ansprechbar.
-t	Der Inhalt des Datentransferregisters wird ohne Warten ausgegeben. Ist das Transfer-FIFO leer, wird eine Fehlermeldung ausgegeben.
-T 0xnnnnnnnn	Der in hexadezimaler Schreibweise angegebene 32-Bit Wert wird ohne Warten in das Datentransferregister geschrieben. Ist das Transfer-FIFO voll, wird eine Fehlermeldung ausgegeben.
-s	Der Inhalt des Statusregisters wird ausgegeben.
-S 0xnnnnnnnn	Der in hexadezimaler Schreibweise angegebene 32-Bit Wert wird in das Statusregister geschrieben. Diese Option sollte mit Vorsicht verwendet werden, da hierduch das Handshake zwischen Host und DSP gestört werden kann.
-v	Die in der Datei `mdsp.h` definierten Makros für die Flags des Status-, Kontroll- und PCI-Registers werden im Zusammenspiel mit den Optionen `-s`, `-c` und `-p` zusätzlich ausgegeben.
-help	Ein englischer Hilfetext wird angezeigt. Dieser wird auch bei fehlerhaften Kommandozeilenparametern ausgegeben.

Tabelle 5: Optionen von `mdsptool`

6.4 xmdsptool

`xmdsptool` ist ein graphisches Frontend für `mdsptool`. Die Oberfläche wurde mit Hilfe der *xforms-Library* und *fdesign* [7] erstellt. Das Programm läuft unter der X-Window Oberfläche und kann daher auch remote von einem X-Terminal oder einem anderen Rechner mit graphischer Oberfläche, die das X11-Protokoll unterstützt, bedient werden.

Alle Funktionen von `mdsptool` stehen als Button per Mausklick oder Tastatur-Shortcut zur Verfügung.

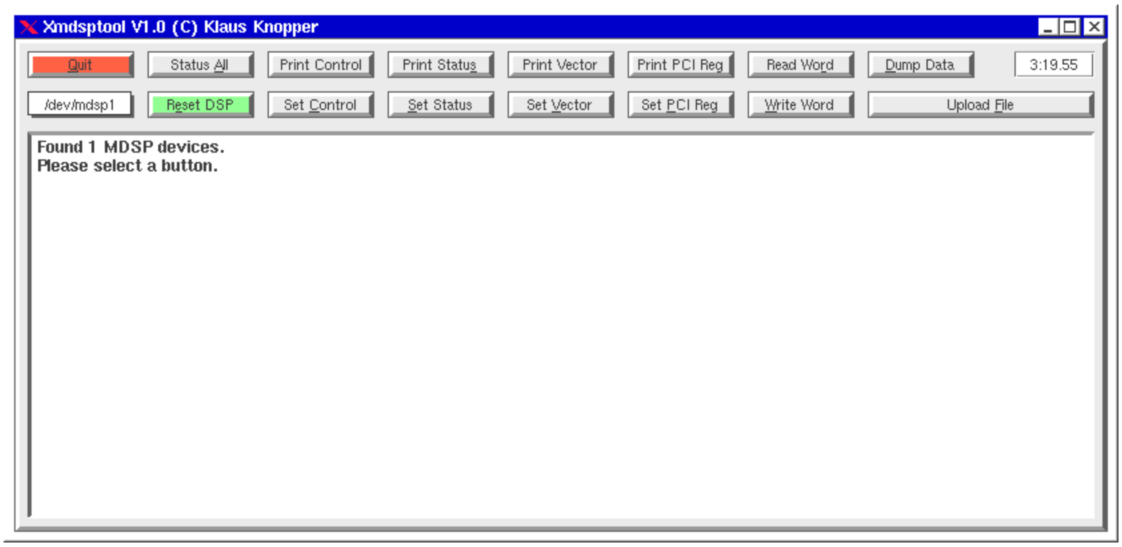

Abbildung 3: Das Hauptfenster von `xmdsptool`.

Abbildung 3 zeigt das Hauptfenster der graphischen Applikation. Die Ausgaben des aufgerufenen Programms `mdsptool` erscheinen im unteren Teil. Sind die Zeilen länger als der angezeigte Bildausschnitt zuläßt, erscheint eine Scrollbar im interen Teil des Fensters, mit der die nicht sichtbaren Teile der Ausgaben erreicht werden können. Gleiches gilt für die Anzahl der angezeigten Zeilen. Hier erscheint bei Bedarf eine vertikale Scrollbar.

Abbildung 4 zeigt das Popup-Fenster zur Anzeige und Auswahl von Bit-Werten am Beispiel des DSP56301 Control-Registers. Das Hauptfenster von `xmdsptool` bleibt während der Darstellung des Popup blockiert, um doppelte Eingaben zu verhindern.

Die vordefinierten Werte des ensprechenden Registers der DSP-Karte werden mit ihren Namenskürzeln (vergl. Datei `mdsp.h`) angezeigt, reservierte oder nicht dokumentierte Bits werden mit Ihrer Nummer versehen. Die aktuell gesetzten Bits sind vorselektiert. Im `Input`-Feld steht der aktuelle Wert aus dem betreffenden Register als Hexa-

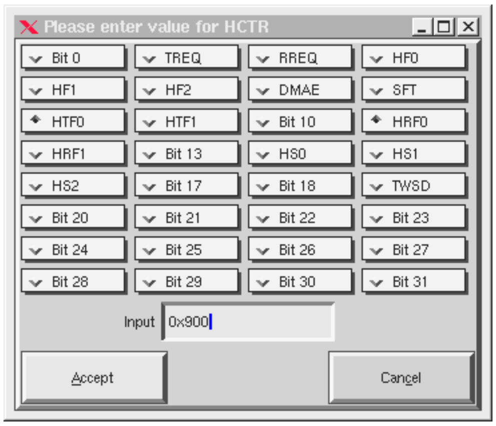

Abbildung 4: Anzeige und Auswahl von Bit-Werten.

dezimalzahl (wie in C üblich mit `0x` als Präfix). Selektiert der Anwender Buttons aus dem Feld der Bitwerte, wird der dargestellte Gesamtwert automatisch angepaßt. Wird der Gesamtwert im `Input`-Feld verädert, so werden die Buttons nach Eingabe von `<Return>` aktualisiert.

Bei Klick auf den `Accept`-Button oder nach Eingabe des Tastatur-Shortcut `a` wird der eingegebene Wert in das betroffene Register der DSP-Karte übernommen.

Der `Upload`-Button dient zum Upload eines DSP-Programms auf die DSP-Karte im Bootstrap-Modus und präsentiert dem Anwender einen Fileselektor wie in Abbildung 5 dargestellt.

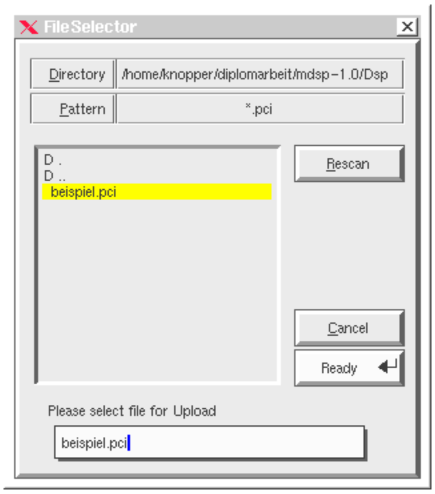

Abbildung 5: Auswahl eines Programms zum Upload

Um zu gewährleisten, daß `xmdsptool` nicht „hängenbleibt“, wenn das aufgerufene mdsptool keine Ausgaben mehr produziert (dies ist beispielsweise der Fall, wenn der DSP keine Werte ins Transfer-Register schreibt und mdsptool versucht, mit der *dump*-Funktion das Register auszulesen), wurde eine Routine eingearbeitet, die mdsptool als Subprozeß startet und die Ausgaben überwacht. Wenn mdsptool 5 Sekunden keine Ausgaben mehr liefert, aber nicht terminiert, wird der Subprozeß mit dem Systemaufruf kill() abgebrochen und es erfolgt eine Fehlermeldung an den Benutzer (Abbildung 6).

Abbildung 6: Timeout durch den mdsptool-Subprozeß

Jedem Button und Auswahl-Menü des Hauptfensters von xmdsptool ist ein Tastatur-Shortcut zugeordnet, wie Tabelle 6 zeigt.

Shortcut	mdsptool Äquivalent	Funktion
a	mdsptool -a	Anzeige der Statusinformationen aller erkannten DSP-Karten
/		Auswahlmenü DSP-Gerätedatei
1-9	mdsptool -#	Auswahl DSP-Karte Nummer #
c	mdsptool -c	Ausgabe Control-Register
C	mdsptool -C 0x#	Setzen Control-Register
s	mdsptool -s	Ausgabe Status-Register
S	mdsptool -S 0x#	Setzen Status-Register
i	mdsptool -i	Ausgabe Vektor-Register
I	mdsptool -I 0x#	Setzen Vektor-Register
p	mdsptool -p	Ausgabe PCI-Konfigurations-/Statusregister
P	mdsptool -P 0x#	Setzen PCI-Konfigurations-/Statusregister
t	mdsptool -t	Ausgabe Transferregister
T	mdsptool -T 0x#	Setzen Transferregister
D	mdsptool -D 0x#	Ausgabe Transferregister 0x# Werte
f	mdsptool -f Datei	Upload Datei
q		Programmende

Tabelle 6: Tastatur-Shortcuts von xmdsptool.

7 Realisierung der TCP/IP-Schnittstelle

Nach der Erläuterung der im `mdsp`-Kernel-Modul implementierten Funktionen und Definition der Anwenderschnittstelle soll nun ein Überblick über die Möglichkeiten gegeben werden, wie die DSP-Karte über ein TCP/IP-basiertes Netzwerk angesprochen werden kann.

7.1 X11, telnet und rsh/ssh

Die netzwerkfähige, graphische Benutzeroberfläche *X-Window* unterstützt die Darstellung einer Applikation und Interaktion mit der Applikation von einem über TCP/IP angesprochenen Rechner im Netzwerk aus. Der von der Applikation gestartete Prozeß, beispielsweise das im vorigen Abschnitt vorgestellte `xmdsptool`, läuft auf dem Hostsystem, in das die DSP-Karte eingebettet ist. Die Ein- und Ausgaben von `xmdsptool` können jedoch auf dem entfernten Rechner erfolgen.

Verfügt der X-Server, der die Darstellung und Umleitung der Eingaben an `xmdsptool` übernehmen soll, über keinen Autorisationsmechanismus oder wurde dieser Mechanismus explizit mit dem Kommando

```
xhost +
```

abgeschaltet, so genügt es, beim Start von `xmdsptool` die Option `-display` gefolgt von der IP-Adresse und Screennummer des X-Servers anzugeben.

Beispiel:

```
xmdsptool -display sushi.unix-ag.uni-kl.de:0
```

Die graphischen Ausgaben von `xmdsptool` werden so per X11-Protokoll an das angegebene X-Display umgeleitet, die Tastatur-Eingaben und Maus-Events, die der Anwender an den X-Server übergibt werden im Gegenzug vom X-Server an `xmdsptool` übermittelt.

Unterstützt der X-Server das `MIT-MAGIC-COOKIE`-Autorisationsprotokoll, so kann der `xmdsptool` Benutzer zunächst den Autorisationsmechanismus auf seiner Seite aktivieren, indem er mit Hilfe des Programms `xauth` die Zugriffsberechtigung zum X-Display in einer Datei mit dem Namen `.Xauthority` in seinem Homeverzeichnis setzt. Diese Zugriffsberechtigung kann auf dem Rechnersystem, auf dessen X-Server zugegriffen werden soll, mit dem Kommando `xauth list $DISPLAY` erfragt, und das Ergebnis mit `xauth add` auf dem DSP-Host zum Eintrag in die

.Xauthority-Datei übernommen werden. Auf diese Weise wird verhindert, daß das X-Display für Zugriffe beliebiger Hosts geöffnet werden muß.

Die Bedienung von xmdsptool über ein X-Display eines entfernten Rechners ist identisch mit der lokalen Bedienung.

Verfügt der entfernte Rechner nicht über eine X11-fähige Benutzeroberfläche, so kann nicht mit dem graphikorientierten xmdsptool gearbeitet werden. Alternativ bietet sich ein telnet-orientierter Zugang an. Hierbei muß sich der Benutzer mit dem Client-Programm telnet vom Remote-Host aus mit seiner Benutzerkennung auf dem Rechner, der die DSP-Karte beherbergt, anmelden, und kann dann mit dem mdsptool-Utility oder einem anderen textorientierten Programm auf die Karte zugreifen.

Mit dem Berkeley-Dienst rsh ist der Vorgang ähnlich, nur kann hier mit Hilfe einer Datei .rhosts im Homedirectory des Benutzers die Paßwortabfrage entfallen, wenn in dieser Datei der Rechnername oder die IP-Adresse des Remote-Systems und die Benutzerkennung auf dem Remote-System vermerkt sind. Ist dies der Fall, so kann der Benutzer auf dem Remote-System das Kommando rsh dsphost mdsptool parameter starten und bekommt die Ausgabe dieses Kommandos übermittelt.

Die Secure Shell ssh verschlüsselt zusätzlich die in beide Richtungen übertragenen Daten nach einem wählbaren Verfahren[42], wobei die Kommunikation zwischen den Anwenderprogrammen auf beiden Seiten transparent bleibt, d.h. diese müssen sich nicht selbst um die Entschlüsselung der Daten kümmern. Abgesehen von der „abhörsicheren" Datenübertragung bietet die ssh den Vorteil, daß die X11-Autorisation für X11-Applikationen automatisiert wird und X11-Verbindungen durch den von der ssh aufgebauten sicheren Kanal getunnelt werden. ssh-Clients sind derzeit für Unix[43] und andere Betriebssysteme[44] erhältlich.

7.2 Zugriff über inetd

Ist keine explizite Autorisation zum Zugriff auf den Host, in dem die DSP-Karte untergebracht ist, erforderlich, und soll die Steuerung und der Datentransfer von der und zur DSP-Karte ohne eine vorherige Anmeldung eines Benutzers auf dem Hostsystem möglich sein, so bietet es sich an, einen Netzwerk-Dienst einzurichten, der den Zugriff von außen auf die Karte verwaltet.

[42] IDEA, DES, 3DES und andere

[43] Der Quelltext von ssh-Server und -Clients ist unter den Bedingungen einer GPL-ähnlichen Lizenz auf diversen ftp-Servern im Internet erhältlich.

[44] kostenpflichtig

Um die Funktionsweise des Meta-Dämons `inetd` zu erläutern, muß zunächst auf die Funktionsweise von TCP/IP-basierenden Netzdiensten im allgemeinen kurz eingegangen werden.[45]

Eine Netzwerkverbindung zwischen zwei Rechnersystemen über TCP/IP ist eindeutig spezifiziert durch die folgenden Parameter:

- Die IP-Adresse des 1. Host,
- die IP-Adresse des 2. Host,
- eine Portnummer des 1. Host,
- eine Portnummer des 2. Host,
- das verwendete Protokoll (`tcp` oder `udp`).

Ein Anwenderprogramm, das auf einen Netzwerk-Dienst eines vernetzten Rechners zugreifen soll, muß folglich auf dem Rechner, auf dem es abläuft, zunächst einen Port belegen. Für nichtprivilegierte Benutzerprozesse sind dies unter Unix die Portnummern oberhalb 1023. Ports werden softwaretechnisch ähnlich wie Dateien vom Anwenderprogramm angesprochen.

Damit auf einem Host im Netzwerk mehrere Netzdienste angeboten werden können, ist eine Unterscheidung dieser Dienste durch die Portnummer nötig. Die standardisierten Netzdienste wie `telnet`, `ftp`, `www` besitzen feste Portnummern, die in der Datei `/etc/services` angegeben sind. Der Meta-Dämon `inetd` verwaltet die in seiner Konfigurationsdatei `/etc/inetd.conf` angegebenen Ports und startet beim Zugriff eines Anwenderprogramme, unabhängig davon ob dieses lokal oder auf einem entfernten Rechner gestartet wird, das ebenfalls in dieser Datei angegebene Server-Programm.

Als Beispiel soll das Programm `mdsptool` bei einem Zugriff auf den Port 1000 gestartet werden und mittels der `-a`-Option den Status aller DSP-Karten ausgeben. Hierzu ist folgender Eintrag in `inetd.conf` erforderlich:

```
1000 stream tcp nowait nobody /usr/local/bin/mdsptool mdsptool -a
```

Der noch aktive `inetd`-Prozeß muß von der Änderung an seiner Konfigurationsdatei in Kenntnis gesetzt werden. Dies kann unter Linux mit dem Kommando `killall -HUP inetd` erreicht werden. Der `inetd` erhält mit dem „hangup“-Signal die Aufforderung, seine Konfigurationsdatei neu einzulesen.

[45] Sehr ausführlich wird die Thematik in [8] behandelt.

Beim Zugriff auf Port 1000 wird nach Wirksamwerden der Änderung das Kommando `/usr/local/bin/mdsptool -a` ausgeführt, welches nach der Umleitung seines Ausgabekanals auf Port 1000 mit den Rechten der Benutzerkennung `nobody` läuft.[46]

Folgender Eintrag in der Datei `inetd.conf` ermöglicht es, ein DSP-Programm als Text-Hexdump auf Port 1001 zu übertragen und auf Port 1002 ständig die vom DSP ins Transferregister geschriebenen Werte ausgegeben zu bekommen.

```
1001 stream tcp nowait nobody /usr/local/bin/mdsptool mdsptool -f -
1002 stream tcp nowait nobody /usr/local/bin/mdsptool mdsptool -d
```

Der Zugriff auf die so mit Ein- und Ausgaben des Programms `mdsptool` verknüpften Ports kann mit dem Programm `telnet` unter Angabe der IP-Adresse des DSP-Hosts und der Portnummer erfolgen, oder mit Hilfe eines kurzen C-Programms, das die standardisierte BSD-Socket Programmierschnittstelle verwendet, wie in Quelltext 37 dargestellt.

[46]Die Änderung der Privilegien eines Prozesses auf die eines Benutzers mit den wenigsten Rechten ist unter Unix bei Netzwerkdämonen, die keine expliziten Berechtigungen zum Zugriff auf Dateien oder Benutzerkennungen benötigen, aus Sicherheitsüberlegungen üblich.

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define SERVER_IP    "131.246.66.222" /* Numerische IP-Adresse DSP-Host */
#define SERVER_PORT  1000             /* Server-Port                     */

#define BUF_LEN 4096

void fatal(char *msg) { fputs(msg, stderr); exit(-1); }

int main(int argc, char **argv)
{
 int sock;
 static struct sockaddr_in server;
 char buf[BUF_LEN];

 if((server.sin_addr.s_addr=inet_addr(SERVER_IP))<0) fatal("unknown host\n");

 /* Socket holen */
 if((sock=socket(AF_INET,SOCK_STREAM,0))<0) fatal("could not open socket\n");

 server.sin_family = AF_INET;            /* Typ INET eintragen    */
 server.sin_port   = htons(SERVER_PORT); /* Server-Port eintragen */

 /* Socket mit Server-Port verbinden */
 if(connect(sock,(struct sockaddr *)&server,sizeof(server)))
   fatal("can't connect\n");

 /* Ausgaben des Servers in Empfang nehmen */
 read(sock, buf, BUF_LEN);
 /* Und ausgeben. */
 puts(buf);

 /* Socket wieder freigeben, Verbindung beenden */
 close(sock);
 return 0;
}
```

Quelltext 37: `mdspnet.c` – Zugriff auf Netzwerkports über TCP/IP

8 Zusammenfassung

In der vorliegenden Arbeit wurde ein Kernel-Treiber für die Motorola DSP56301ADM Karte in der PCI-Variante entwickelt und eine Schnittstelle zum Zugriff und zur Steuerung dieser Karte für Anwendungsentwickler definiert. Zusätzlich wurden Beispiel-Applikationen implementiert, die dem Anwender einen wahlfreien Zugriff auf die Register der Karte ermöglichen.

Im `mdsp`-Kernel-Modul wurden die für das Linux-Dateisystem elementaren Dateioperationen implementiert (Abschnitt 5). Es können über das PCI-Interface der DSP56301ADM-Entwicklerkarte Daten bidirektional zwischen dem Hostsystem und der HI32-Schnittstelle des DSP ausgetauscht werden. Das Handshake zwischen DSP und Hostsystem wurde über einen Polling-Mechanismus realisiert, der unabhängig vom auf dem DSP ablaufenden Programm arbeitet. Die Unterstützung mehrerer gleichzeitig arbeitender DSP-Karten und die Möglichkeit zur Erkennung weiterer DSP-Karten ähnlichen Typs wurde vorgesehen.

Ein Upload von DSP-Programmen per PCI-Bootstrap sowie das Lesen und Schreiben aller Register der HI32-Schnittstelle des DSP ist mit Hilfe der zusätzlich implementierten Hilfsprogramme `mdsptool` und `xmdsptool` möglich (Abschnitt 6). Als Programmierschnittstelle werden die unter Unix in der Standard C-Bibliothek üblichen Systemaufrufe für Dateioperationen `open()`, `close()`, `read()`, `write()`, `ioctl()` und `select()` im `mdsp`-Kernel-Modul unterstützt.

Neben der leichten Erweiterbarkeit des Treibers wurde auf Stabilität, Fehlertoleranz und Multitasking-Konformität in der Linux-Umgebung Wert gelegt.

Als TCP/IP-Schnittstelle kann die graphische, netzwerkfähige X11-Oberfläche, oder eine telnet-, rsh- oder ssh-Session dienen (Abschnitt 7.1). Ein Zugriff ohne Autorisationsmechanismus ist durch den Meta-Netzwerkdämon `inetd` möglich, der die Serverdienste über Ports des DSP-Rechnersystems verwaltet (Abschnitt 7.2).

Für zukünftige Erweiterungen interessant ist die bisher nicht genutzte Möglichkeit, über eine Implementation als DSP-Programm das von der HI32-Schnittstelle des DSP56301 unterstützte PCI-Busmastering mit Datentransfer im Burst-Modus höhere als die in Abschnitt 6.2 ermittelte Datentransferrate zu erreichen.

Die Quelltexte des Kernel-Treibers und der Hilfsprogramme sowie die von Motorola zur Verfügung gestellten Dokumentationen im PDF-Format, Distributionen der verwendeten Toolkits für Linux und diese Dokumentation sind auf einer CD am Lehrstuhl für Hochspannungstechnik und EMV erhältlich.

Literatur

[1] DSP56301ADMUM/AD *DSP56301ADM User's Manual*, Motorola Inc. 1995

[2] DSP56301UM/AD *DSP56301 24-Bit Digital Signal Processor User's Manual*, Motorola inc. 1995

[3] DSP56301FM/AD *DSP56301 24-Bit Digital Signal Processor Family Manual*, Motorola Inc. 1995

[4] IEEE Standards Association, POSIX 1003.1, 1994

[5] Michael Beck, Harald Böhme et al., *Linux-Kernel-Programmierung – Algorithmen und Strukturen der Version 2.0*, Addison-Wesley 1997

[6] Tom Shanley/Don Anderson, *PCI System Architecture*, Addison-Wesley 1996

[7] T.C. Zhao, Mark Overmars, *Forms Library – A Graphical User Interface Toolkit for X*, Internet 1998

[8] Andrew S. Tanenbaum, *Computer-Netzwerke*, 2. Auflage, Attenkirchen 1990

Anhang

A PCI Bootstrap Programm des DSP 56301

Das Bootstrap Programm für den PCI-Modus aus dem on-Chip ROM des DSP56301 liest zunächst die Länge des auszuführenden Programms aus dem HI32-Transferregister, danach die Startadresse. Anschließend werden in einer Schleife die Opcodes des Programms aus dem Transferregister in den Programmspeicher ab der angegebenen Startadresse geschrieben. Hierbei wird das DSP-Statusregister auf das vom Host setzbare `HF0`-Bit überprüft. Ist es gesetzt, so wird der Ladevorgang unterbrochen und sofort in das geladene Programmfragment verzweigt, was sonst erst nach dem Lesen des letzten Wortes vom Transferregister geschieht. Schließlich wird das in den Programmspeicher geladene Programm gestartet (Startadresse in Register `r1`).[47]

```
;======================================================================
; This is the routine that loads from the Host Interface in PCI mode.
; MC:MB:MA=100

PCIHOSTLD
  bset #20,X:M_DCTR      ; Configure HI32 as PCI
  jclr #2,X:M_DSR,*      ; Wait for SRRQ to go high (i.e. data ready)
  movep X:M_DRXR,a0      ; Store number of words
  jclr #2,X:M_DSR,*      ; Wait for SRRQ to go high (i.e. data ready)
  movep X:M_DRXR,r0      ; Store starting address
  move r0,r1             ; save r0

  do a0,_LOOP5           ; Load instruction words
_LBLK
  jset #2,X:M_DSR,_LBBL  ; Wait for SRRQ to go high (i.e. data ready)
  jclr #3,X:M_DSR,_LBLK  ; If HF0=1, stop loading data. Else check SRRQ.
  enddo                  ; Must terminate the do loop
  bra <_LOOP5
_LBBL
  movep X:M_DRXR,P:(r0)+ ; Store 24-bit data in P mem
_LOOP5                   ; and go get another 24-bit word.
  bra <FINISH            ; finish bootstrap

FINISH
; This is the exit handler that returns execution to normal
; expanded mode and jumps to the RESET vector
  andi #$0,ccr             ; Clear CCR as if RESET to 0.
  jmp (r1)
```

Quelltext 38: Das PCI Bootstrap Programm des DSP 56301

[47] Siehe auch [2].

B Installation von Kernel-Modul und Utilities

```
`-- mdsp-1.0
    |-- INSTALL
    |-- Makefile
    |-- README
    |-- Driver
    |   |-- Makefile
    |   |-- mdsp.c
    |   |-- mdsp.h
    |   `-- mdsp.o
    |-- Dsp
    |   |-- beispiel.asm
    |   |-- beispiel.lod
    |   `-- beispiel.pci
    |-- Tools
    |   |-- Makefile
    |   |-- lod2pci
    |   |-- lod2pci.c
    |   |-- mdsp.h -> ../Driver/mdsp.h
    |   |-- mdsptest
    |   |-- mdsptest.c
    |   |-- mdsptool
    |   `-- mdsptool.c
    `-- Gui
        |-- Makefile
        |-- functions.c
        |-- functions.o
        |-- main.c
        |-- main.o
        |-- mdsp.h -> ../Driver/mdsp.h
        |-- xinterface.c
        |-- xinterface.fd
        |-- xinterface.h
        |-- xinterface.o
        `-- xmdsptool
```

Abbildung 7: Verzeichnisbaum `mdsp-1.0`

Im Verzeichnis `mdsp-1.0` befinden sich die Quelltexte des `mdsp`-Treibers und der Utility-Programme. Zur Übersetzung und Installation befinden sich in diesem Verzeichnis und den Unterverzeichnissen `Makefiles`. Mit dem Kommando

```
make all
```

werden die Quelltexte übersetzt. Mit

```
make install
```

wird das Kernel-Modul im Verzeichnis `/lib/modules/misc` installiert, die Gerätedateien `/dev/mdsp1`... `/dev/mdsp16` und `/dev/mdspstat` erzeugt und zur Konfiguration des `kerneld` das Kommando `depmod -a` aufgerufen. Die Utility-Programme `mdsptool`, `lod2pci`, `mdsptest` und `xmdsptool` werden durch `make install` im Verzeichnis `/usr/local/bin/` installiert.

Zum automatischen Starten des Treibers beim Zugriff auf die Gerätedateien `/dev/mdsp*` ist der folgende Eintrag in der Datei `/etc/conf.modules` vorzunehmen:[48]

```
alias char-major-101 mdsp
```

Soll das Modul bereits beim Systemstart geladen werden und ein automatisches Entfernen des Moduls bei Inaktivität verhindert werden, so ist ein Eintrag in die Datei `/etc/rc.d/rc.modules` notwendig.

```
modprobe mdsp
```

Nach dem Laden des Treibers mittels `modprobe` und erfolgreicher Erkennung der vorhandenen DSP-Karten ist der Treiber im Kernel registriert und verwaltet Gerätedateien mit der major number 101.

Eine benutzer- oder gruppenbasierte Zugriffskontrolle kann durch das Setzen der gewünschten Berechtigungen für die `mdsp`Gerätedateien in `/dev` mit Hilfe des Kommandos `chmod` erfolgen.[49]

[48] Siehe auch die Abschnitte 2 auf Seite 16 und 5.2.1 auf Seite 5.2.1.

[49] Siehe **man chmod**.

C Quelltexte des Treibers und der Hilfsprogramme

`mdsp`-Modul – mdsp.h

Diese Hilfsdatei wird für die Übersetzung des `mdsp`-Moduls und der Anwenderprogramme benötigt. Sie enthält Konstanten und Makros, die für den Gebrauch des `mdsp`-Moduls relevant sind.

```
/****************************************************************************\
*                              mdsp.h                                        *
*                                                                            *
* Contains contants and macros for the PCI mode driver mdsp and Register     *
* declarations for the Motorola DSP56xxx Board Series.                       *
\****************************************************************************/

#include <sys/ioctl.h>            /* ioctl() Interface parameters    */

#define MDSP_MAX 16               /* Maximum number of DSP boards    */

#define MDSP_MAJOR 101            /* Registered Major Device Number  */

/* Documentation of Registers and Flags: See "DSP56301 Digital Signal        */
/* Processor User's Manual", Motorola 1995                                   */

/* Registers offsets of the DSP56301, add to Base memory address             */
#define HCTR 0x0010      /* Host Side Control Register  */
#define HSTR 0x0014      /* Host Side Status Register   */
#define HCVR 0x0018      /* Host Side Vector Register   */
#define HTXR 0x001c      /* Host Side Transmit Register */
#define HRXR 0x001c      /* Host Side Receive Register  (same as HTXR) */

/* Status flags for HSTR */
#define HSTR_TRDY 0x00000001 /* Bit 0: Flag Transmitter Ready          */
#define HSTR_HTRQ 0x00000002 /* Bit 1: Flag Host Transmit Data Request */
#define HSTR_HRRQ 0x00000004 /* Bit 2: Flag Host Receive Data Request  */
#define HSTR_HF3  0x00000008 /* Bit 3: Flag Host Flag 3                */
#define HSTR_HF4  0x00000010 /* Bit 4: Flag Host Flag 4                */
#define HSTR_HF5  0x00000020 /* Bit 5: Flag Host Flag 5                */
#define HSTR_HINT 0x00000040 /* Bit 6: Flag Host Interrupt A           */
#define HSTR_HREQ 0x00000080 /* Bit 7: Flag Host Request               */
/* Bits 8-31 reserved */

/* Control Flags for HCTR */
/* Bit 0: Reserved */
#define HCTR_TREQ 0x00000002 /* Bit 1:  Transmit Request Enable        */
#define HCTR_RREQ 0x00000004 /* Bit 2:  Receive  Request Enable        */
#define HCTR_HF0  0x00000008 /* Bit 3:  Host Flag 0                    */
#define HCTR_HF1  0x00000010 /* Bit 4:  Host Flag 1                    */
#define HCTR_HF2  0x00000020 /* Bit 5:  Host Flag 2                    */
#define HCTR_DMAE 0x00000040 /* Bit 6:  DMA Enable (ISA/EISA)          */
#define HCTR_SFT  0x00000080 /* Bit 7:  Prefetch/Fetch Mode            */
#define HCTR_HTF0 0x00000100 /* Bit 8:  Host->DSP   Transfer Mode 0    */
#define HCTR_HTF1 0x00000200 /* Bit 9:  Host->DSP   Transfer Mode 1    */
/* Bit 10: Reserved */
#define HCTR_HRF0 0x00000800 /* Bit 11: DSP->Host   Receive  Mode 0    */
```

```
#define HCTR_HRF1 0x00001000 /* Bit 12: DSP->Host   Receive  Mode 1    */
/* Bit 13: Reserved */
#define HCTR_HS0  0x00004000 /* Bit 14: Host Semaphore 0               */
#define HCTR_HS1  0x00008000 /* Bit 15: Host Semaphore 1               */
#define HCTR_HS2  0x00010000 /* Bit 16: Host Semaphore 2               */
/* Bits 17,18: Reserved */
#define HCTR_TWSD 0x00080000 /* Bit 19: Host Target Wait State Disable */
/* Bits 20-31: Reserved */

/*************************************************************************\
*                        mdsp driver IOCTL Commands                       *
* param 3 is _address_ of unsigned long value to be read to               *
* or to be written.                                                       *
\*************************************************************************/
/* Read/Write Status Register        */
#define MDSPCTL_READ_HSTR  _IOR(MDSP_MAJOR,’s’,unsigned long)
#define MDSPCTL_WRITE_HSTR _IOW(MDSP_MAJOR,’s’,unsigned long)
/* Read/Write Control Register       */
#define MDSPCTL_READ_HCTR  _IOR(MDSP_MAJOR,’c’,unsigned long)
#define MDSPCTL_WRITE_HCTR _IOW(MDSP_MAJOR,’c’,unsigned long)
/* Read/Write Receive FIFO Register  */
#define MDSPCTL_READ_HRXR  _IOR(MDSP_MAJOR,’r’,unsigned long)
#define MDSPCTL_WRITE_HRXR _IOW(MDSP_MAJOR,’r’,unsigned long)
/* Read/Write Transmit FIFO Register */
#define MDSPCTL_READ_HTXR  _IOR(MDSP_MAJOR,’t’,unsigned long)
#define MDSPCTL_WRITE_HTXR _IOW(MDSP_MAJOR,’t’,unsigned long)
/* Read/Write Host Vector Register   */
#define MDSPCTL_READ_HCVR  _IOR(MDSP_MAJOR,’i’,unsigned long)
#define MDSPCTL_WRITE_HCVR _IOW(MDSP_MAJOR,’i’,unsigned long)
/* Read/Write PCI Command and Status Register */
#define MDSPCTL_READ_CCMR  _IOR(MDSP_MAJOR,’p’,unsigned long)
#define MDSPCTL_WRITE_CCMR _IOW(MDSP_MAJOR,’p’,unsigned long)

/* EOF mdsp.h */
```

Quelltext 39: `mdsp.h`

mdsp-Modul – mdsp.c

```
/*****************************************************************************\
*            mdsp.c - Motorola DSP56301 Board PCI Driver for Linux            *
*                                                                             *
* Copyright (C) 1998  Klaus Knopper                                           *
* see mdsp.h for definitions of registers and flags                           *
* #define DEBUG for more verbose output of module                             *
* #define SLOW if you do not care if your system hangs during poll            *
\*****************************************************************************/

#include <linux/config.h>

#ifdef MODULE
#include <linux/module.h>
#include <linux/version.h>
#else
#error "MDSP must be built as Module. Please #define MODULE."
#endif

#ifdef CONFIG_PCI
#include <linux/pci.h>
#include <linux/bios32.h>
#else
#error "You must enable PCI support in your Kernel."
#endif

#include <linux/types.h>
#include <linux/fs.h>
#include <linux/mm.h>
#include <linux/errno.h>
#include <linux/utsname.h> /* system_utsname */
#include <linux/ioport.h>  /* check_region(), request_region() */
#include <asm/segment.h>
#include <asm/io.h>

#include "mdsp.h" /* System-Includes, register definitions, makros */

/* Constants */
#ifndef PCI_VENDOR_ID_MOTOROLA
#define PCI_VENDOR_ID_MOTOROLA 0x1057
#endif

#ifndef PCI_DEVICE_ID_DSP56301
#define PCI_DEVICE_ID_DSP56301 0x1801
#endif

#define MDSP_MEM_SIZE 0x00010000  /* DSP56301 has 64K memory-mapped   */

#define MAX_STATUS 4096           /* Max size of /dev/mdspstat buffer */

#define MDSP_INT_LEN 4            /* 32-bit transfers                 */

/* Utility Functions */
#define PRINTK(format,a...) printk("MDSP: "format, ##a)
#ifdef DEBUG
#define DPRINTK(format,a...) printk("MDSP_DEBUG: "format, ##a)
#else
#define DPRINTK(format,a...)
```

```
#endif

/* For better readability for non-linux programmers: Indicating Names of */
/* kernel functions (Michael wants it)                                   */
#define kernel_readl readl
#define kernel_inl inl
#define kernel_writel writel
#define kernel_outl outl
#define kernel_verify_area verify_area
#define kernel_var_need_resched need_resched
#define kernel_var_current current
#define kernel_var_jiffies jiffies
#define kernel_schedule schedule
#define kernel_put_user put_user
#define kernel_get_user get_user
#define kernel_memcpy_tofs memcpy_tofs
#define kernel_memcpy_fromfs memcpy_fromfs
#define kernel_pcibios_present pcibios_present
#define kernel_pcibios_find_device pcibios_find_device
#define kernel_pcibios_read_config_word pcibios_read_config_word
#define kernel_pcibios_read_config_dword pcibios_read_config_dword
#define kernel_pcibios_write_config_byte pcibios_write_config_byte
#define kernel_pcibios_write_config_word pcibios_write_config_word
#define kernel_pcibios_write_config_dword pcibios_write_config_dword
#define kernel_pcibios_read_config_byte pcibios_read_config_byte
#define kernel_check_region check_region
#define kernel_request_region request_region
#define kernel_release_region release_region
#define kernel_register_chrdev register_chrdev
#define kernel_select_wait select_wait
#define kernel_unregister_chrdev unregister_chrdev


/* Mapping of physical address into kernel space changed from 2.0 to 2.1 */
#if (LINUX_VERSION_CODE < 0x020100)
#define kernel_ioremap vremap
#define kernel_iounmap vfree
#else
#define kernel_ioremap ioremap
#define kernel_iounmap iounmap
#endif

/* Board parameters */
typedef struct _mdsp_info
{
 unsigned char busnr,devnr,irq,irqtype;
 unsigned short class,ready;
 unsigned int boardtype;
 unsigned int base;        /* Base address MDSP registers                   */
 unsigned int basetype;    /* Memory-mapped for DSP56301                    */
 unsigned int hctr,        /* Control Register Address                      */
              hstr,        /* Status Register Address                       */
              hcvr,        /* Host Control Vector Register                  */
              htxr,        /* Transmission Register Address (Input to DSP) */
              hrxr;        /* Receive Register Address   (Output from DSP) */
 struct wait_queue *wait;
} mdsp_info;

static mdsp_info mdsp_board[MDSP_MAX]; /* structs for all boards found */
```

```
/* Static variables */
static int   mdsp_major=MDSP_MAJOR;  /* Major Device Number          */
static char *mdsp_name="mdsp";       /* Name of device               */
static char *status_buf = NULL;      /* /dev/mdspstat info buffer    */
static int   status_len, status_ptr; /* Current buffer params for mdspstat */


static unsigned short mdsp_type[]=     /* Supported DSP board types        */
 { PCI_DEVICE_ID_DSP56301 };           /* Currently only DSP56301, please  */
                                       /* add others                       */

/* Basic register read/write functions. Though the DSP56301 currently      */
/* uses memory-mapped access, other cards might be mapped into io-space,   */
/* so both is supported.                                                   */
static __inline void mdsp_write_reg(unsigned int addr, __u32 val,
                                    unsigned int type)
{
 if(type==PCI_BASE_ADDRESS_SPACE_IO) kernel_outl(val,addr);
 else writel(val,addr);
}

static __inline __u32 mdsp_read_reg(unsigned int addr,unsigned int type)
{
 return (type==PCI_BASE_ADDRESS_SPACE_IO)?kernel_inl(addr)
                                         :readl(addr);
}


/**************************** Status Utilities *****************************\
* State of all DSP56xxx cards may be read from /dev/mdspstat (similar to      *
* /dev/sndstat provided by sound driver).                                     *
\***************************************************************************/
static int put_status (char *s)
{
 int l=strlen(s);
 if(status_len+l>=MAX_STATUS) return 0;
 memcpy(&status_buf[status_len],s,l);
 status_len+=l;
 return 1;
}

static int put_status_int(unsigned int val,int radix)
{
 int l,v;
 static char hx[] = "0123456789abcdef";
 char        buf[11];
 if(!val) return put_status ("0");
 l=0; buf[10]=0;
 while(val)
  {
   v=val%radix; val=val/radix;
   buf[9-l]=hx[v]; l++;
  }
 if (status_len+l>=MAX_STATUS) return 0;
 memcpy(&status_buf[status_len],&buf[10-l],l);
 status_len+=l;
 return 1;
}
```

```
static void init_status (void)
{
 int i;
 char *bl=" ";
 char *cr="\n";
 status_ptr = 0;
 put_status("Kernel: ");
 put_status(system_utsname.sysname);  put_status (bl);
 put_status(system_utsname.nodename); put_status (bl);
 put_status(system_utsname.release);  put_status (bl);
 put_status(system_utsname.version);  put_status (bl);
 put_status(system_utsname.machine);  put_status (cr);
 for(i=0;i<MDSP_MAX;i++)
  {
   if(mdsp_board[i].ready)
    {
     unsigned int base=mdsp_board[i].base;
     unsigned int type=mdsp_board[i].basetype;
     put_status("MDSP ");       put_status_int(i+1,10); put_status(":");
     put_status(" bus 0x");     put_status_int(mdsp_board[i].busnr,16);
     put_status(", device 0x"); put_status_int(mdsp_board[i].devnr,16);
     put_status(", class 0x");  put_status_int(mdsp_board[i].class,16);
     put_status(", address 0x");
     put_status_int(mdsp_board[i].base,16);
     put_status(", irq 0x");    put_status_int(mdsp_board[i].irq,16);
     put_status(", control 0x");
     put_status_int(mdsp_read_reg(base+HCTR,type),16);
     put_status(", status 0x");
     put_status_int(mdsp_read_reg(base+HSTR,type),16);
     put_status(", vector 0x");
     put_status_int(mdsp_read_reg(base+HCVR,type),16);
     put_status(cr);
    }
  }
}

static int read_status (char *buf, int count)
{ /* Return at most 'count' bytes from the status_buf. */
  int l=count,c=status_len-status_ptr;
  if(l>c) l=c;
  if(l<=0) return 0;
  memcpy_tofs(buf,&status_buf[status_ptr],l);
  status_ptr+=l;
  return l;
}

/**************************** Polling **************************************/

/***************************************************************************\
* poll() waits for the given flag to be set in the given register.          *
* returns 0 on success. There is no timeout limit, though the polling may be  *
* terminated on reception of any non-blocked signal.                        *
\***************************************************************************/
static int poll(unsigned int address,__u32 flag,unsigned int type)
{
 for(;;)
  {
   __u32 status=mdsp_read_reg(address,type);
```

```
   unsigned int sig;
   if(status&flag) break;
   /* Did we receive any non-blocked signals? (^C etc.) */
   sig=kernel_var_current->signal&~kernel_var_current->blocked;
   if(sig)
    { /* Handle SIGSTOP etc. */
     DPRINTK("Command aborted by signal 0x%x\n",sig);
     return -ERESTARTSYS;
    }
#ifndef SLOW
   /* Be nice to other processes */
   if(kernel_var_need_resched) kernel_schedule();
#endif
  }
 return 0;
}

/************************* VFS functions **********************************/
static int read_mdsp(struct inode *inode, struct file *file, char *buffer,
                     int count)
{
 int minor=MINOR(inode->i_rdev),boardnr=minor-1;
 if((minor==0)||(minor>0&&minor<=MDSP_MAX&&mdsp_board[boardnr].ready))
  {
   int i,result;
   if(count<=0) return -EINVAL;
   if((result=kernel_verify_area(VERIFY_WRITE,buffer,count))!=0) return result;
   if(minor==0)
    { /* Write status information for all cards */
     return read_status(buffer,count);
    }
   else
    {
     if(!mdsp_board[boardnr].ready) return -ENODEV;
     /* Count must be a multiple of sizeof(__u32) */
     count &= ~(MDSP_INT_LEN-1);
     DPRINTK("Reading %d 32-bit words from DSP.\n",count/sizeof(int));
     for(i=0; i<count; i+=MDSP_INT_LEN) /* 32-bit values */
      {
       __u32 val;
       /* Wait for status flag, then read. HSTR_HRRQ is set by the DSP       */
       /* automatically after write operations to DTXR.                      */
       result=poll(mdsp_board[boardnr].hstr,HSTR_HRRQ,
                   mdsp_board[boardnr].basetype);
       if(result) return i?i:result;    /* return bytes read so far or error */
       val=mdsp_read_reg(mdsp_board[boardnr].hrxr,
                         mdsp_board[boardnr].basetype);
       /* In case of alignment problems, use                               */
       /* kernel_memcpy_tofs(&buffer[i],&val,sizeof(unsigned int));        */
       /* instead of kernel_put_user(val,(__u32 *)&buffer[i]);             */
       kernel_put_user(val,(__u32 *)&buffer[i]);
      }
     return count;
    }
  }
 return -ENODEV;
}

static int write_mdsp(struct inode *inode, struct file *file,
```

```
                      const char *buffer, int count)
{
 int minor=MINOR(inode->i_rdev),boardnr=minor-1;
 if(minor>0&&minor<=MDSP_MAX&&mdsp_board[boardnr].ready)
  {
   int i,result;
   if((result=kernel_verify_area(VERIFY_READ, buffer, count))!=0) return result;
   if(count<sizeof(long)) return -EINVAL;
   /* Count must be a multiple of sizeof(__u32). */
   count &= ~(MDSP_INT_LEN-1);
   DPRINTK("Writing %d 32-bit words to DSP.\n",count/MDSP_INT_LEN);
   for(i=0; i<count; i+=MDSP_INT_LEN) /* 32-bit words */
    {
     __u32 val;
     /* In case of alignment problems, use                               */
     /* kernel_memcpy_fromfs(&val,&buffer[i],sizeof(__u32));             */
     /* instead of val=kernel_get_user((__u32 *)&buffer[i]);             */
     val=kernel_get_user((__u32 *)&buffer[i]);
     /* Wait for status flag in HSTR, then write. HTRQ is set by the DSP */
     /* automatically if there is space in FIFO for another 32-bit word. */
     result=poll(mdsp_board[boardnr].hstr,HSTR_HTRQ,
                 mdsp_board[boardnr].basetype);
     if(result) return i?i:result;  /* return bytes read so far or error */
     mdsp_write_reg(mdsp_board[boardnr].htxr,val,
                    mdsp_board[boardnr].basetype);
    }
   return count;
  }
 return -ENODEV;
}

static int select_mdsp(struct inode *inode, struct file *file, int type,
                       select_table *wait)
{
 int minor=MINOR(inode->i_rdev),boardnr=minor-1;
 if(minor>0&&minor<=MDSP_MAX&&mdsp_board[boardnr].ready)
  {
   int flag;
   for(flag=0;flag<2;flag++)
    { /* try two times, pause in between */
     __u32 status=mdsp_read_reg(mdsp_board[boardnr].hstr,
                                mdsp_board[boardnr].basetype);
     switch(type)
      {
       case SEL_IN:  if(status&HSTR_HRRQ) return 1; break;
       case SEL_OUT: if(status&HSTR_HTRQ) return 1; break;
       case SEL_EX:  if(status&HSTR_TRDY) return 1; break;
       default:      return -EINVAL;
      }
     if(flag==0) kernel_select_wait(&mdsp_board[boardnr].wait,wait);
    }
  }
 else return -ENODEV;
 return 0;
}

static int open_mdsp(struct inode *inode, struct file *file)
{
 int minor=MINOR(inode->i_rdev),boardnr=minor-1;
```

```
 if((minor==0)||(minor>0&&minor<=MDSP_MAX&&mdsp_board[boardnr].ready))
  {
   MOD_INC_USE_COUNT; /* Increment in-use counter */
   if(minor==0)
    {
     if(status_buf||(status_buf=(char *)vmalloc(MAX_STATUS))==NULL) return -EIO;
     status_len=status_ptr=0;
     init_status();
    }
   DPRINTK("Minor %d opened.\n",minor);
   return 0;
  }
 return -ENODEV;
}

static void release_mdsp(struct inode *inode, struct file *file)
{
 int minor=MINOR(inode->i_rdev),boardnr=minor-1;
 if((minor==0)||(minor>0&&minor<=MDSP_MAX&&mdsp_board[boardnr].ready))
  {
   if(minor==0 && status_buf) {vfree(status_buf); status_buf=NULL; }
   MOD_DEC_USE_COUNT; /* Increment in-use counter */
   DPRINTK("Minor %d closed.\n",minor);
  }
}

static int ioctl_mdsp(struct inode *inode, struct file *file, unsigned int cmd,
                      unsigned long param)
{
 int minor=MINOR(inode->i_rdev),boardnr=minor-1,result;
 if(minor==0||_IOC_TYPE(cmd)!=MDSP_MAJOR) return -EINVAL;
 if(minor>0&&minor<=MDSP_MAX&&mdsp_board[boardnr].ready)
  {
   __u32 val,oldval;
   __u32 status;
   unsigned int address=0;
   unsigned int type=mdsp_board[boardnr].basetype;
   int pcicommand=0;
   switch(_IOC_NR(cmd))
    {
     case 'c': address=mdsp_board[boardnr].hctr; break;
     case 's': address=mdsp_board[boardnr].hstr; break;
     case 'r': status=mdsp_read_reg(mdsp_board[boardnr].hstr,type);
               if(!(status&HSTR_HRRQ)) return -EAGAIN;
               address=mdsp_board[boardnr].hrxr;
               break;
     case 't': status=mdsp_read_reg(mdsp_board[boardnr].hstr,type);
               if(!(status&HSTR_HTRQ)) return -EAGAIN;
               address=mdsp_board[boardnr].htxr;
               break;
     case 'i': address=mdsp_board[boardnr].hcvr; break;
     case 'p': pcicommand=1; break;
     default:  return -EINVAL;
    }
   if(_IOC_DIR(cmd)&_IOC_READ)
    {
     if((result=kernel_verify_area(VERIFY_WRITE, (unsigned long *)param,
                                   sizeof(unsigned long)))!=0) return result;
     if(pcicommand) /* Read PCI CCMR/CSTR */
```

```
      {
       int result;
       if((result=kernel_pcibios_read_config_dword(mdsp_board[boardnr].busnr,
                                                  mdsp_board[boardnr].devnr,
               /* dword -> COMMAND and STATUS */  PCI_COMMAND,
                                                  &oldval))) return result;
      }
     else /* Read MDSP register */
       oldval=mdsp_read_reg(address,type);
    }
   if(_IOC_DIR(cmd)&_IOC_WRITE)
    {
     if((result=kernel_verify_area(VERIFY_READ, (unsigned long *)param,
                                   sizeof(unsigned long)))!=0) return result;
     val=(__u32)kernel_get_user((unsigned long *)param);
     if(pcicommand) /* Read PCI CCMR/CSTR */
      {
       int result;
       if((result=kernel_pcibios_write_config_dword(mdsp_board[boardnr].busnr,
                                                   mdsp_board[boardnr].devnr,
                /* dword -> COMMAND and STATUS */  PCI_COMMAND,
                                                   val))) return result;
      }
     else /* Write MDSP register */
       mdsp_write_reg(address,val,type);
    }
   if(_IOC_DIR(cmd)&_IOC_READ) /* write remembered value */
     kernel_put_user((unsigned long)oldval,(unsigned long *)param);
   return 0;
  } /* end if minor>0&&minor<=MDSP_MAX) */
 return -ENODEV;
}

/* Per default, Linux returns a successful lseek if set to NULL in          */
/* file_operations. The purpose of this function is to return a correct     */
/* error message to the caller.                                             */
static int lseek_mdsp(struct inode *inode, struct file *file, off_t offset,
                      int origin)
{
 return -ESPIPE;
}

static struct file_operations fops_mdsp =
{
 lseek_mdsp,              /* dummy lseek()            */
 read_mdsp,
 write_mdsp,
 NULL,                    /* readdir() not implemented */
 select_mdsp,
 ioctl_mdsp,
 NULL,                    /* mmap() not implemented    */
 open_mdsp,
 release_mdsp,
 NULL,                    /* fsync() not implemented   */
 NULL,                    /* fasync() not implemented  */
 NULL,                    /* check_media_change() not implemented */
 NULL                     /* revalidate() not implemented */
};
```

```
/* Board initialization */
static int init_mdsp()
{
 int i,result;
 unsigned int base;

 if(!kernel_pcibios_present())
  {
   PRINTK("No PCI-BIOS present, exiting.\n");
   return -1;
  }
 for(i=0;i<MDSP_MAX;i++)
  {
   mdsp_board[i].ready=0;   /* Flag non-initialized    */
   mdsp_board[i].wait=NULL; /* Zero waitqueue          */
  }
 for(i=0;i<MDSP_MAX;i++)
  {
   int j,found=0;
   /* Find all supported devices */
   for(j=0;j<sizeof(mdsp_type)/sizeof(unsigned short);j++)
    {
     found=(!kernel_pcibios_find_device(PCI_VENDOR_ID_MOTOROLA,mdsp_type[j],
                                   i,&mdsp_board[i].busnr,
                                   &mdsp_board[i].devnr));
     if(found)
      {
       mdsp_board[i].boardtype=mdsp_type[j];
       break;
      }
    }
   if(!found) break;
   if((result=kernel_pcibios_read_config_word(mdsp_board[i].busnr,
                                            mdsp_board[i].devnr,
                                            PCI_CLASS_DEVICE,
                                            &mdsp_board[i].class)))
    {
     PRINTK("Failed to read class type from PCI board, errno=%d\n",result);
     continue;
    }
   if((result=kernel_pcibios_read_config_dword(mdsp_board[i].busnr,
                                             mdsp_board[i].devnr,
                                             PCI_BASE_ADDRESS_0,
                                             &base)))
    {
     PRINTK("Failed to read base from PCI board. errno=%d\n",result);
     continue;
    }
   if((result=kernel_pcibios_read_config_byte(mdsp_board[i].busnr,
                                            mdsp_board[i].devnr,
                                            PCI_INTERRUPT_LINE,
                                            &mdsp_board[i].irq)))
    {
     PRINTK("Failed to read interrupt line from PCI board, errno=%d\n",result);
     continue;
    }
   /* Enable Memory space access and parity error handling */
   kernel_pcibios_write_config_word(mdsp_board[i].busnr,
                                  mdsp_board[i].devnr,
```

```
                                     PCI_COMMAND,
                                     PCI_COMMAND_MEMORY|
                                     PCI_COMMAND_PARITY);
    /* Memory-mapping if necessary, otherwise register IO-space */
    if(base&PCI_BASE_ADDRESS_SPACE) /* Normal IO-Port */
     {
      DPRINTK("ioaddress is already io-mapped.\n");
      if(kernel_check_region(base,MDSP_MEM_SIZE))
       {
        PRINTK("Cannot access IO memory at 0x%x, skipping board.\n",
               (unsigned int)base);
        continue;
       }
      /* Register IO region  to /proc/ioports */
      base&=PCI_BASE_ADDRESS_IO_MASK;
      kernel_request_region(base,MDSP_MEM_SIZE,mdsp_name);
      mdsp_board[i].base=base;
      mdsp_board[i].basetype=PCI_BASE_ADDRESS_SPACE_IO;
     }
    else /* Memory-mapped, must be mapped into Kernel-Space */
     {
      DPRINTK("ioaddress is memory-mapped, mapping into kernel-space.\n");
      base &= PCI_BASE_ADDRESS_MEM_MASK;
      mdsp_board[i].base = (unsigned int) kernel_ioremap(base,MDSP_MEM_SIZE);
      mdsp_board[i].basetype=PCI_BASE_ADDRESS_SPACE_MEMORY;
     }
    /* Precalculate addresses for easier access in functions. */
    mdsp_board[i].hctr=mdsp_board[i].base+HCTR;
    mdsp_board[i].hstr=mdsp_board[i].base+HSTR;
    mdsp_board[i].hcvr=mdsp_board[i].base+HCVR;
    mdsp_board[i].htxr=mdsp_board[i].base+HTXR;
    mdsp_board[i].hrxr=mdsp_board[i].base+HRXR;
    mdsp_board[i].ready=1; /* Everything ok, flag board ready */
    /* Print status of card to console */
    PRINTK("BUS: %d, DEVICE: 0x%x, CLASS: 0x%x, ADDRESS: 0x%x, IRQ: %d\n",
           (int)mdsp_board[i].busnr,(int)mdsp_board[i].devnr,
           (int)mdsp_board[i].class, (int)mdsp_board[i].base,
           (int)mdsp_board[i].irq);
   } /* for i < MDSP_MAX */
 PRINTK("Found %d Motorola DSP56xxx board%s.\n",i,i!=1?"s":"");
 return 0;
}

/* Module initialization at loading time of driver */
int init_module(void)
{
 int result;
 if(init_mdsp())
  {
   PRINTK("No DSP boards found.\n");
   return -EIO;
  }
 /* Register file operations for mdsp device */
 if((result=kernel_register_chrdev(mdsp_major, mdsp_name, &fops_mdsp))==-EBUSY)
  {
   PRINTK("Unable to get major %d for MDSP device, errno=%d.\n",mdsp_major,
          result);
   return -EIO;
  }
```

```
 PRINTK("Registered Device %s, major %d.\n", mdsp_name, mdsp_major);
 return 0;
}

/* Cleanup before unloading module, release io or unmap memory space */
static void cleanup_mdsp(void)
{
 int i;
 for(i=0;i<MDSP_MAX;i++)
  {
   if(mdsp_board[i].ready)
    {
     if(mdsp_board[i].basetype==PCI_BASE_ADDRESS_SPACE_IO)
       kernel_release_region((unsigned int)mdsp_board[i].base,MDSP_MEM_SIZE);
     else
       kernel_iounmap((void *)mdsp_board[i].base);
    }
   mdsp_board[i].ready=0;
  }
}

void cleanup_module(void)
{
 kernel_unregister_chrdev(mdsp_major,mdsp_name);
 cleanup_mdsp();
 PRINTK("Unregistered Device %s, major %d.\n", mdsp_name, mdsp_major);
}
```

Quelltext 40: `mdsp.c`

Utility `mdsptest` – mdsptest.c

```
/* Test for Motorola DSP driver and benchmark program. */
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <linux/fs.h>
#include "mdsp.h"

/* Simple DSP program, reads from and writes to HI32 in a loop */
unsigned long prog[]={
0x00000009, /* Length */
0x00000100, /* Address */
0x00200013, /* DSP prog */
0x000cc902,
0x00000003,
0x00084e0b,
0x000cc901,
0x00fffffd,
0x0008ce0d,
0x00014180,
0x000c0101};

void success(int yes)
{
 printf(yes?"success.\n":"failed.\n");
}

#define BUFSIZE 0x10000
int main(int argc, char **argv)
{
 const struct timeval tc={10,0};
 struct timeval tv=tc,tv2;
 fd_set fds;
 unsigned long val,oldval,totalbytes,msecs;
 int result;
 int fd;
 int i;
 struct stat fdstat;
 char *name;
 __u32  buffer[BUFSIZE];
 if(argc<2||!strcmp(argv[1],"-h"))
  {
   fprintf(stderr,"Usage: %s /dev/mdsp[1-16]\n",argv[0]);
   return 1;
  }
 name=argv[1];
 if((result=stat(name,&fdstat))!=0)
  {
   fprintf(stderr,"Error: '%s' does not exist, errno=%d\n",name,errno);
   return 2;
  }
 if(MAJOR(fdstat.st_rdev)!=MDSP_MAJOR)
  {
   fprintf(stderr,
      "Error: '%s' is not a  Motorola DSP56xxx device file (major != %d)\n",
```

```
       name,MAJOR(fdstat.st_rdev));
    return 3;
   }
  printf("Device file '%s': Major %d, Minor %d.\n",name,
         MAJOR(fdstat.st_rdev),MINOR(fdstat.st_rdev));
  if((fd=open(argv[1],O_RDWR))<0)
   {
    fprintf(stderr,
       "Error: Can't open '%s' in read/write mode.\n",name);
    return 3;
   }
  if(ioctl(fd,MDSPCTL_READ_HSTR,&val))
   {
    fprintf(stderr,"ioctl() failed.");
    close(fd);
    return 4;
   }
  printf("Status register: 0x%lx\n",val);
  ioctl(fd,MDSPCTL_READ_HCTR,&val);
  printf("Control register: 0x%lx\n",val);
  if(val&HCTR_HF0)
   {
    printf("WARNING: HF0 is set, so there will be no upload of test program.\n"
           "Hope the right program is already running!\n");
   }
  else
   {
    printf("Trying to set control register for upload: "); fflush(stdout);
    oldval=val;
    val=((val&~HCTR_HTF1)&~HCTR_HF0)|HCTR_HTF0; /* 3-Byte transfer mode */
    ioctl(fd,MDSPCTL_WRITE_HCTR,&val);
    val=0;
    ioctl(fd,MDSPCTL_READ_HCTR,&val);
    success(val!=0);
    printf("Uploading program..."); fflush(stdout);
    success(write(fd,(char *)prog,sizeof(prog))==sizeof(prog));
    val=(val&(~HCTR_HRF1))|HCTR_HRF0|HCTR_HF0; /* Force start of program */
    ioctl(fd,MDSPCTL_WRITE_HCTR,&val);
    ioctl(fd,MDSPCTL_READ_HSTR,&val);
    if(val&HSTR_HRRQ) printf("Program running.\n");
   }
  printf("Select()ing DSP for read (10s)..."); fflush(stdout);
  FD_ZERO(&fds);
  FD_SET(fd,&fds);
  result=select(FD_SETSIZE,&fds,NULL,NULL,&tv);
  printf("returned after %d sec, %d usec, status=%d\n",tc.tv_sec-tv.tv_sec,
                                                      tc.tv_usec-tv.tv_usec,
                                                      result);
  printf("Select()ing DSP for write (10s)..."); fflush(stdout);
  tv=tc;
  FD_ZERO(&fds);
  FD_SET(fd,&fds);
  result=select(FD_SETSIZE,NULL,&fds,NULL,&tv);
  printf("returned after %d sec, %d usec, status=%d\n",tc.tv_sec-tv.tv_sec,
                                                      tc.tv_usec-tv.tv_usec,
                                                      result);
  printf("Benchmarking read()..."); fflush(stdout);
  gettimeofday(&tv,NULL);
  for(i=0,totalbytes=0;i<0x00ff;i++)
```

```
  {
   if((result=read(fd,buffer,sizeof(buffer)))!=sizeof(buffer))
    {
     fprintf(stderr,"Error in read, only %d of %d bytes read.\n",
             result,sizeof(buffer));
    }
   totalbytes+=result;
  }
 gettimeofday(&tv2,NULL);
 msecs=(tv2.tv_sec-tv.tv_sec)*1000+(tv2.tv_usec-tv.tv_usec)/1000;
 printf("Read %lu bytes in %d.%03d secs.\nThroughput was %d Bytes/sec.\n",
        (unsigned long)totalbytes,(int)(msecs/1000),(int)(msecs%1000),
        (int)(totalbytes/(msecs/1000)));
 printf("Benchmarking write()..."); fflush(stdout);
 gettimeofday(&tv,NULL);
 for(i=0,totalbytes=0;i<0x00ff;i++)
  {
   if((result=write(fd,buffer,sizeof(buffer)))!=sizeof(buffer))
    {
     fprintf(stderr,"Error in write, only %d of %d bytes written.\n",
             result,sizeof(buffer));
    }
   totalbytes+=result;
  }
 gettimeofday(&tv2,NULL);
 msecs=(tv2.tv_sec-tv.tv_sec)*1000+(tv2.tv_usec-tv.tv_usec)/1000;
 printf("Wrote %lu bytes in %d.%03d secs.\nThroughput was %d Bytes/sec.\n",
        (unsigned long)totalbytes,(int)(msecs/1000),(int)(msecs%1000),
        (int)(totalbytes/(msecs/1000)));
 close(fd);
 return 0;
}
```

Quelltext 41: `mdsptest.c`

Utility `mdsptool` – mdsptool.c

```
/**************************************************************************\
 * Example program for use with the MDSP driver. Can be used for          *
 * uploading a DSP-program in PCI bootstrap mode or for reading/writing   *
 * of registers of the DSP56301 board.                                    *
 * Copyright (C) by Klaus Knopper, March 1998                             *
\**************************************************************************/

#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <fcntl.h>
#include <linux/pci.h>
#include "mdsp.h" /* Macros, addresses, features */

#define VERSION "MDSPTOOL (C) Klaus Knopper, March 1998"

int usage(int argc, char **argv, char *msg)
{
 const char *help=
 VERSION"\n"
 "Usage: %s options\nList of options:\n"
 "  -#            Apply following command to DSP-Card number # (default 1)\n"
 "  -a            Print information for all MDSP boards from /dev/mdspstat\n"
 "  -c            Print contents of host control register\n"
 "  -C 0xnnnnnnnn Write hex number nnnnnnnn to host control register\n"
 "  -d            Dump data from DSP (forever loop)\n"
 "  -D 0xnnnnnnnn Read 0xn data words from DSP\n"
 "  -f file       Upload hexdump file in bootstrap mode (first thing to do\n"
 "                after a reset, see documentation for file format.\n"
 "                Use - for stdin and make sure that HCTR_HF0 is cleared.)\n"
 "  -i            Print contents of host vector control register\n"
 "  -I 0xnnnnnnnn Write hex number nnnnnnnn to host vector control register\n"
 "  -p            Print contents of pci control register\n"
 "  -P 0xnnnnnnnn Write hex number nnnnnnnn to pci control register\n"
 "  -t            Print contents of data transfer register immediate\n"
 "  -T 0xnnnnnnnn Write hex number nnnnnnnn to data transfer register "
 "immediate\n"
 "  -s            Print contents of host statusregister\n"
 "  -S 0xnnnnnnnn Write hex number nnnnnnnn to host status register\n"
 "  -v            Be verbose (print decoded values of registers)\n";
 fprintf(stderr,help,argv[0]);
 if(msg)
  {
   fprintf(stderr,"\nBad option: %s\n",msg);
   return -1;
  }
 return 0;
}

/* Opens MDSP-device with the given ID n, returns result of open() */
int mreopen(int fd,int n,int verbose)
{
 const char *common="/dev/mdsp";
 char mname[32]="";
```

```
 if(fd>=0) close(fd);
 sprintf(mname,"%s%d",common,n);
 if((fd=open(mname,O_RDWR))<0)
   fprintf(stderr,"\nERROR: cannot open MDSP device '%s': %s\n",
           mname,sys_errlist[errno]);
 return(fd);
}

unsigned long parse_hex(char *string,__u32 *valptr)
{
 int result;
 unsigned long val;
 if((result=sscanf(string,"0x%lx",&val))!=1)
  {
   fprintf(stderr,"\nERROR: %s is not a hex number (like '0xfefefefe')\n",
           string);
  }
 *valptr=(__u32) val;
 return result;
}

int printreg(int fd,unsigned int cmd,int verbose)
{
 unsigned long val;
 if(ioctl(fd,cmd,&val))
  {
   perror("Can't ioctl()");
   return -1;
  }
 /* print it. */
 printf("0x%lx",val);
 if(verbose)
  {
   switch(cmd)
    {
     case MDSPCTL_READ_HSTR:
       if(val&HSTR_TRDY) printf(" HSTR_TRDY");
       if(val&HSTR_HTRQ) printf(" HSTR_HTRQ");
       if(val&HSTR_HRRQ) printf(" HSTR_HRRQ");
       if(val&HSTR_HF3)  printf(" HSTR_HF3");
       if(val&HSTR_HF4)  printf(" HSTR_HF4");
       if(val&HSTR_HF5)  printf(" HSTR_HF5");
       if(val&HSTR_HINT) printf(" HSTR_HINT");
       if(val&HSTR_HREQ) printf(" HSTR_HREQ");
       break;
     case MDSPCTL_READ_HCTR:
       if(val&HCTR_TREQ) printf(" HCTR_TREQ");
       if(val&HCTR_RREQ) printf(" HCTR_RREQ");
       if(val&HCTR_HF0)  printf(" HCTR_HF0");
       if(val&HCTR_HF1)  printf(" HCTR_HF1");
       if(val&HCTR_HF2)  printf(" HCTR_HF2");
       if(val&HCTR_DMAE) printf(" HCTR_DMAE");
       if(val&HCTR_SFT)  printf(" HCTR_SFT");
       if(val&HCTR_HTF0) printf(" HCTR_HTF0");
       if(val&HCTR_HTF1) printf(" HCTR_HTF1");
       if(val&HCTR_HRF0) printf(" HCTR_HRF0");
       if(val&HCTR_HRF1) printf(" HCTR_HRF1");
       if(val&HCTR_HS0)  printf(" HCTR_HS0");
       if(val&HCTR_HS1)  printf(" HCTR_HS1");
```

```
       if(val&HCTR_HS2)  printf(" HCTR_HS2");
       if(val&HCTR_TWSD) printf(" HCTR_TWSD");
       break;
    }
  }
 printf("\n");
 return 0;
}

int writereg(int fd,unsigned long val,unsigned int cmd,int verbose)
{
 if(ioctl(fd,cmd,&val))
  {
   perror("Can't ioctl()");
   return -1;
  }
 return 0;
}

/* mupload() reads Hexdump, writes 32-bit values to the transfer register  */
/* of the DSP board. The byte order is automatically corrected to match    */
/* the byte order of the DSP when accessing PCI registers.                 */
int mupload(int fd,char *fname,int verbose)
{
 FILE *in;
 unsigned long val;
 unsigned long control;
 if(fname==NULL||fd<0)
  {
   fprintf(stderr,"\nInvalid argument to upload(): FD=%d, fname='%s'\n",
           fd,fname?fname:"(null)");
   return -1;
  }
 in=(fname[0]=='-'&&fname[1]==0)?stdin:fopen(fname,"r");
 if(in==NULL)
  {
   fprintf(stderr,"\nERROR: Can't open file '%s' for upload.\n",
           fname);
   return -1;
  }
 /* Set Host Transfer Mode 0 */
 /* Read control register */
 if(ioctl(fd,MDSPCTL_READ_HCTR,&control))
  {
   perror("Can't ioctl()");
   if(in!=stdin) fclose(in);
   return -1;
  }
 /* Clear HTF1, set HTF0 -> 3 Byte in-a-block transfer   */
 /* Clear HF0! (HF0 signals end of upload)               */
 control=((control&~HCTR_HTF1)&~HCTR_HF0)|HCTR_HTF0;
 ioctl(fd,MDSPCTL_WRITE_HCTR,&control);
 /* I wish we could reset the DSP before doing this. :(  */
 while(!feof(in)&&!ferror(in)&&fscanf(in,"%lx",&val)==1)
  {
   if(write(fd,&val,4)!=4)
    {
     fprintf(stderr,"\nERROR: Can't write to DSP.\n");
     if(in!=stdin) fclose(in);
```

```
     return -1;
    }
  }
 if(in!=stdin) fclose(in);
 /* Force start of program (set HF0, Bootstrap loader will finish download */
 /* and start program immediately)                                         */
 control=(control&(~HCTR_HRF1))|HCTR_HRF0|HCTR_HF0;
 ioctl(fd,MDSPCTL_WRITE_HCTR,&control);
 return 0;
}

int printstatus(void)
{
 int fd;
 const char *sname="/dev/mdspstat";
 char buffer[4096];
 int bytes;
 if((fd=open(sname,O_RDONLY))<=0) return -1;
 if((bytes=read(fd,buffer,4096))<=0)
  {
   close(fd);
   return -1;
  }
 close(fd);
 fwrite(buffer,bytes,1,stdout);
 return 0;
}

int main(int argc,char **argv)
{
 int mnum=1,mfd=-1,    /* DSP Device # and file descriptor */
     verbose=0,result,i;

 if(argc<2||argv[1][1]=='h') return(usage(argc,argv,NULL));
 for(i=1;i<argc;i++)
  {
   __u32 val; int max=0,j;
   if(argv[i][0]!='-') return(usage(argc,argv,argv[i]));
   switch(argv[i][1])
    {
     case  0 : break;
     case 'a': if(printstatus())
                {
                 fprintf(stderr,"Can't read /dev/mdspstat.\n");
                 return errno;
                }
               break;
     case 'v': verbose=1; break;
     case 'c': if(mfd<0&&(mfd=mreopen(mfd,mnum,verbose))<0) return errno;
               if(printreg(mfd,MDSPCTL_READ_HCTR,verbose)) return errno;
               break;
     case 'C': if(mfd<0&&(mfd=mreopen(mfd,mnum,verbose))<0) return errno;
               ++i;
               if(i==argc||argv[i]==NULL||parse_hex(argv[i],&val)!=1)
                 return(usage(argc,argv,argv[i]));
               if(writereg(mfd,val,MDSPCTL_WRITE_HCTR,verbose)) return errno;
               break;
     case 'D': ++i;
               if(i==argc||argv[i]==NULL||parse_hex(argv[i],&val)!=1)
```

```
                return(usage(argc,argv,argv[i]));
              max=(int)val; /* NO break here. */
    case 'd': if(mfd<0&&(mfd=mreopen(mfd,mnum,verbose))<0) return errno;
              for(j=0;(max==0)||j<max;j++)
               {
                __u32 data;
                if(read(mfd,&data,sizeof(__u32))!=sizeof(__u32)) return errno;
                printf("0x%x\n",(unsigned int)data);
               }
              break;
    case 'f': if(mfd<0&&(mfd=mreopen(mfd,mnum,verbose))<0) return errno;
              if(i==argc) return(usage(argc,argv,argv[i]));
              if((result=mupload(mfd,argv[++i],verbose))) return result;
              break;
    case 'i': if(mfd<0&&(mfd=mreopen(mfd,mnum,verbose))<0) return errno;
              if(printreg(mfd,MDSPCTL_READ_HCVR,verbose)) return errno;
              break;
    case 'I': if(mfd<0&&(mfd=mreopen(mfd,mnum,verbose))<0) return errno;
              ++i;
              if(i==argc||argv[i]==NULL||parse_hex(argv[i],&val)!=1)
                return(usage(argc,argv,argv[i]));
              if(writereg(mfd,val,MDSPCTL_WRITE_HCVR,verbose)) return errno;
              break;
    case 's': if(mfd<0&&(mfd=mreopen(mfd,mnum,verbose))<0) return errno;
              if(printreg(mfd,MDSPCTL_READ_HSTR,verbose)) return errno;
              break;
    case 'S': if(mfd<0&&(mfd=mreopen(mfd,mnum,verbose))<0) return errno;
              ++i;
              if(i==argc||argv[i]==NULL||parse_hex(argv[i],&val)!=1)
                return(usage(argc,argv,argv[i]));
              if(writereg(mfd,val,MDSPCTL_WRITE_HSTR,verbose)) return errno;
              break;
    case 't': if(mfd<0&&(mfd=mreopen(mfd,mnum,verbose))<0) return errno;
              if(printreg(mfd,MDSPCTL_READ_HRXR,verbose)) return errno;
              break;
    case 'T': if(mfd<0&&(mfd=mreopen(mfd,mnum,verbose))<0) return errno;
              ++i;
              if(i==argc||argv[i]==NULL||parse_hex(argv[i],&val)!=1)
                return(usage(argc,argv,argv[i]));
              if(writereg(mfd,val,MDSPCTL_WRITE_HTXR,verbose)) return errno;
              break;
    case 'p': if(mfd<0&&(mfd=mreopen(mfd,mnum,verbose))<0) return errno;
              if(ioctl(mfd,MDSPCTL_READ_CCMR,&val)) return errno;
              printf("0x%lx ",(unsigned long)val);
              if(verbose) /* print decoded values, see linux/pci.h */
               {
                if(val&PCI_COMMAND_IO)
                  printf(" PCI_COMMAND_IO");
                if(val&PCI_COMMAND_MEMORY)
                  printf(" PCI_COMMAND_MEMORY");
                if(val&PCI_COMMAND_MASTER)
                  printf(" PCI_COMMAND_MASTER");
                if(val&PCI_COMMAND_SPECIAL)
                  printf(" PCI_COMMAND_SPECIAL");
                if(val&PCI_COMMAND_INVALIDATE)
                  printf(" PCI_COMMAND_INVALIDATE");
                if(val&PCI_COMMAND_VGA_PALETTE)
                  printf(" PCI_COMMAND_VGA_PALETTE");
                if(val&PCI_COMMAND_PARITY)
```

```
                   printf(" PCI_COMMAND_PARITY");
                 if(val&PCI_COMMAND_WAIT)
                   printf(" PCI_COMMAND_WAIT");
                 if(val&PCI_COMMAND_SERR)
                   printf(" PCI_COMMAND_SERR");
                 if(val&PCI_COMMAND_FAST_BACK)
                   printf(" PCI_COMMAND_FAST_BACK");
                 if(val&PCI_STATUS_66MHZ<<16L)
                   printf(" PCI_STATUS_66MHZ");
                 if(val&PCI_STATUS_UDF<<16L)
                   printf(" PCI_STATUS_UDF");
                 if(val&PCI_STATUS_FAST_BACK<<16L)
                   printf(" PCI_STATUS_FAST_BACK");
                 if(val&PCI_STATUS_PARITY<<16L)
                   printf(" PCI_STATUS_PARITY");
                 if(val&PCI_STATUS_DEVSEL_FAST<<16L)
                   printf(" PCI_STATUS_DEVSEL_FAST");
                 if(val&PCI_STATUS_DEVSEL_MEDIUM<<16L)
                   printf(" PCI_STATUS_DEVSEL_MEDIUM");
                 if(val&PCI_STATUS_DEVSEL_SLOW<<16L)
                   printf(" PCI_STATUS_DEVSEL_SLOW");
                 if(val&PCI_STATUS_SIG_TARGET_ABORT<<16L)
                   printf(" PCI_STATUS_SIG_TARGET_ABORT");
                 if(val&PCI_STATUS_REC_TARGET_ABORT<<16L)
                   printf(" PCI_STATUS_REC_TARGET_ABORT");
                 if(val&PCI_STATUS_REC_MASTER_ABORT<<16L)
                   printf(" PCI_STATUS_REC_MASTER_ABORT");
                 if(val&PCI_STATUS_SIG_SYSTEM_ERROR<<16L)
                  printf(" PCI_STATUS_SIG_SYSTEM_ERROR");
                 if(val&PCI_STATUS_DETECTED_PARITY<<16L)
                   printf(" PCI_STATUS_DETECTED_PARITY");
                } /* end if verbose */
               printf("\n");
               break;
     case 'P': if(mfd<0&&(mfd=mreopen(mfd,mnum,verbose))<0) return errno;
               ++i;
               if(i==argc||argv[i]==NULL||parse_hex(argv[i],&val)!=1)
                 return(usage(argc,argv,argv[i]));
               if(ioctl(mfd,MDSPCTL_WRITE_CCMR,&val)) return errno;
               break;
     default : if((mnum=atoi(&argv[i][1]))==0)
                 return(usage(argc,argv,argv[i]));
               if((mfd=mreopen(mfd,mnum,verbose))<0) return errno;
    }
  }
 if(mfd>=0) close(mfd);
 return 0;
}
```

Quelltext 42: mdsptool.c

Utility `xmdsptool` – xinterface.fd

`xinterface.fd` wird durch das Xforms Designertool `fdesign` erzeugt und beschreibt das zu erzeugende graphische Benutzerinterface.

```
Magic: 13000

Internal Form Definition File
    (do not change)

Number of forms: 1
Unit of measure: FL_COORD_PIXEL

=============== FORM ===============
Name: xmdsptool
Width: 820
Height: 370
Number of Objects: 19

--------------------
class: FL_BOX
type: UP_BOX
box: 0 0 820 370
boxtype: FL_UP_BOX
colors: FL_COL1 FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label:
shortcut:
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name:
callback:
argument:

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 10 10 80 20
boxtype: FL_UP_BOX
colors: FL_TOMATO FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Quit
shortcut: Qq^C^[
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: quit
callback: callback
argument: 0

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 100 40 80 20
boxtype: FL_UP_BOX
colors: FL_PALEGREEN FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Reset DSP
shortcut: eE
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: reset
callback: callback
argument: 12

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 100 10 80 20
boxtype: FL_UP_BOX
colors: FL_COL1 FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Status All
shortcut: Aa
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: statusall
callback: callback
argument: 3

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 190 10 80 20
boxtype: FL_UP_BOX
colors: FL_COL1 FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Print Control
shortcut: c
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: printcontrol
callback: callback
```

```
argument: 4

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 190 40 80 20
boxtype: FL_UP_BOX
colors: FL_COL1 FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Set Control
shortcut: C
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: setcontrol
callback: callback
argument: 5

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 280 10 80 20
boxtype: FL_UP_BOX
colors: FL_COL1 FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Print Status
shortcut: s
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: printstatus
callback: callback
argument: 6

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 280 40 80 20
boxtype: FL_UP_BOX
colors: FL_COL1 FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Set Status
shortcut: S
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: setstatus
callback: callback
argument: 7

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 370 10 80 20
boxtype: FL_UP_BOX
colors: FL_COL1 FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Print Vector
shortcut: vi
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: printvector
callback: callback
argument: 8

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 370 40 80 20
boxtype: FL_UP_BOX
colors: FL_COL1 FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Set Vector
shortcut: VI
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: setvector
callback: callback
argument: 9

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 460 10 80 20
boxtype: FL_UP_BOX
colors: FL_COL1 FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Print PCI Reg
shortcut: p
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: printpci
callback: callback
argument: 10

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 460 40 80 20
boxtype: FL_UP_BOX
colors: FL_COL1 FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
```

```
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Set PCI Reg
shortcut: P
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: setpci
callback: callback
argument: 11

--------------------
class: FL_BROWSER
type: MULTI_BROWSER
box: 10 70 800 290
boxtype: FL_DOWN_BOX
colors: FL_WHITE FL_YELLOW
alignment: FL_ALIGN_BOTTOM
style: FL_BOLD_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label:
shortcut:
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: text
callback:
argument:

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 550 40 80 20
boxtype: FL_UP_BOX
colors: FL_COL1 FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Write Word
shortcut: WwT
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: write
callback: callback
argument: 13

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 550 10 80 20
boxtype: FL_UP_BOX
colors: FL_COL1 FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Read Word
shortcut: rt
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: read
callback: callback
argument: 14

--------------------
class: FL_CLOCK
type: DIGITAL_CLOCK
box: 750 10 60 20
boxtype: FL_FRAME_BOX
colors: FL_LEFT_BCOL FL_BLACK
alignment: FL_ALIGN_BOTTOM
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label:
shortcut:
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: clock
callback:
argument:

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 640 10 80 20
boxtype: FL_UP_BOX
colors: FL_COL1 FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Dump Data
shortcut: Dd
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: dump
callback: callback
argument: 15

--------------------
class: FL_BUTTON
type: NORMAL_BUTTON
box: 640 40 170 20
boxtype: FL_UP_BOX
colors: FL_COL1 FL_COL1
alignment: FL_ALIGN_CENTER
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label: Upload File
shortcut: FfUu
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: upload
callback: callback
argument: 2
```

```
--------------------
class: FL_CHOICE
type: NORMAL_CHOICE
box: 10 40 80 20
boxtype: FL_SHADOW_BOX
colors: FL_TOP_BCOL FL_BLACK
alignment: FL_ALIGN_LEFT
style: FL_NORMAL_STYLE
size: FL_DEFAULT_SIZE
lcol: FL_BLACK
label:
shortcut: /
resize: FL_RESIZE_ALL
gravity: FL_NoGravity FL_NoGravity
name: setdevice
callback: callback
argument: 1
  content: /dev/mdsp1
  shortcut: 1
  content: /dev/mdsp2
  shortcut: 2
  content: /dev/mdsp3
  shortcut: 3
  content: /dev/mdsp4
  shortcut: 4
  content: /dev/mdsp5
  shortcut: 5
  content: /dev/mdsp6
  shortcut: 6
  content: /dev/mdsp7
  shortcut: 7
  content: /dev/mdsp8
  shortcut: 8
  content: /dev/mdsp9
  shortcut: 9
  content: /dev/mdsp10
  content: /dev/mdsp11
  content: /dev/mdsp12
  content: /dev/mdsp13
  content: /dev/mdsp14
  content: /dev/mdsp15
  content: /dev/mdsp16
  value: 1

==============================
create_the_forms
```

Quelltext 43: `xinterface.fd`

Utility `xmdsptool` – xinterface.c

`xinterface.c` wird von `fdesign` aus der im vorigen Abschnitt angegebenen Beschreibungsdatei generiert.

```
/* Form definition file generated with fdesign. */

#include "forms.h"
#include <stdlib.h>
#include "xinterface.h"

FD_xmdsptool *create_form_xmdsptool(void)
{
  FL_OBJECT *obj;
  FD_xmdsptool *fdui = (FD_xmdsptool *) fl_calloc(1, sizeof(*fdui));

  fdui->xmdsptool = fl_bgn_form(FL_NO_BOX, 820, 370);
  obj = fl_add_box(FL_UP_BOX,0,0,820,370,"");
  fdui->quit = obj = fl_add_button(FL_NORMAL_BUTTON,10,10,80,20,"Quit");
    fl_set_button_shortcut(obj,"Qq^C^[",1);
    fl_set_object_color(obj,FL_TOMATO,FL_COL1);
    fl_set_object_callback(obj,callback,0);
  fdui->reset = obj = fl_add_button(FL_NORMAL_BUTTON,100,40,80,20,"Reset DSP");
    fl_set_button_shortcut(obj,"eE",1);
    fl_set_object_color(obj,FL_PALEGREEN,FL_COL1);
    fl_set_object_callback(obj,callback,12);
  fdui->statusall = obj = fl_add_button(FL_NORMAL_BUTTON,100,10,80,20,
                                      "Status All");
    fl_set_button_shortcut(obj,"Aa",1);
    fl_set_object_callback(obj,callback,3);
  fdui->printcontrol = obj = fl_add_button(FL_NORMAL_BUTTON,190,10,80,20,
                                         "Print Control");
    fl_set_button_shortcut(obj,"c",1);
    fl_set_object_callback(obj,callback,4);
  fdui->setcontrol = obj = fl_add_button(FL_NORMAL_BUTTON,190,40,80,20,
                                       "Set Control");
    fl_set_button_shortcut(obj,"C",1);
    fl_set_object_callback(obj,callback,5);
  fdui->printstatus = obj = fl_add_button(FL_NORMAL_BUTTON,280,10,80,20,
                                        "Print Status");
    fl_set_button_shortcut(obj,"s",1);
    fl_set_object_callback(obj,callback,6);
  fdui->setstatus = obj = fl_add_button(FL_NORMAL_BUTTON,280,40,80,20,
                                      "Set Status");
    fl_set_button_shortcut(obj,"S",1);
    fl_set_object_callback(obj,callback,7);
  fdui->printvector = obj = fl_add_button(FL_NORMAL_BUTTON,370,10,80,20,
                                        "Print Vector");
    fl_set_button_shortcut(obj,"vi",1);
    fl_set_object_callback(obj,callback,8);
  fdui->setvector = obj = fl_add_button(FL_NORMAL_BUTTON,370,40,80,20,
                                      "Set Vector");
    fl_set_button_shortcut(obj,"VI",1);
    fl_set_object_callback(obj,callback,9);
  fdui->printpci = obj = fl_add_button(FL_NORMAL_BUTTON,460,10,80,20,
                                     "Print PCI Reg");
    fl_set_button_shortcut(obj,"p",1);
```

```
    fl_set_object_callback(obj,callback,10);
  fdui->setpci = obj = fl_add_button(FL_NORMAL_BUTTON,460,40,80,20,
                                    "Set PCI Reg");
    fl_set_button_shortcut(obj,"P",1);
    fl_set_object_callback(obj,callback,11);
  fdui->text = obj = fl_add_browser(FL_MULTI_BROWSER,10,70,800,290,"");
    fl_set_object_color(obj,FL_WHITE,FL_YELLOW);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE);
  fdui->write = obj = fl_add_button(FL_NORMAL_BUTTON,550,40,80,20,"Write Word");
    fl_set_button_shortcut(obj,"WwT",1);
    fl_set_object_callback(obj,callback,13);
  fdui->read = obj = fl_add_button(FL_NORMAL_BUTTON,550,10,80,20,"Read Word");
    fl_set_button_shortcut(obj,"rt",1);
    fl_set_object_callback(obj,callback,14);
  fdui->clock = obj = fl_add_clock(FL_DIGITAL_CLOCK,750,10,60,20,"");
    fl_set_object_boxtype(obj,FL_FRAME_BOX);
    fl_set_object_color(obj,FL_LEFT_BCOL,FL_BLACK);
  fdui->dump = obj = fl_add_button(FL_NORMAL_BUTTON,640,10,80,20,"Dump Data");
    fl_set_button_shortcut(obj,"Dd",1);
    fl_set_object_callback(obj,callback,15);
  fdui->upload = obj = fl_add_button(FL_NORMAL_BUTTON,640,40,170,20,
                                    "Upload File");
    fl_set_button_shortcut(obj,"FfUu",1);
    fl_set_object_callback(obj,callback,2);
  fdui->setdevice = obj = fl_add_choice(FL_NORMAL_CHOICE,10,40,80,20,"");
    fl_set_object_shortcut(obj,"/",1);
    fl_set_object_boxtype(obj,FL_SHADOW_BOX);
    fl_set_object_color(obj,FL_TOP_BCOL,FL_BLACK);
    fl_set_object_callback(obj,callback,1);
    fl_addto_choice(obj, "/dev/mdsp1");
    fl_set_choice_item_shortcut(obj, 1, "1");
    fl_addto_choice(obj, "/dev/mdsp2");
    fl_set_choice_item_shortcut(obj, 2, "2");
    fl_addto_choice(obj, "/dev/mdsp3");
    fl_set_choice_item_shortcut(obj, 3, "3");
    fl_addto_choice(obj, "/dev/mdsp4");
    fl_set_choice_item_shortcut(obj, 4, "4");
    fl_addto_choice(obj, "/dev/mdsp5");
    fl_set_choice_item_shortcut(obj, 5, "5");
    fl_addto_choice(obj, "/dev/mdsp6");
    fl_set_choice_item_shortcut(obj, 6, "6");
    fl_addto_choice(obj, "/dev/mdsp7");
    fl_set_choice_item_shortcut(obj, 7, "7");
    fl_addto_choice(obj, "/dev/mdsp8");
    fl_set_choice_item_shortcut(obj, 8, "8");
    fl_addto_choice(obj, "/dev/mdsp9");
    fl_set_choice_item_shortcut(obj, 9, "9");
    fl_addto_choice(obj, "/dev/mdsp10");
    fl_addto_choice(obj, "/dev/mdsp11");
    fl_addto_choice(obj, "/dev/mdsp12");
    fl_addto_choice(obj, "/dev/mdsp13");
    fl_addto_choice(obj, "/dev/mdsp14");
    fl_addto_choice(obj, "/dev/mdsp15");
    fl_addto_choice(obj, "/dev/mdsp16");
   fl_set_choice(obj,1);
  fl_end_form();

  fdui->xmdsptool->fdui = fdui;
```

```
  return fdui;
}
/*---------------------------------------*/
```

Quelltext 44: `xinterface.c`

Utility `xmdsptool` – functions.c

`functions.c` enthält die Callback-Prozeduren des graphischen Benutzerinterfaces und die Wrapper-Funktionen für das Backend-Programm `mdsptool`.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/types.h>
#include <asm/types.h> /* __u32 */
#include <sys/time.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include "forms.h"
#include "xinterface.h"
#include "mdsp.h"

#define BUFSIZE 4096
#define CMDSIZE 1024
#define MDSPTOOL "mdsptool" /* Must be in $PATH */
#define MAX_MDSP 16

/* from main.c */
extern FD_xmdsptool *xmd;
extern FL_OBJECT    *textbox;


/* Create command subprocess, redirect output */
int subprocess(char *cmd, int *out) /* returns PID of subprocess */
{
 char *argv[16],*p;
 int i,pid;
 for(i=0,p=cmd;;)
  {
   argv[i++]=p;
   p=strchr(p, ' ');
   if(p==NULL)break;
   *p++=0;
  }
 argv[i] = NULL;
 if((pid=fork())==0)
  { /* this is the child process */
   dup2(out[1],1); /* let out[1] be child processes stdout */
   dup2(1,2); /* force stderr to the pipe */
   execvp(argv[0], argv);
   perror(argv[0]);
   exit(1);
  }
 return pid;
}

/* Writes output of subprocess to textbox and retuns char buffer of current */
/* output */
char *mdsptool(int n,char *args)
```

```
{
 const char *command=MDSPTOOL;
 const char *cr="\n";
 char com[CMDSIZE];
 static char buf[BUFSIZE];
 size_t bytes=0;
 const struct timeval tc={1,0};
 struct timeval tv=tc;
 int fd=-1;
 int sfd[2];
 int pid;
 int result=0;
 int pstatus=0;
 int timeout;
 fd_set fds;

 sprintf(com,"%s -%d %s",command,n,args);
 if(socketpair(AF_UNIX,SOCK_STREAM,0,sfd)<0)
  {
   return NULL;
  }
#ifdef DEBUG
 fl_addto_browser_chars(textbox, "Executing command: ");
 fl_addto_browser_chars(textbox, com);
 fl_addto_browser_chars(textbox, cr);
#endif
 pid=subprocess(com,sfd);
 fd=sfd[0];
 if(pid<=0)
  {
   fl_show_alert("Fatal!","Cannot create subprocess.",NULL,1);
   if(fd>=0) { close(sfd[0]); close(sfd[1]); }
   return NULL;
  }
 FD_ZERO(&fds);
 FD_SET(fd,&fds);
 for(;;)
  {
   for(timeout=0;timeout<5&&result==0;timeout++)
    { /* Wait for data from subprocess */
     result=select(FD_SETSIZE,&fds,NULL,&fds,&tv);
     if((pstatus=waitpid(pid,NULL,WNOHANG))!=0) break;
    }
   if(result>0) /* Data arrived */
    {
     bytes=read(fd,buf,BUFSIZE-1);
     if(bytes<=0) { close(sfd[0]); close(sfd[1]); return NULL; }
     buf[bytes]=0;
     fl_addto_browser_chars(textbox, buf); /* Already has newline */
    }
   if(pstatus!=0) break; /* Process terminated */
   if(timeout==5) /* Timeout occured, kill process. */
    {
     kill(pid,SIGHUP);
     fl_show_alert("Command timed out",
                   "DSP is not responding to your command",
                   "(Maybe it's dead?)",1);
     if(fd>=0) { close(sfd[0]); close(sfd[1]); }
     return NULL;
```

```
    }
   }
  if(fd>=0) { close(sfd[0]); close(sfd[1]); }
  return buf;
 }

char *mdspcmd2(int n,char *s1, char *s2)
{
 char cmd[CMDSIZE];
 strncpy(cmd,s1,CMDSIZE);
 strncat(cmd," ",CMDSIZE);
 strncat(cmd,s2,CMDSIZE);
 return mdsptool(n,cmd);
}

int hexparse(char *s,__u32 *val)
{
 unsigned long l;
 int result;
 result=sscanf(s,"%lx",&l);
 *val=(__u32)l;
 return result!=1;
}

int ishex(char *s)
{
 unsigned long dummy;
 return sscanf(s,"%lx",&dummy);
}

/* Global */
static FL_FORM *fsel=NULL;

/* A struct for write register command callbacks */
struct fsel_struct
{
 FL_OBJECT *accept,*cancel,*input,*button[32];
 char *action;
 char **flags;
 int devno;
 unsigned long val;
 char cval[128];
};

void flag_selection_update_buttons(struct fsel_struct *fs)
{
 int i;
 unsigned long val;
 char *c=(char *)fl_get_input(fs->input);
 if(sscanf(c,"0x%lx",&val)==1)
  {
   fs->val=val;
   for(i=0;i<32;i++)
     if(val&1L<<i) fl_set_button(fs->button[i],1);
  }
}

void flag_selection_update_input(struct fsel_struct *fs)
{
```

```
 int i;
 unsigned long val;
 for(i=0,val=0;i<32;i++)
  {
   if(fl_get_button(fs->button[i])!=0)
     val|=1<<i;
   else
     val&=~(1<<i);
  }
 sprintf(fs->cval,"0x%lx",val);
 fl_set_input(fs->input,fs->cval);
}

void flag_selection_callback(FL_OBJECT *o, void *v)
{
 struct fsel_struct *fs=(struct fsel_struct *)v;
 if(o==fs->accept)
  {
   mdspcmd2(fs->devno,fs->action,fs->cval);
   fl_hide_form(fsel); fl_free_form(fsel);
   fl_activate_form(xmd->xmdsptool);
  }
 else if(o==fs->cancel)
  {
   fl_hide_form(fsel); fl_free_form(fsel);
   fl_activate_form(xmd->xmdsptool);
  }
 else if(o==fs->input)
  {
   flag_selection_update_buttons(fs);
  }
 else /* Bit Buttons */
  {
   flag_selection_update_input(fs);
  }
}

#define CHOICE_WIDTH 350
#define CHOICE_HEIGHT 280

int flag_selection(char *title,struct fsel_struct *fs)
{
 int i;
 sprintf(fs->cval,"0x%lx",(unsigned long)fs->val);
 fsel = fl_bgn_form(FL_UP_BOX,CHOICE_WIDTH,CHOICE_HEIGHT);
 for(i=0;i<32;i++)
  {
   int x=(((CHOICE_WIDTH-10)/4)*(i%4))+5;
   int y=((CHOICE_HEIGHT/46)*(i-(i%4)))+5;
   static char btext[32][32];
   sprintf(btext[i],fs->flags[i]!=NULL?fs->flags[i]:"Bit %d",i);
   fs->button[i]=fl_add_checkbutton(FL_PUSH_BUTTON,x,y,80,20,btext[i]);
   fl_set_object_boxtype(fs->button[i],FL_SHADOW_BOX);
   if(fs->val&1L<<i) fl_set_button(fs->button[i], 1);
   fl_set_object_color(fs->button[i],FL_TOP_BCOL,FL_BLACK);
  }
 fs->input=fl_add_input(FL_NORMAL_INPUT,CHOICE_WIDTH/2-65,CHOICE_HEIGHT-82,
                        130,30,"Input");
 fl_set_input(fs->input,fs->cval);
```

```
 fs->accept=fl_add_button(FL_NORMAL_BUTTON,5,CHOICE_HEIGHT-45,
                          110,40,"Accept");
 fl_set_button_shortcut(fs->accept,"Aa",1);
 fs->cancel=fl_add_button(FL_NORMAL_BUTTON,CHOICE_WIDTH-115,CHOICE_HEIGHT-45,
                          110,40,"Cancel");
 fl_set_button_shortcut(fs->cancel,"^[c^c",1);
 fl_set_form_callback(fsel,flag_selection_callback,fs);
 fl_end_form();
 fl_show_form(fsel,FL_PLACE_MOUSE,0,title);
 fl_deactivate_form(xmd->xmdsptool); /* deactivate main form */
                                     /* while showing popup */
 return 0;
}

int set_register(int n,char *cmd)
{
 static char *hstr_flags[32]=
  { "TRDY","HTRQ","HRRQ","HF3","HF4","HF5", "HINT","HREQ", };
 static char *hctr_flags[32]=
  { NULL,"TREQ","RREQ","HF0","HF1","HF2","DMAE","SFT","HTF0",
    "HTF1",NULL,"HRF0","HRF1",NULL,"HS0","HS1","HS2",
    NULL,NULL,"TWSD",};
 static char *pci_flags[32]=
  { "IO","MEMORY","MASTER","SPECIAL","INVALIDT","PALETTE","PARITY","WAIT",
    "WAIT","SERR","FASTBCK",NULL,NULL,NULL,NULL,NULL,
    NULL,NULL,NULL,NULL,NULL,"S_66MHZ","S_UDF","S_FASTB",
    "S_PRTY","S_MEDI","S_SLOW","S_STABRT","S_RTABRT","S_RMABRT","S_SSERR",
    "S_DPARTY",};
 static char *hcvr_flags[32]=
  { "HC","HV0","HV1","HV2","HV3","HV4","HV5","HV6","HNMI", };
 char *cval,*title;
 __u32 val;
 static struct fsel_struct fs;
 fs.action=cmd;
 fs.devno=n;
 switch(cmd[1])
  {
   case 'S':
     cval=mdsptool(n,"-s");
     fs.flags=(char **)hstr_flags;
     title="Please enter value for HSTR";
     break;
   case 'C':
     cval=mdsptool(n,"-c");
     fs.flags=(char **)hctr_flags;
     title="Please enter value for HCTR";
     break;
   case 'I':
     cval=mdsptool(n,"-i");
     fs.flags=(char **)hcvr_flags;
     title="Please enter value for HCVR";
     break;
   case 'P':
     cval=mdsptool(n,"-p");
     fs.flags=(char **)pci_flags;
     title="Please enter value for PCI (bad values may hang PCI transfers)";
     break;
   default: return 0;
  }
```

```
 if(!hexparse(cval,&val))
  {
   fs.val=(unsigned long)val;
   sprintf(fs.cval,"0x%x",val);
   flag_selection(title,&fs);
  }
 else
  {
   fl_show_alert("Command failed", "mdsptool cannot determine status",
                 "Please try again",1);
   return 1;
  }
 return 0;
}

char *upload(int n)
{
 char *fname;
       /* message   dir  pattern  default */
 fname=(char *)fl_show_fselector("Please select file for Upload",NULL,
                                 "*.pci", NULL);
 if(!fname||*fname==0) return NULL;
 return mdspcmd2(n,"-f",fname);
}

char *mdsptool_p(int n,char *cmd)
{
 char *input;
 const char *enterhex="Please enter hex number (like 0x0f)";
 unsigned int dummy,ok=0;
 do
  {
   input=(char *)fl_show_input(enterhex,"0x");
   if(input==NULL) return NULL; /* User hit Cancel */
   if(sscanf(input,"%x",&dummy)!=1)
    {
     fl_show_alert("Not a Hex number:", input,
            "Please try again",1);
    }
   else ok=1;
  }
 while(!ok);
 return mdspcmd2(n,cmd,input);
}

int verify_devices(FD_xmdsptool *x)
{
 int i,found;
 for(i=1,found=0;i<=MAX_MDSP;i++)
  {
   int fd;
   char *name=(char *)fl_get_choice_item_text(x->setdevice,i);
   if((fd=open(name,O_RDONLY))<0)
    {
     fl_set_choice_item_mode(x->setdevice,i,FL_PUP_INACTIVE);
    }
   else
    {
     close(fd);
```

```
      found++;
     }
   }
  return(found);
 }

 /* Universal Callback */
 void callback(FL_OBJECT *o, long l)
 {
  static int devno=1; /* Default DSP Board No. */
  switch(l)
   {
    case 0: exit(0);
    case 1: /* Device */
            devno=fl_get_choice(o); break;
    case 2: /* Upload */
            upload(devno);
            break;
    case 3: /* Stat All */
            mdsptool(devno,"-a");
            break;
    case 4: /* Print Control */
            mdsptool(devno,"-v -c");
            break;
    case 5: /* Set Control */
            set_register(devno,"-C");
            break;
    case 6: /* Print Status */
            mdsptool(devno,"-v -s"); break;
    case 7: /* Set Status */
            set_register(devno,"-S");
            break;
    case 8: /* Print Vector */
            mdsptool(devno,"-v -i"); break;
    case 9: /* Set Vector */
            set_register(devno,"-I");
            break;
    case 10: /* Print PCI */
            mdsptool(devno,"-v -p");
            break;
    case 11: /* Set PCI */
            set_register(devno,"-P");
            break;
    case 12: /* Reset */
             fl_show_alert("Function not implemented.",
             "The DSP56301 does not support a soft reset.",
             "Please use the reset button at the back of the board.",1);
             break;
    case 13: /* Write */
            mdsptool_p(devno,"-T");
            break;
    case 14: /* Read */
            mdsptool(devno,"-v -t");
            break;
    case 15: /* Dump */
            mdsptool_p(devno,"-D");
            break;
    default: break;
   }
```

```
}
```

Quelltext 45: `functions.c`

Utility `xmdsptool` – main.c

```
#include <sys/signal.h>
#include "forms.h"
#include "xinterface.h"

/* Global, unfortunately the only way to pass these to callback routines */
FD_xmdsptool *xmd;
FL_OBJECT *textbox;

extern int verify_devices(FD_xmdsptool *); /* functions.c */

int main(int argc,char **argv)
{
 int d;
 char msg[128];
 fl_set_icm_color(0,0,0,0);
 fl_set_icm_color(1,0,0,0);
 fl_set_icm_color(2,0,0,0);
 fl_initialize(&argc,argv, "Xmdsptool", 0, 0);
 xmd=create_form_xmdsptool();
 textbox=xmd->text;
 d=verify_devices(xmd);
 sprintf(msg,"Found %d MDSP devices.\nPlease select a button.\n",d);
 fl_addto_browser(textbox, msg);
 fl_set_browser_fontsize(textbox, 12);
 fl_set_browser_fontstyle(textbox, 1);
 fl_show_form(xmd->xmdsptool,0,0,"Xmdsptool V1.0 (C) Klaus Knopper");
 signal(SIGCHLD,SIG_IGN); /* No zombie processes */
 fl_do_forms();
 return 0;
}
```

Quelltext 46: `main.c`

D GNU General Public License

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You

may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and an idea
of what it does.
Copyright © 19yy name of author

This program is free software; you can
redistribute it and/or modify it under the terms
of the GNU General Public License as published
by the Free Software Foundation; either version
2 of the License, or (at your option) any later
version.

This program is distributed in the hope that it
will be useful, but WITHOUT ANY WARRANTY; without
even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU
General Public License along with this program;
if not, write to the Free Software Foundation,
Inc., 675 Mass Ave, Cambridge, MA 02139,
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright © 19yy name of
author
Gnomovision comes with ABSOLUTELY NO WARRANTY;
for details type 'show w'.  This is free
```

```
software, and you are welcome to redistribute
it under certain conditions; type `show c' for
details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision' (which
makes passes at compilers) written by James
Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

E Abbildungs-, Tabellen- und Quelltextverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis