Realisierung eines eingebetteten NMP Echtzeitclients und Implementierung einer konfigurationslosen Netzanbindung

André Frambach

8. August 2007

Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund

Studienarbeit

Realisierung eines eingebetteten NMP Echtzeitclients und Implementierung einer konfigurationslosen Netzanbindung

von
André Frambach

Aufgabenstellung und Betreuung:
Prof. Dr. Lars Wolf und Dipl.-Inform. Zefir Kurtisi

Braunschweig, den 8. August 2007

Erklärung

Ich versichere, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Braunschweig, den 8. August 2007

Kurzfassung

Das NMP-Projekt beschäftigt sich mit dem echtzeitfähigen, verteilten Musizieren über Computernetzwerke. Bisherige Prototypen wurden rein in Software und auf Standard-PC-Hardware implementiert. Diese Arbeit beschäftigt sich nun mit der Implementierung der echtzeitkritischen Komponenten auf einer dedizierten, externen Hardwarekomponente, um so den störenden Einfluss von Arbeitsplatzbetriebsystemen zu eliminieren. Um trotz dieser Verteilung des Gesamtsystems die Einfachheit der Benutzung sicherzustellen, wird ein System zur konfigurationslosen Inbetriebnahme der externen Hardware untersucht und implementiert.

Abstract

The NMP-Project examines distributed, real-time music making in computer networks. Current prototypes are pure software based. This theses follows up the implementation of the real-time critical part of NMP using dedicated embedded hardware to eliminate the problems caused by desktop operating systems. To preserve the usibility of the system, a configuration-less operation is implemented.

TECHNISCHE UNIVERSITÄT BRAUNSCHWEIG
INSTITUT FÜR
BETRIEBSSYSTEME UND RECHNERVERBUND
Prof. Dr. L. Wolf

Braunschweig, den 28.09.06

Aufgabenstellung für die Studienarbeit

Realisierung eines eingebetteten NMP Echtzeitclients und Implementierung einer konfigurationslosen Netzanbindung

Betreuer: Zefir Kurtisi

vergeben an: André Frambach (a.frambach@tu-bs.de), Matrikel-Nr. 2741903

Überblick

Am IBR wird im Projekt NMP interaktives Netz gestütztes Musizieren erforscht. Ähnlich einer Onlinekonferenz soll diese Anwendung Nutzern ermöglichen, sich über den Computer zu einem NMP Server zu verbinden und gemeinsam mit anderen Teilnehmern in Echtzeit zu musizieren. Um die dafür erforderliche geringe Verzögerung zu gewährleisten, müssen Abläufe kontinuierlich und sehr zeitnah abgearbeitet werden, was nur von einem Echtzeit-Betriebssystem zuverlässig erreicht wird. Unsere Prototypen belegen, dass Desktop-Betriebssysteme hierfür unbrauchbar oder unzuverlässig sind, bzw. Anpassungen erfordern, die das Systemverhalten negativ beeinflussen.

Der Unvereinbarkeit zwischen der Anforderung, den NMP Client in die gewohnte Benutzerumgebung einzubinden, und der mangelnden Echtzeitfähigkeit von Desktop-Betriebssystemen soll in dieser Arbeit dadurch begegnet werden, dass der echtzeitkritische Teil des NMP Clients auf externe Hardware ausgelagert werden soll. Dieser eingebettete Echtzeitclient soll an das bestehende Netzwerk angeschlossen werden und ohne Benutzereingriffe der NMP Anwendung zur Verfügung stehen.

Aufgabenstellung

Zunächst sind die Anforderungen an die Hardware zu ermitteln und, unter Berücksichtigung einer einfachen Portierung bestehender Software, geeignete Entwickler Platinen für den Echtzeitclient zu bestimmen. Die in Zusammenarbeit mit der Betreuung gewählte Hardware ist dann in Betrieb zu nehmen, eine Entwicklungsumgebung aufzusetzen und die funktionsrelevanten Komponenten (Netzwerk, Audio, CPU) zu testen. Der Funktionstest soll mit einer Portierung des gegenwärtigen NMP Client Prototypen und der verwendeten Bibliotheken abgeschlossen werden.

Im zweiten Teil der Aufgabe soll dann eine Einbindung des Echtzeitclients in ein bestehendes Netzwerk ohne Benutzereingriff über *zeroconf* (RFC 3927) implementiert werden. PC-seitig ist ein Modul zu entwickeln, dass das Auffinden des Gerätes demonstriert und beispielhaft eine Verbindung dorthin aufbaut. Für die Umsetzung können quelloffene Bibliotheken (wie Apples *bonjour*) verwendet werden.

Laufzeit: 3 Monate

Die Hinweise zur Durchführung on Studien- und Diplomarbeiten am IBR sind zu beachten (siehe http://www.ibr.cs.tu-bs.de/lehre/arbeiten-howto/)!

Aufgabenstellung und Betreuung:

Prof. Dr. L. Wolf _____

Dipl.-Inform. Z. Kurtisi _____

Bearbeitung: André Frambach

_____

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Im Rahmen des Projektes Network-centric Music Performance (im Weiteren NMP) [1][2] wird am Institut für Betriebssysteme und Rechnerverbund (IBR) der TU Braunschweig das verteilte, Computernetzwerk gestützte Musizieren erforscht. Die erdachte Architektur besteht aus einem zentralen Server, welcher die Audioströme der partizipierenden Clients synchronisiert, mischt und das Ergebnis anschließend an die Clients verteilt.

Die Kernanforderung an ein solches System besteht in darin, eine maximale Latenz zwischen der Erzeugung eines Tons durch den Musiker und der Wahrnehmung desselben zu garantieren. Als von Musikern akzeptierte Obergrenze gelten hierbei 30 ms [3]. In dieser Zeitspanne muss das Audiosignal digitalisiert, zum Server übertragen, mit den Signalen der anderen Teilnehmer synchronisiert und gemischt, zum Client zurück übertragen und dort ausgegeben werden.

Jeden Teilaspekt dieser Verarbeitungskette gilt es auf seine Latenz hin zu optimieren, der Fokus dieser Arbeit liegt jedoch auf der Verarbeitung der Audiodaten im Client. Das erforderliche, präzise Timing der einzelnen Arbeitsschritte (Analog-Digitalwandlung, Komprimierung der Audiodaten, Versenden über ein IP-basiertes Netz, Empfangen des gemischten Audiosignals, Dekomprimierung, Digital-Analogwandlung) stellt hohe Anforderungen an das zugrunde liegende Hard- und Softwaresystem.

Die ersten NMP-Implementierungen basieren auf Standard-PC Hardware und Arbeitsplatzbetriebssystemen, auf denen eine reine Softwareimplementierung des NMP-Clients betrieben wird. Untersuchungen am IBR zeigen jedoch, das speziell Microsoft *Windows XP*, das am weitesten verbreitete Betriebsystem, im Bereich echtzeitkritischer Audioverarbeitung große Defizite aufweist. Diese Arbeit nun untersucht die Möglichkeit, den echtzeit-kritischen Teil des NMP-Clients (Audio-Ein- und Ausgabe, Netzwerkzugriff) auf eine dedizierte Hardware-Komponente auszulagern, während der Standard-PC als Eingabegerät für Konfigurationsdaten per Web-Interface erhalten bleibt.

Der erste Teil dieser Arbeit beschäftigt sich mit der Auswahl und anschließenden Evaluierung einer Hardware-Plattform und der zugehörigen Betriebssystemumgebung. Dazu werden die Anforderungen an eine geeignete Hardware-Plattform definiert und ein passendes System angeschafft. Der Inbetriebnahme des Boards folgt die Anpassung des eingesetzten Betriebsystems an den Aufgabenbereich eines NMP-Clients. Durch verschiedene Tests wird abschließend die Eignung der gewählten Plattform sichergestellt.

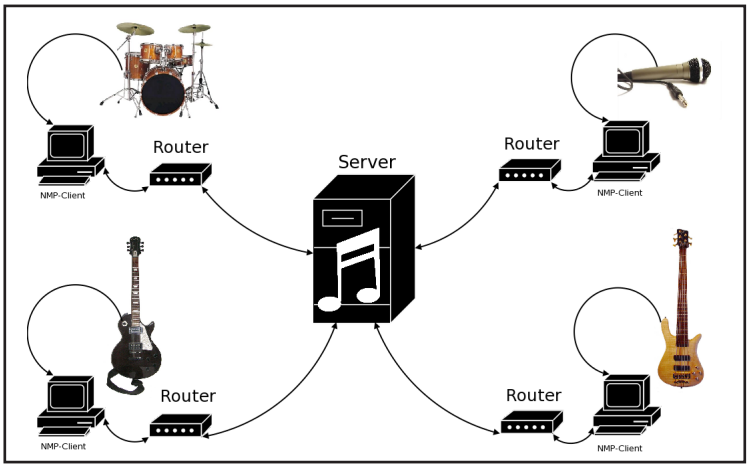

Abbildung 1.1: Die Architektur der ersten NMP-Prototypen

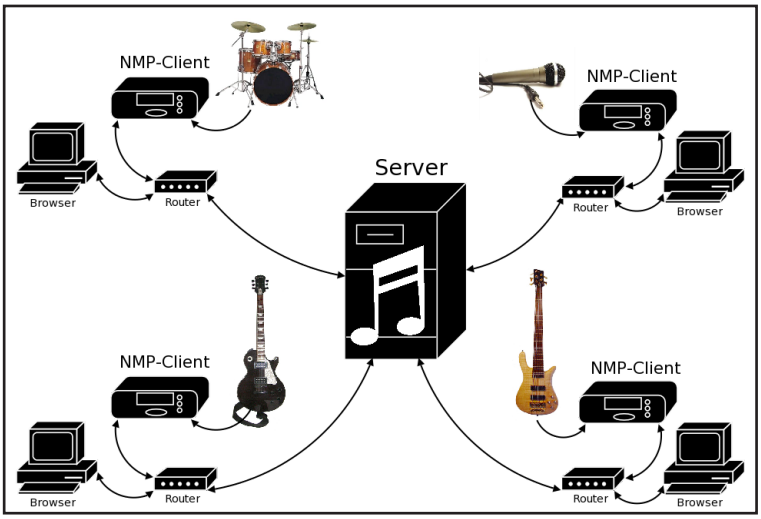

Abbildung 1.2: Die angestrebte Architektur mit dedizierter NMP-Hardware

Der zweite Teil dieser Arbeit betrachtet die konfigurationslose Einbindung der im ersten Teil vorgestellten Plattform in das Computer-Netzwerk des potentiellen NMP-Nutzers. Diese soll durch unter Anderem durch Nutzung des *Zeroconf*-Protokolls [4] sichergestellt werden. Dazu wird eine geeignete Zeroconf-Implementierung auf der Zielplattform in Betrieb genommen und auch die Unterstützung durch Arbeitsplatzbetriebssysteme untersucht.

2. Realisierung eines eingebetteten NMP Echtzeitclients

2.1 Die Zielplattform

2.1.1 Anforderungen

Zur Einschätzung der Anforderungen an eine Zielplattform bedarf es einer eingehenden Analyse der vorhandenen Implementierung der NMP-Clientkomponente. Dabei ist im Bereich der Software auf eine möglichste große Ähnlichkeit Wert zu legen, um die Portierung der bestehenden Implementierung weitestgehend zu vereinfachen. Die Betrachtung der Hardware hingegen muss differenzierter erfolgen, da die Zielplattform aus dem Bereich der eingebetteten Systeme stammen soll und dies Randbedingungen bezüglich Stromverbrauch, Ausmaße und Leistungsfähigkeit impliziert.

Der vorhandene Prototyp

PROGRAMMIERSPRACHEN, APIS UND BETRIEBSSYSTEME

Die in der Aufgabenstellung genannten Untersuchungen wurden mit Hilfe eines softwarebasierten Prototypen vorgenommen. Dieser wurde in der Programmiersprache *C++* realisiert, ebenso sind die genutzten Bibliotheken. Dabei wurde Wert auf eine möglichst gute Portierbarkeit des Quellcodes wie auch der Bibliotheken gelegt. Erfolgreich in Betrieb genommen wurde der Prototyp unter den Betriebssystem *Linux*, Microsoft *Windows XP* und Apple *MacOSX*.

Der Zugriff auf die Netzwerkroutinen des Betriebssystems wurden durch die plattform übergreifende *SOCKET*-Schnittstelle [5].

Zur Abstrahierung der verschiedenen APIs zum Zugriff auf Audiohardware wurde die Softwarebibliothek *RtAudio* [6] verwendet. Diese unterstützt *ALSA* (www.alsa-project.org), *OSS* und *JACK* (Linux), *DirectAudio*, *ASIO* (Windows) und *CoreAudio* (OSX). Sie ist speziell für den Gebrauch in echtzeitkritischen Anwendungen konzipiert. RtAudio nutzt unter Linux und OSX die Posix-Threadimplementierung *libpthread*, unter Windows die hauseigene.

Bei vorherigen Versuchen zeigten sich unter Linux die Kombination aus ALSA-API und Treibern am geeignetsten, unter Windows ASIO. Die ASIO-Spezifikation verlangt nach spezieller Treibersoftware, welche hauptsächlich für Erweiterungskarten aus dem (semi-)professionellen Bereich erhältlich sind.

Die Steuerung durch den Benutzer erfolgt über Kommandozeilenparameter, für die Zukunft ist ein Web-Interface vorgesehen.

Hardware

Die Betriebssysteme Linux und Windows wurden auf handelsüblichen Personal Computern betrieben. Die Audiounterstützung wurde durch vorhandene Onboard-hardware bzw. Erweiterungskarten mit PCI-Schnittstelle im Verband mit der zugehörigen Treibersoftware bereitgestellt. Zum Betrieb von OSX wurde ein Apple *iMac G5* inklusive dessen integrierter Audiohardware verwendet.

Das gesamte NMP-System ist für den Betrieb mit einer Abtastrate von 48 kHz bei 16 Bit/Sample Auflösung im Full-Duplexbetrieb.

Als Netzwerkschnittstelle kamen die bereits integrierten Fast-Ethernet-Adapter zum Einsatz, welche Übertragungsraten von 100 MBit/s im Full-Duplexbetrieb erreichen.

Anforderungsprofil einer NMP-geeigneten Hard- und Softwareplattform

Software

Im Hinblick auf eine möglichst problemelose Portierung der echtzeitkritischen Komponenten des Prototypen ist die Unterstützung der oben genannten Programmiersprachen unabdingbar. Optimal wäre eine ausgereifte Unterstützung der Zielplattform durch die *GNU Compiler Collection* (http://gcc.gnu.org), da diese bereits für die Plattformen Linux/IA32 und OSX/POWER Verwendung fand.

Aufgrund der Fülle der genutzten Betriebsystemfunktionen (Netzwerk, Threads, etc) ist Betriebssystemunterstützung erforderlich, ansonsten müssten diese Funktionen selbst implementiert oder auf bestehende, aber unerprobte Bibliotheken zurückgegriffen werden. Aufgrund der Fokussierung der vorhandenen Prototypen auf Posix-kompatible Systeme, bietet sich Linux an, das aufgrund seiner Offenheit im Umfeld der eingebetteten System gut unterstützt wird. Das potenzielle Linux-System sollte optimalerweise die eingesetzten Bibliotheken und APIs unterstützen. Besonders ist auf die Unterstützung der ALSA-Bibliothek zu achten, was auch einen passenden Treiber für die Audiohardware voraussetzt.

Hardware

Die gesuchte Hardwareplattform soll jegliche benötigte Computer-Hardware auf möglichst kleinem Raum vereinigen. Dazu gehört ein genügend leistungsfähiger

Prozessor, ausreichend großer Arbeitsspeicher (RAM), Festspeicher für das Betriebssystem, ein den allgemeinen NMP-Anforderungen (48 kBit/s Abtastrate, 16 Bit/Sample Auflösung, Full-Duplex) entsprechender Audio-Codec und eine Fast-Ethernet kompatible Netzwerkschnittstelle. Passend zu der Hardwareplattform sollte ein bereits vorkonfiguriertes, freies Linux-System vorhanden sein, welches die im vorigen Abschnitt genannten Anforderungen erfüllt.

2.1.2 Marktübersicht

Der Markt für eingebettete Systeme in der angepeilten Größenordnung wird momentan von mehreren Architekturen dominiert. Die *MIPS*-Architektur wird neben ihrer Verwendung in Highend-Unix-Workstations von *SGI* auch in eingebetteten Systemen eingesetzt. Dort dominiert der Automotivebereich, aber auch in Spielekonsolen (Nintendo 64) und Kommunikationsgeräten (AVM *FritzBox*, Cisco Router) finden MIPS-Prozessoren Verwendung.

Eher im Multimedia- und Mobilfunkbereich zu finden ist die *ARM*-Architektur (z. B. Apple *iPod*). Da die Firma ARM selbst keine Prozessoren produziert, sondern ausschließlich Lizenzen vergibt, existieren verschiedenste Ausprägungen von ARM-Prozessoren von allerlei Herstellern.

Die beiden größten Hauptprozessorhersteller *Intel* und *AMD* haben sich kürzlich von ihren Embedded-Architekturen getrennt, somit ist deren Zukunft ungewiss.

2.1.3 Auswahl

Aufgrund seiner besonderen Eignung für Multimediaaufgaben bei gleichzeitigem minimalen Stromverbrauch wurde die ARM-Architektur gewählt. Die vielen Lizenznehmer sorgen für eine enorme Auswahl, was die Chancen erhöht, ein den Anforderungen entsprechendes Derivat zu finden. So zeigten Recherchen in den Produktportfolios einiger Hersteller auch verschiendeste Ausprägungen, mit entsprechenden Zielanwendungen. Darunter korrespondieren die für den Einsatz in kompakten Multimediaabspielgeräten konzipierten Modelle am meisten mit den Anforderungen von NMP. Ein Anbieter solcher ARM-Prozessoren ist die amerikanische Firma *Cirrus Logic* (http://www.cirrus.com) mit ihrer *ARM9 EP93XX* Familie. Diese beinhaltet Prozessoren mit integrierter Netzwerk- und Audiounterstützung zu verhältnismäßig niedrigen Preisen. Gleichzeitig ist der Verlustleistung dieser hochintegrierten System so gering, dass sie ohne Kühler bzw. Lüfter auskommen und auch den potentiellen Betrieb ohne externe Stromversorgung zulassen.

Zur einfachen Evaluation dieser Plattform bietet der Hersteller verschiedene Development-Kits an, welche die Prozessoren auf einer Platine inklusive aller nöti-

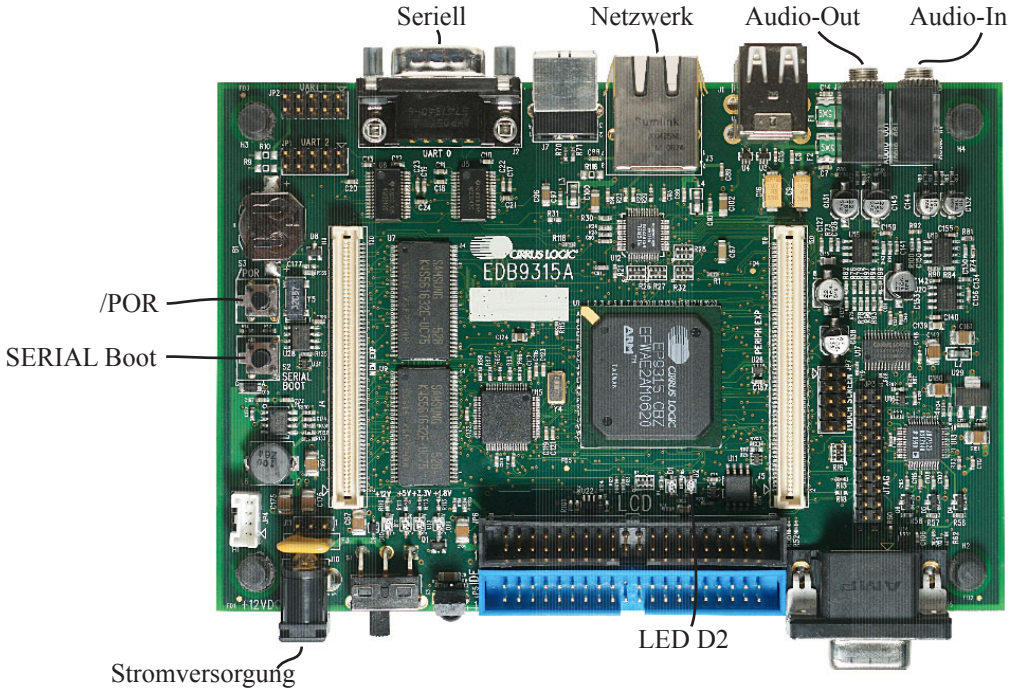

Abbildung 2.1: Das Entwicklungs-Board EDB9315A

gen Peripherie liefern.
Aufgrund der besten Verfügbarkeit zum Zeitpunkt dieser Arbeit wurde das ***EDB9315A*** [7] gewählt.

Die wichtigesten Leistungsmerkmale:

- Prozessor **EP9315-CB** [8] (200 MHz, Hardware-FP-Einheit, mathematischer Co-Prozessor *Math Crunch Engine*)
- 64 MByte SD-Ram
- 16 MByte Flash Speicher
- Stereo, 24 Bit Audio Ein-/Ausgabe
- 10/100 MBit/s Ethernetschnittstelle

2.2 Inbetriebnahme

2.2.1 Auslieferungszustand

Das gewählte Development-Kit von Cirrus Logic beinhaltet:

- EDB9315A – das Entwicklungsboard
- Netzteil mit tauschbarem Stromkabel (US-Ausführung)

- Null Modem Kabel
- IDE Kabel
- 3.5”-auf-2.5” Notebook IDE-Festplattenadapter
- Stromadapter: 2-Pin Anschluss auf Standard-IDE Stecker, jedoch ausschließlich die 5V Schiene
- 2 Adapter für UART1 und UART2 des Boards auf TIA-574 Stecker
- Software: diverse Evaluationsversionen kommerzieller Entwicklungstools

2.2.2 Arbeitsplatz und Verkabelung

Zur Durchführung der Arbeiten mit dem Entwicklungsset wird ein Standard-PC mit serieller RS232-Schnittstelle und Linux-Betriebssystem benötigt. Als Linux-Distribution wird seitens Cirrus Logic *Debian* empfohlen, da es dort zur ebenfalls zur Entwicklung genutzt wird. Im Rahmen dieser Studienarbeit wurde *Ubuntu 6.06 LTS* (http://www.ubuntu.com) verwendet, welches auf Debian basiert.

Zur Netzwerk-Kommunikation wurde ein Breitband-Router mit integriertem 4-Port Switch und DHCP-Server verwendet.

Zur Stromversorgung wird statt des gelieferten US-Kabels ein Kaltgerätekabel benötigt.

Der auf dem Board vorhandene *TIA-574* Stecker wird über das mitgelieferte Null-Modem Kabel an die entsprechende serielle *RS232* Schnittstelle des Entwicklungs-PCs angeschlossen.

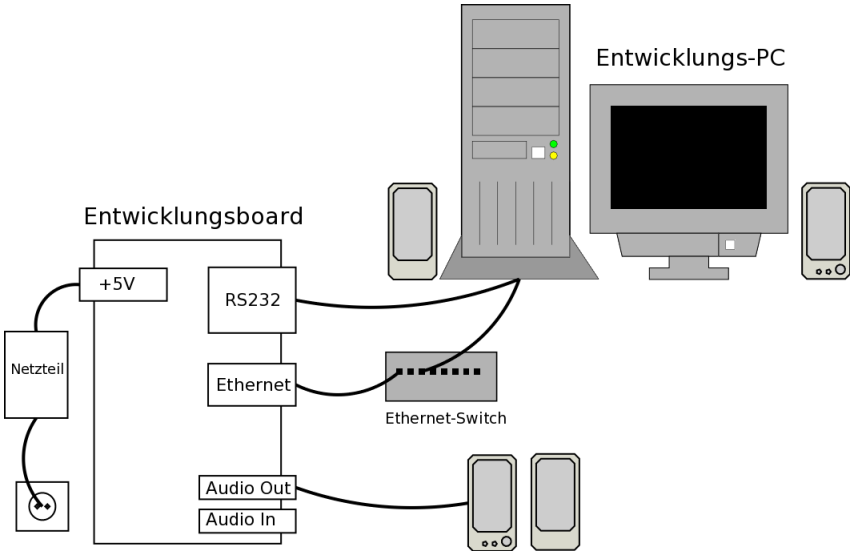

Abbildung 2.2: Der Arbeitsplatz

Die Ethernet-Schnittstelle des Boards wird mit Hilfe eines handelsüblichen CAT5 Ethernetkabels mit dem Router verbunden, ebenso der Entwicklungs-PC.

Die **AUDIO OUT**-Buchse des Boards wird durch eines 3.5 mm Klinkenkabels an einen Audioverstärker angeschlossen. An die **AUDIO IN**-Buchse kann je nach Bedarf ein beliebiges Audiosignal mit Line-Pegel angelegt werden.

Die weiteren vorhandenen Schnittstellen des Boards (VGA, IDE, LCD, etc) fanden im Rahmen dieser Arbeit keine Verwendung.

2.2.3 Erster Systemstart

Bei Auslieferung ist das Entwicklungsboard mit *Windows CE* vorkonfiguriert. Bei erfolgtem Anschluss eines Monitors und einer USB-Maus kann so ein erster Test der gebotenen Leistung der Plattform erfolgen.

2.3 Aufsetzen eines Linux-Systems

2.3.1 Vorbereitung des Entwicklungs-PCs

TFTP-Server

Zur Übertragung von Daten auf das ARM-Board wird ein TFTP-Server benötigt, welcher sich unter Ubuntu folgendermaßen einrichten lässt:

```
$ sudo apt-get install tftpd
```

Dieser bietet standardmäßig die Dateien innerhalb der Verzeichnisses */srv/tftp* im Netzwerk an. Zu übertragene Dateien müssen daher dorthin bzw. in einen Unterordner kopiert werden.

Terminal-Emulation

Zur Kommunikation mit dem ARM-Board über die serielle Schnittstelle ist ein Terminal-Emulationsprogramm vonnöten. Unter Ubuntu bietet sich *GtkTerm* an:

```
$ sudo apt-get install gtkterm
```

Cross-Compiler-Umgebung

Aufgrund dessen geringen Systemresourcen ist das Kompilieren von Software auf der Zielhardware nicht empfehlenswert. Daher wird auf dem Entwicklungs-PC eine sogenannte *Cross-Compiler-Toolchain* eingerichtet. Diese dient zum Über-

setzen von Software für ein Zielsystem, welches nicht dem System entspricht auf dem der Übersetzungsvorgang stattfindet (in diesem Fall wird auf einen x86-kompatiblen PC Software für die ARM-Architektur übersetzt). Eine bereits fertig konfigurierte Toolchain wird von Cirrus Logic im Internet bereitgestellt:

```
$ wget http://arm.cirrus.com/files/tools/arm-linux-gcc-
4.1.1-920t.tar.bz2
```

Dieses Archiv kann an einen beliebigen Ort im Dateisystem entpackt werden, von Cirrus Logic vorgesehen ist */usr/local/arm*

```
$ sudo mkdir /usr/local/arm
$ sudo tar xjf -C /usr/local/arm arm-linux-gcc-4.1.1-
920t.tar.bz2
```

In dem somit erzeugten Verzeichnis */usr/local/arm/4.1.1-920t* befinden alle zur Toolchain gehörigen Werkzeuge. Um diese bequem nutzen zu können und auch *configure*-Skripten zugänglich zu machen, sind mehrer Umgebungsvariablen zu setzten:

```
$ export PATH=$PATH:/usr/local/arm/4.1.1-920t/bin
$ export LDFLAGS=-L/usr/local/arm/4.1.1-920t/lib/
$ export LIBPATH=/usr/local/arm/4.1.1-920t/lib
$ export CC=arm-linux-gcc
$ export CXX=arm-linux-g++
$ export RANLIB=arm-linux-ranlib
$ export STRIP=arm-linux-strip
$ export LD=arm-linux-ld
$ export AR=arm-linux-ar
```

So wird sicher gestellt, dass während des Übersetzungsvorgangs jeweils die ARM-Version der benötigten Werkzeuge und Bibliotheken verwendet wird.

Das bekannte „Hallo Welt“-Programm

```
$ cat hello.c
#include <stdio.h>

int main()
{
     printf("Hello World!\n");
     return 0;
}
```

kann anschließend per

```
$ arm-linux-cc -o hello hello.c
```

übersetzt werden. Mit

```
$ file hello
hello: ELF 32-bit LSB executable, ARM, version 1
(ARM), dynamically linked (uses shared libs), not
stripped
```

kann überprüft werden, dass die Binärdatei *hello* für die ARM-Architektur übersetzt wurde.
Das selbe gilt für „Hallo Welt“ in C++:

```
$ cat hello.cpp
#include <iostream>

int main() {
     std::cout << "Hello World!" << std::endl;
     return 0;
}
$ arm-linux-g++ -o hello2 hello.cpp
$ file hello2
hello2: ELF 32-bit LSB executable, ARM, version 1
(ARM), dynamically linked (uses shared libs), not
stripped
```

Linux-System von Cirrus Logic

Fertiges System

Ein bereits an das verwendete ARM-Board angepasstes Linux-System stellt Cirrus Logic im Internet bereit:

```
$ wget http://arm.cirrus.com/files/linux/releases/
linux-2.6/1.0.0/linux_1-0-0-9315a.tar.bz2
```

Ausgepackt wird es durch

```
$ tar xfj linux_1-0-0-9315a.tar.bz2
```

Das so erzeugte Verzeichnis *9315A* enthält folgende Dateien:

download	Werkzeug zur Installation des Linux-Bootloader
redboot.bin	Fertig nutzbarer Bootloader RedBoot
zImage	Fertig nutzbarer Linux-Kernel
ramdisk.gz	Fertig nutzbare komprimierte Ramdisk

zImage und *ramdisk.gz* werden per TFTP zum ARM-Board übertragen und daher nach */srv/tftp* kopiert:

```
$ cp ramdisk.gz /srv/tftp
$ cp zImage /srv/tftp
```

Quellen und Buildsystem

Zusätzlich zum fertig nutzbaren Linux-System bietet Cirrus Logic auch dessen kompletten Quellen inklusive des verwendeten Build-Systems an. Diese sind notwendig um das Linux-System an die Erfordernisse eines NMP-Clients anzupassen.

```
$ wget http://arm.cirrus.com/files/linux/releases/
linux-2.6/1.0.0/linux_1-0-0-src.tar.bz2
$ tar xfj linux_1-0-0-src.tar.bz2
```

Im damit erzeugten Verzeichnis *linux-crater_1-0-0* befindet sich ein Build-System, mit dem recht einfach das Cirrus Logic Linux-System angepasst und der Linux-Kernel wie auch die zugehörige Ramdisk erzeugt werden kann. Dieses Build-System besitzt einige Abhängigkeiten und erfordert daher die Installation zusätzlicher Ubuntu-Pakete:

```
$ sudo apt-get install build-essential libncurses-dev
```

2.3.2 Installation des Bootloaders

Um das Board für die Verwendung mit Linux vorzubereiten, muss zunächst der Bootloader *RedBoot* fest in dessen Flashspeicher geschrieben werden. Dazu verbindet man Board und Desktop-PC per mitgelieferten seriellen Kabel und schaltet anschließend beide Geräte an. In der Konsole des Entwicklungs-PCs wechselt man in das im vorigen Abschnitt erzeugte Verzeichnis *9315A* und initiiert die Übertragung des Bootloaders:

```
$ ./download redboot.bin
Waiting for the board to wakeup...
```

Nun muss der Druckknopf **SERIAL BOOT** des Boards **gedrückt und gehalten** werden. Bei gedrückter Taste betätigt man den Knopf **/POR** und lässt ihn wieder los. **SERIAL BOOT** muss nun solange gehalten werden, bis die rote **LED D2** aufgehört hat zu leuchten. Nun sollte einige Meldungen wie

```
Downloading first boot code...( 26%)
```

und ähnliches erscheinen. Nach Abschluss des obigen dann

```
Programming the FLASH...( 45%)
```

und abschließend

```
Successfully programmed 'redboot.bin'.
```

Dabei stellt die Prozentangabe selbstverständlich den Fortschritt da und verändert sich daher naturgemäß.

Nach Abschluss obiger Vorgänge kann mit einem Terminal-Emulationsprogramm eine Verbindung zum Board hergestellt werden (57600 Bit/s, keine Pariät, 8 Bit, 1 Stopbit, keine Flusskontrolle). Nach betätigen der **/POR**-Taste des Boards sollten im Terminalfenster Meldungen von RedBoot erscheinen:

```
RedBoot(tm) bootstrap and debug environment [ROMRAM]
Non-certified release, version UNKNOWN - built
12:51:49, Apr 30 2007
Platform: Cirrus Logic EDB9315A Board (ARM920T) Rev A
Copyright (C) 2000, 2001, 2002, 2003, 2004 Red Hat,
Inc.
Copyright (C) 2003, 2004, 2005, 2006 eCosCentric
Limited
RAM: 0x00000000-0x04000000, [0x00042ee0-0x03fdd000]
available
FLASH: 0x60000000 - 0x61000000, 128 blocks of
0x00020000 bytes each.
RedBoot>
```

Damit befindet man sich auf der Eingabeaufforderung von RedBoot.

Die Konfiguration von RedBoot wird in eine eigens angelegte Partition des Flash-speichers geschrieben und ist daher nicht-flüchtig. Das Konfigurationswerkzeug startet man wie folgt:

```
RedBoot> fconfig -i
Initialize non-volatile configuration
- continue (y/n)? y
Run script at boot: false
Use BOOTP for network configuration: true
Default server IP address: 192.168.4.2
Set eth0 network hardware address [MAC]: false
GDB connection port: 9000
Force console for special debug messages: false
Network debug at boot time: false
Update RedBoot non-volatile configuration
- continue (y/n)? y
... Erase from 0x60fe0000-0x61000000: .
... Program from 0x03fe0000-0x04000000 at 0x60fe0000:
```

Für die ersten Gehversuche mit Linux auf dem Entwicklungsboard ist auf ein automatisch ausgeführtes Bootskript zu verzichten. Die automatische Konfiguration der Netzwerkschnittstelle des Boards (ausschließlich für RedBoot, das später startende Linux nutzt dazu eigene Mechanismen) setzt einen *BOOTP* bzw. *DHCP*-Server im Netzwerk voraus. Vom *Default server* lädt RedBoot den Linuxkernel wie auch die Ramdisk per *TFTP*, die IP-Adresse des Entwicklungs-PCs ist daher meist die richtige Angabe. Anschließend wird das Board per /**POR**-Taste zurückgesetzt und wenig später sollte folgendes im Terminalemulationsprogramm zu lesen sein:

```
Ethernet eth0: MAC address 00:60:2b:03:41:80
IP: 192.168.4.4/255.255.255.0, Gateway: 192.168.4.1
Default server: 192.168.4.2
RedBoot(tm) bootstrap and debug environment [ROMRAM]
Non-certified release, version UNKNOWN - built
12:51:49, Apr 30 2007
Platform: Cirrus Logic EDB9315A Board (ARM920T) Rev A
Copyright (C) 2000, 2001, 2002, 2003, 2004 Red Hat,
Inc.
Copyright (C) 2003, 2004, 2005, 2006 eCosCentric
Limited
RAM: 0x00000000-0x04000000, [0x00042ee0-0x03fdd000]
available
FLASH: 0x60000000 - 0x61000000, 128 blocks of
0x00020000 bytes each.
RedBoot>
```

2.3.3 Starten des fertigen Linux-Systems

Hat man wie im vorigen Abschnitt beschrieben RedBoot konfiguriert, kann man nun aus der Eingabeaufforderung von RedBoot heraus den zu startenden Linux-Kernel wie auch die Ramdisk per TFTP in den Arbeitsspeicher des ARM-Boads übertragen. Wurden die von Cirrus Logic bereitgestellten Abbilder dieser, wie in Abschnitt 2.3.1 beschrieben, in den Arbeitsordner des TFTP-Servers kopiert, starten man den Transfer wie folgt:

```
RedBoot> load -r -b 0x1000000 ramdisk.gz
Using default protocol (TFTP)
Raw file loaded 0x01000000-0x01b15a58, assumed entry at
0x01000000
```

Dabei sorgt *-r* für die Interpretation der Daten als *Raw* bzw. Binär, der Parameter nach *-b* bestimmt das Zieladresse der Übertragung im Arbeitsspeicher des Boards.

Analog wird mit dem Linux-Kernel verfahren:

```
RedBoot> load -r -b 0x80000 zImage
Using default protocol (TFTP)
Raw file loaded 0x00080000-0x001fa583, assumed entry at
0x00080000
```

Nun kann Linux gestartet werden:

```
RedBoot> exec -r 0x1000000 -s 0x1000000
-c "root=/dev/ram console=ttyAM" 0xc0080000
```

Dabei gibt *-r* die Adresse der Ramdisk an. Der Parameter nach *-s* bestimmt die Größe des Ramdisk-Images und sollte etwas größer gewählt werden, als der Platzbedarf dessen auf der Festplatte des PCs . Die an den Kernel übergebenen Kommandozeile definiert man mit *-c*, der abschließende Parameter ist die Einstiegsadresse des Kernels. Daraufhin erscheint folgendes:

```
Using base address 0x00080000 and length 0x0017a584
Uncompressing Linux.....done, booting the
kernel.
```

Im folgenden kann man den Startvorgang des Linux-Systems anhand der Ausgaben der Terminalemulation verfolgen. Bei Erscheinen von

```
Please press Enter to activate this console.
```

ist Linux vollständig gestartet und nach betätigen der *Enter*-Taste befindet man sich angemeldet als Benutzer *root* in der Eingabeaufforderung des Linux-Systems auf dem Entwicklungsboard.
Alternativ dazu kann das System auch mit Hilfe von USB-Maus, USB-Tastatur und VGA-Monitor per graphischer Oberfläche bedient werden. Diese ist für diese Arbeit irrelevant und wird daher hier nicht näher behandelt.

2.3.4 Anpassen eines optimierten Linux-Systems

Das von Cirrus Logic angebotene Linux-System ist als Ersatz für Windows CE konzipiert. Die gebotene graphische Benutzeroberfläche ist auf ein optionales, kleinformatiges LC-Display angepasst und unterstützt typische PDA-Anwendungen wie die Verwaltung von Emails, Dokumenten und Kontaktdaten. Die Systemleistung ist daher auf diesen breit gefächerten Anwendungsbereich optimiert, worunter die angestrebte Leistung im Bereich der echtzeitkritischen Netzwerk- und Audioanwendungen leidet. Daher muss der Linux-Kernel wie auch die zugehörige Ramdisk an die gegeben Anforderungen angepasst und auf eine möglichst störungsfreie Operation hin optimiert werden. Dazu werden in dem in Abschnitt 2.3.1 erzeugtem Verzeichnis *linux-crater_1-0-0* eine Kernelkonfiguration erstellt und das Build-System so angepasst, dass es nur das absolut nötige in die neu erstellte Ramdisk integriert.

Anpassen der Gesamtkonfiguration

Das *nCurses*-basierte (http://www.gnu.org/software/ncurses/) Konfigurationswerkzeug startet man wie folgt:

```
$ cd linux-crater_1-0-0
$ make menuconfig
```

Im folgenden Menusystem muss der Pfad zu der in Abschnitt 2.3.1 installierten Cross-Compiler-Toolchain gesetzt werden (*Build options - Toolchain and header file location*). Des weitern sollte das Erstellen von RedBoot und *download* deaktiviert werden, da diese aus dem Standard-Linux-System verwendet werden (*Bootloader Options*). Unter *Individual Package Selection for the target* können die Pakete ausgewählt werden, welche in die erzeugte Ramdisk integriert werden sollen. Dabei können Programme der graphischen Benutzerstelle (*Graphic libraries and applications*) wie auch zu nicht genutzter/vorhandener Peripherie gehörende Werkzeuge (in *Hardware handling libraries and applications*) deaktiviert werden. Nach verlassen des Menusystems wird die versteckte Datei *.config* erzeugt, welche die komplette Konfiguration wiederspiegelt. Ein Kopie der in die-

ser Arbeit genutzten *.config*-Datei befindet sich in Anhang A.2.1.

Kernel konfigurieren

Die Konfiguration des Kernels kann nun angepasst werden:

```
$ cd linux-crater_1-0-0
$ make ep=9315a linux-config
```

In *Boot options - Default kernel command string* sollte *root=/dev/ram console=ttyAM* gesetzt werden, damit der Kernel standardmäßig sein Root-Dateisystem in einer Ramdisk vermutet.

Die meisten weiteren Voreinstellungen können beibehalten werden, speziell im Unterpunkt *Device Drivers* sollte jedoch alles unnötige deaktiviert werden. Dazu gehören neben *USB suppor*t auch *ATA/ATAPI/MFM/RLL*, *Graphics support* und *SCSI device support*. Die in dieser Arbeit genutzte Konfiguration befindet sich in Anhang A.2.2. Sie entspricht der Datei *.config* im Unterverzeichnis *linux-crater 1-0-0/kernel/linux-2.6.17.14*.

Sobald man das Menu-System verlässt, beginnt automatisch die Erstellung des neu konfigurierten Kernels.

Um eine möglichst kleine Latenz des Audiosystems zu gewehrleisten, muss der entsprechende Treiber im Linux-Quellcode modifiziert werden [9]. Dazu ist die Datei *linux-crater_1-0-0/kernel/linux-2.6.17.14/sound/arm/ep93xx-i2s.c* zu bearbeiten: Das Feld

```
.period_bytes_min
```

der Struktur

```
static const snd_pcm_hardware_t ep93xx_i2s_pcm_
hardware
```

ist von **1 * 1024** auf **1 * 256** zu ändern (Patch in Anhang A.1.1). Diese Struktur übergibt der Treiber an die ALSA-Zwischenschicht und legt damit die mininale Gesammtpuffergröße einer ALSA-Anwendung fest. Bei den für NMP typischen Parametern (Stereo, 16 Bit/Sample) ergibt sich also

```
256 Byte / (2 Kanäle * 2 Byte/Sample)
          = 64 Samples/Kanal
```

Bei einer Abtastrate von 48 kHz entspricht dies ca. 1,3 ms.

Ein weiterer Patch [10] (Anhang A.1.2, anzuwenden in *linux-crater_1-0-0/kernel/linux-2.6.17.14/include/asm-arm/arch-ep93xx/*) korrigiert einen falsch gesetzten Parameter im Kernelcode, der die korrekte Auswertung der Echtzeituhr verhinderte.

Diese Änderungen ist unbedingt nach dem vorigen *make ep=9315a linux-config* auszuführen, damit diese Änderung nicht von anderen Patches, die das Build-System einspielt, überschrieben wird.

Durch

```
$ make
```

wird der Erstellungsprozess für das Gesamtsystem angestoßen. Dieser versucht erneut den Kernel zu erstellen, bemerkt jedoch die aktuelle Version, aktualisiert nur die von den vorigen Änderungen abhängigen Teile und fährt mit dem Rest des Systems fort. Dafür werden nacheinander die benötigten Quellen aus dem Internet nachgeladen, gegebenenfalls gepatched und anschließend kompiliert.

Nach Abschluss der obigen Vorgänge befinden sich der neu erstelle Kernel (*zImage*) und die dazugehörige Ramdisk mit den eben erstellten Softwarepaketen (*ramdisk.gz*) im Verzeichnis *linux-crater 1-0-0/images/9315A*. Zur weiteren Nutzung sollten sie nach */srv/tftp* kopiert werden, damit sie für das ARM-Board erreichbar sind.

Ramdisk anpassen

Die im vorigen Abschnitt neu erstellte Ramdisk ist, nachdem der Linux-Kernel sie als Root-Dateisystem eingebunden hat, vollständig belegt. Somit muss um Platz für zusätzliche Programme zu schaffen ihre Kapazität vergrößert werden. Dazu wird sie an einen geeigneten Platz kopiert und dekomprimiert:

```
$ cd /tmp
$ cp /srv/tftp/ramdisk.gz
$ gunzip ramdisk.gz
```

Nun wird ein Verzeichnis als Einhängepunkt für die alte (kleine) Ramdisk, wie auch für die neue (große) Ramdisk erzeugt:

```
$ mkdir mountAlt
$ mkdir mountNeu
```

Die kleine Ramdisk wird eingehängt:

```
$ sudo mount ramdisk mountAlt -o loop
```

Eine genügend große neue Ramdisk (hier 30 MByte) wird erzeugt und mit Nullen beschrieben:

```
$ dd if=/dev/zero of=ramdiskNeu bs=30M count=1
```

Anschließend wird ein *ext2*-Dateisystem darin erzeugt:

```
$ mkfs.ext2 ramdiskNeu
```

Nun kann auch diese Ramdisk eingebunden werden:

```
$ sudo mount ramdiskNeu mountNeu -o loop
```

Jetzt kann der gesamte Inhalt von *mountAlt* nach *mountNeu* kopiert werden:

```
$ sudo cp -R mountAlt/* mountNeu/
```

Zusätzlich können nun beliebige Datein nach */tmp/mountNeu* kopiert und somit dem ARM-Board dauerhaft zur Verfügung gestellt werden.
Nach aushängen und Komprimierung ist die neue, größere Ramdisk einsatzbereit:

```
$ sudo umount mountNeu
$ sudo umount mountAlt
$ gzip ramdiskNeu
$ cp ramdiskNeu.gz /srv/tftp/ramdisk.gz
```

Wie in Abschnitt 2.33 beschrieben kann nun das ARM-Board mit angepasstem Kernel und Ramdisk gestartet werden.

2.4 Test der Komponenten

Nachdem in den vorigen Abschnitten eine Hardwareplattform ausgesucht und in Betrieb genommen und das zugehörige Softwaresystem angepasst wurde, gilt es nun die Kombination aus beiden im Hinblick auf ihre NMP-Tauglichkeit zu testen.

Besonders relevant ist die Verarbeitung von Audiodaten mit niedriger Latenz.

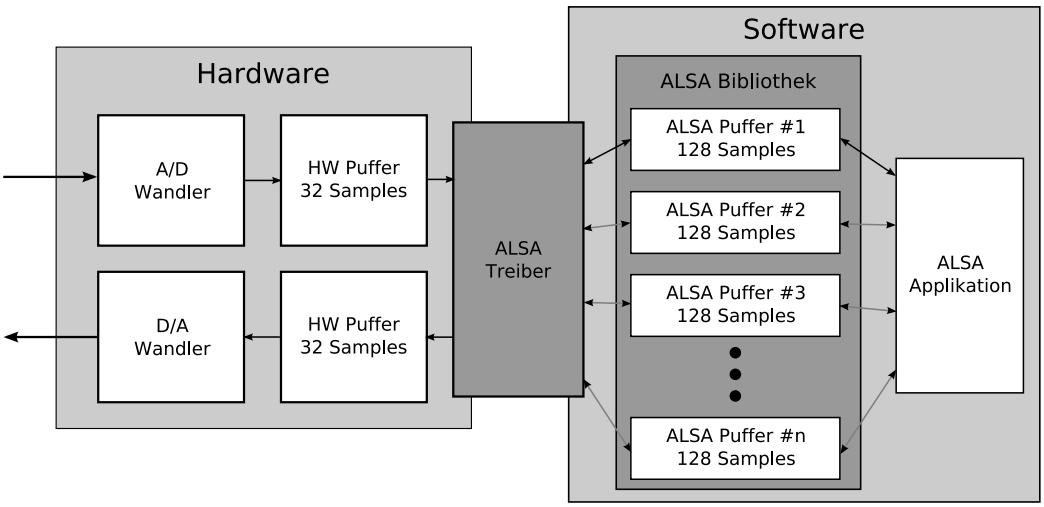

Abbildung 2.3: Audioverarbeitung durch ALSA

Dieser Zeitversatz zwischen Aufnahme und Wiedergabe des selben Samples wird durch mehrere Einflussgrößen bestimmt, einige fest durch die Hardware vorgegeben, einige als Parameter einstellbar. Abbildung 2.3 zeigt die grundlegende Architektur der Audioverarbeitung auf Basis von ALSA, wobei die Größe der Hardware- und ALSA-Puffer als Beispiele zu verstehen sind.

Wir nehmen an alle Puffer seinen leer. Nun verfolgen wir den Weg eines Samples *S0* durch die NMP-Hardware. Im ersten Schritt wird das Sample generiert, indem ein analoges Signal (z. B. von einem Mikrophone aufgenommen) digitalisiert wird. Nun wird es im Hardwarepuffer der Audiohardware abgelegt. Dort bleibt es, bis *S1 - S31* (bei einen HW-Puffeer von 32 Samples) ebenfalls digitalisiert und abgelegt wurden. Nun wird per Interrupt Request der ALSA-Treiber aufgeweckt, welcher diese 32 Samples in aktuell aktiven ALSA-Puffer ablegt (in Abb. XXX *ALSA Puffer #1*). Sobald dieser ALSA-Puffer (AP) voll ist (in Beispiel vier Transfers aus dem Hardwarepuffer) wird die ALSA-Applikation geweckt. Diese kann nun auf den Samples *S0-S127* arbeiten. Um eine minimale Latenz zu erreichen werden die verarbeiteten Samples direkt wieder in *AP#1* geschrieben. Während die Applikation mit *S0-S127* beschäftigt ist, füllt der Treiber im Hintergrund *AP2*. Angenommen die Anzahl der ALSA-Puffer ist auf zwei festgelegt, wird der nachdem *AP#2* komplett gefüllt ist, *AP#1* in 32 Samples-Blöcken an die Hardware übertragen und dort Sample für Sample digital-analog-gewandelt, d. h. wiedergegeben.
Angenommen alle anderen Vorgänge würden keine Zeit beanspruchen, ergäbe sich trotzdem prinzipbedingt eine Latenz zwischen Aufnahme und Wiedergabe von *S0* (2 Puffer, 128 Samples/Puffer): Nach *S0* müssen noch 127 weitere Samples für *AP#1* und 128 Samples für *AP#2* aufgenommen werden. Erst dann wird *S0* wiedergegeben. Somit ergibt sich bei 48 kHz Abtastfrequenz eine theoretische, minimale Latenz von 255 Samples, was

```
(255 / 48) ms ≈ 5,31 ms
```

entspricht. Dazu kommen in der Realität noch der Datentransfer zwischen den einzelnen Puffern, die Verzögerung der Reaktion auf den Interrupt Request und der Prozesswechsel, sobald die ALSA-Applikation aufgeweckt wird. Direkt durch die ALSA-Applikation beeinflussen lassen sich nur die Anzahl und Größe der ALSA-Puffer. So werden diese beiden Größen die Parameter im Großteil der folgenden Tests sein, durch welche die Eignung der Plattform als echtzeitaudioverarbeitendes System sichergestellt werden soll. Bei zu geringer Puffergröße ist zu befürchten, das der Zeitaufwand bis die Applikation aufgeweckt wurde den Großteil der vorhandenen Zeitspanne bis zu nächsten Aktivierung verbraucht und sie Applikation so nicht rechtzeitig neue Daten liefern kann. So könnten komplette ALSA-Puffer unverarbeitet bleiben bzw. mit zufälligen Daten gefüllt werden, was sich in Störgeräuschen in der Aufnahme bzw. Wiedergabe manifestiert.

Neben der Audioverarbeitung ist auch der Netzwerkzugriff ein entscheidender

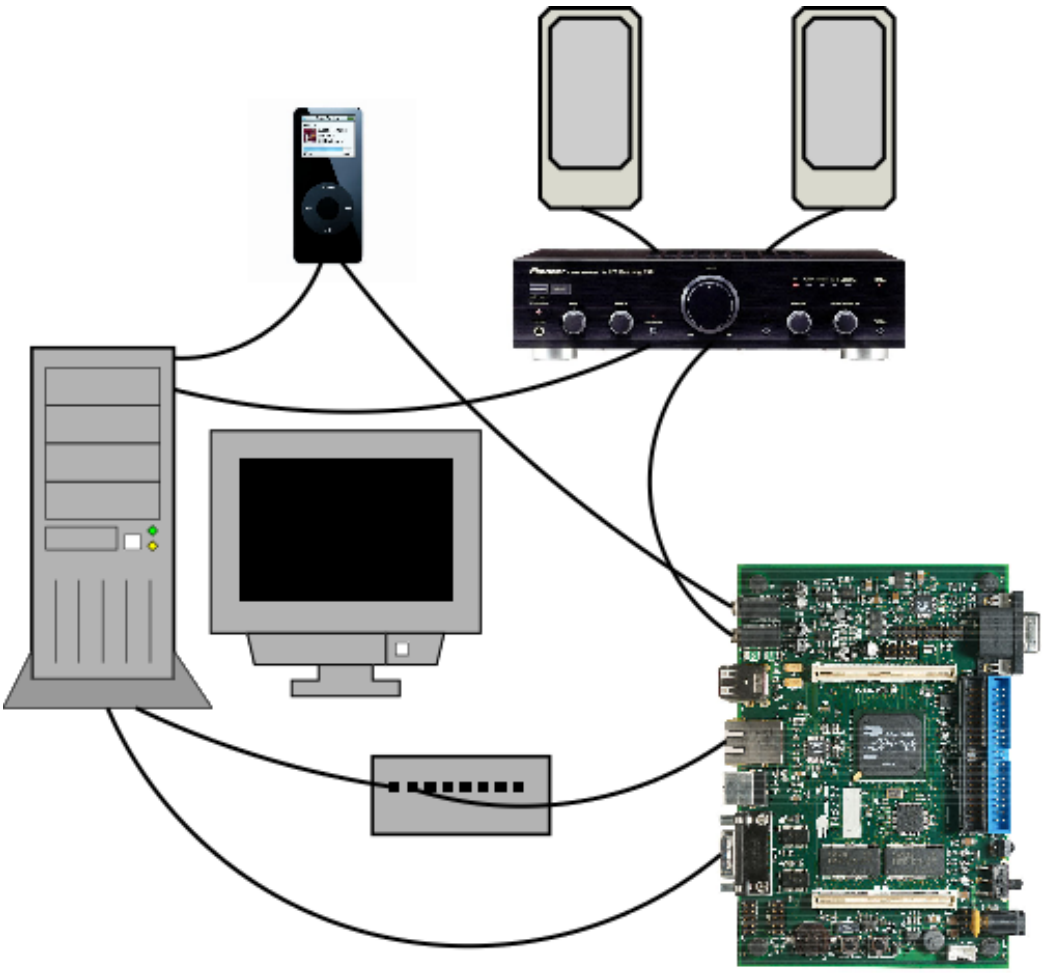

Abbildung 2.4: Allgemeiner Testaufbau

Faktor bei der Beurteilung des Gesamtsystems. Daher wird das Verhalten des Systems bei NMP-typischem Datenverkehr allein und auch in Verbindung mit Audioverarbeitung getestet.

Abschließend wird die Leistungsfähigkeit des Systems bei der Komprimierung von Audiodaten mit einem verlustlosen Verfahren überprüft.

Abbildung 2.4 zeigt den grundlegenden Testaufbau, Abweichungen bzw. Ergänzungen dazu finden sich bei den Beschreibungen der einzelnen Tests.

Entwicklungs-PC und Board sind per CAT5-Kabel mit einem Fast-Ethernet-Switch (integriert in einen Digitus *DN-11004-N* Breitband-Router inkl. DHCP-Server) verbunden. Zusätzlich sind beide per seriellem Kabel direkt verbunden, was einen einfachen Zugriff auf den Bootloader erlaubt. Sowie PC als auch Board sind mit einem Audioverstärker (Pioneer *A-109*) verbunden, um gegebenfalls ihre Audiowiedergabe akkustisch zu überwachen. Als Zuspieler dient ein handelsübliches, tragbares Audioabspielgerät (Apple *iPod Nano*), welcher je nach Bedarf mit dem PC oder dem Board verbunden ist.

Die eingesetzten Testprogramme werden auf dem Entwicklungs-PC cross-kompiliert und anschließend per TFTP zum Entwicklungs-Board übertragen.

Beispiel:

```
$ arm-linux-g++ -o testprog1 testprog1.cpp
$ cp testprog1 /srv/tftp
$ telnet <IP-Adresse des Boards>
# tftp -l testprog1 -g <IP-Adresse des PCs>
# chmod +x testprog1
#./testprog1
```

2.4.1 Audio

Die Leistung des Systems in der echtzeitkritischen Verarbeitung von Audiodaten ist der wichtigste Teilaspekt der Beurteilung der gesamten Plattform als NMP-Echtzeitclient. Zum Erreichen der angestrebten niedrigen Latenz ist ein optimales Zusammenspiel aus Betriebssystem, Treiberprogramm und Audio-API Voraussetzung. Um die Audioleistung möglichst unabhängig von anderen Einflußgrößen wie Netzwerk oder Festspeicherzugriff zu beurteilen, wurde ein Testprogramm entwickelt, welches ausschließlich den Full-Duplex-Audiobetrieb abdeckt. Um die Ergebnisse auf die NMP-Clientsoftware übertragen zu können, wurden die selben Bibliotheksfunktionen (RtAudio und ALSA) benutzt.

Den Ablauf des Testprogramms beschreibt folgender Pseudocode, der komplette

C++-Quellcode befindet sich in Anhang A.3.1.

```
Programm inout:

initialisiere_Audio_mit_Kommandozeilenargumenten
WIEDERHOLE:
     nehme_[Puffergröße]_Samples_auf
     gebe_[Puffergröße]_Samples_aus
```

Die Kommandozeilenargumente entsprechen denen in der Einleitung beschriebenen (Puffergröße und Anzahl der Puffer).

Durch sukzessive Absenkung der Pufferanzahl- und Größe (und damit der Latenz) soll herausgefunden werden, bei welcher minimalen Latenz störungsfreier Betrieb sichergestellt ist (Funktionstest). Gleichzeitig soll die reale Latenz gemessen werden, die sich aufgrund unbekannter Hardware-Parameter stark von der theoretischen unterscheiden kann (Latenztest).

Testaufbau und Durchführung

FUNKTIONSTEST

Der Audiozuspieler wird mit der **LINE-IN**-Buchse Entwicklungs-Boards verbunden und aufgrund des geringen Pegels seine Ausgangsleistung auf Maximum geregelt. Der Audiozuspieler wird in den Wiedergabemodus versetzt. Anschließend wird das Testprogramm vom Entwicklungs-PC aus per Telnet auf dem Entwicklungs-Board mit den entsprechenden Kommandozeilenargumenten gestartet. Für den Zeitraum von 60 Sekunden wird die Musikwiedergabe akustisch auf hörbare Störungen hin überwacht. In einer weiteren Telnet-Sitzung wird durch ein einfaches Skript (Anhang A.3.4) die Systemlast während des Tests aufgezeichnet und ihr Mittelwert berechnet. Dieser Vorgänge werden mit geänderten Parametern *Puffergröße* und *Pufferanzahl*

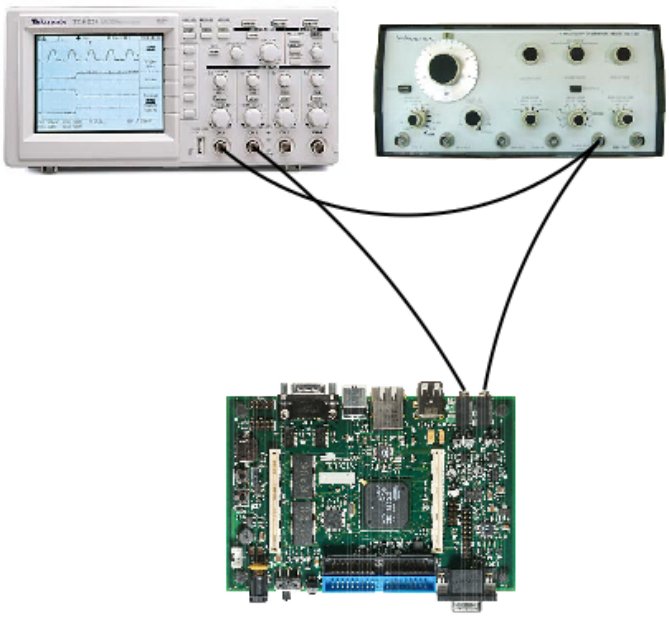

Abbildung 2.5: Aufbau Latenztest

wiederholt.

Latenztest

Die **AUDIO-IN**-Buchse des Entwicklungs-Boards ist mit einem 3,5 mm Stereo-Klinkenkabel bestückt. Die Kontakte für den linken Kanal sowie Masse des offenen Endes dieses Kabels sind mit dem **Pulse Out (TTL)**-Ausgang eines Funktionsgenerator verbunden (Wavetek 5MHz *XCG/SWEEP Generator Model 183*, Rechtecksignal, ca. 36 Hz bei eine Pulsbreite von ca. 6,4 ms, manueller Trigger). Zudem ist dieser Ausgang des Funktionsgenerator mit Kanal 1 eines digitalen Speicher-Oszilloskops verbunden (Tektronix *TDS 220*). Kanal 2 des Oszilloskops ist mit dem losen Ende eines weiteren 3,5 mm Klinkenkabels verbunden, dessen anderes Ende die **AUDIO-OUT**-Buchse des Boards belegt. Die Trigger-Quelle des Oszilloskops ist eine negative Flanke auf Kanal 2, der Trigger-Modus *Single Shot*.

Per Terminal-Emulation wird auf dem Boards das Testprogramm *inout* unter Angabe von Puffergröße und Pufferanzahl gestartet. Der Single Shot-Trigger des Oszilloskops wird zurückgesetzt bzw. *scharf* (armed) gemacht. Per Druckknopf wird am Funktionsgenerator ein einzelner Rechteckimpuls emittiert. Per Cursor wird am Oszilloskop der Zeitversatz zwischen dem Erscheinen des Rechteckimpulses auf Kanal 1 und Kanal 2 bestimmt. Dieser Vorgänge werden mit geänderten Parametern *Puffergröße* und *Pufferanzahl* wiederholt.

Beobachtung und Auswertung

Folgende Tabelle zeigt die jeweils eingestellten Testparameter und das beobachtete Ergebnis des Funktionstests wie auch die gemessene Latenz.

Puffergröße (Samples)	Anzahl Puffer	CPU-Last	Latenz	Hörtest
1024	8	5,57 %	192	i. O.
512	8	6,20 %	96	i. O.
256	8	8,83 %	48	i. O.
128	8	11,90 %	24,4	i. O.
64	8	17,33 %	12,8	Störungen
1024	4	5,21 %	108	i. O.
512	4	6,04 %	53	i. O.
256	4	7,73 %	27,2	i. O.
128	4	12,00 %	14	i. O.
64	4	19,00 %	7,4	Störungen
1024	2	5,34 %	64	i. O.
512	2	6,23 %	32	i. O.

Puffergröße (Samples)	Anzahl Puffer	CPU-Last	Latenz	Hörtest
256	2	7,81 %	16,4	i. O.
128	2	11,15 %	8,6	i. O.
64	2	17,66 %	4,6	Störungen

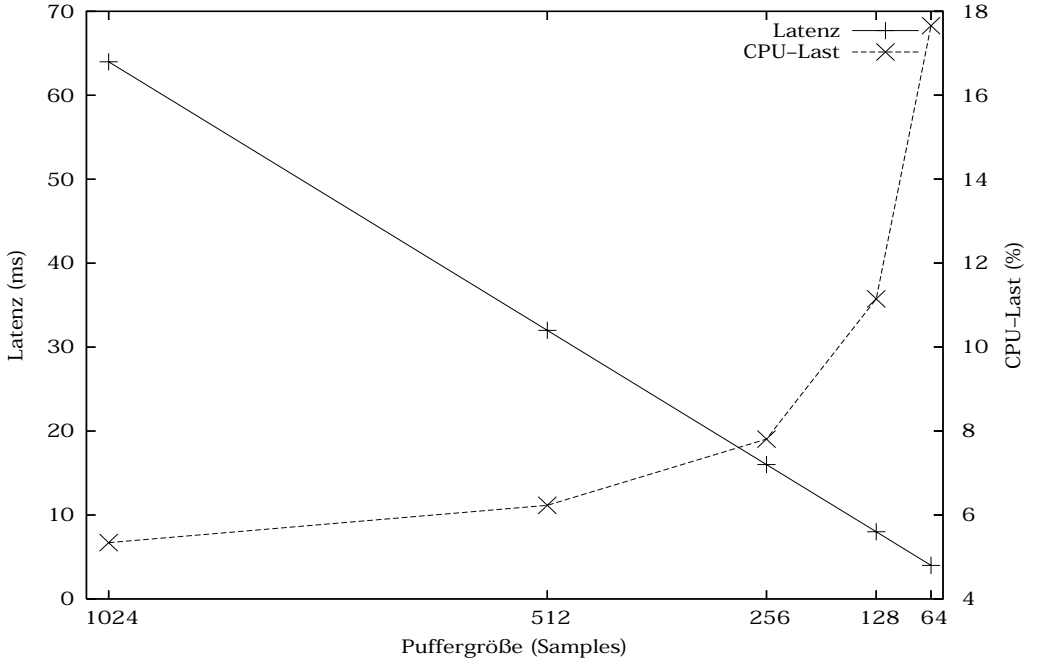

Abbildung 2.6: Zusammenhang zwischen Puffergröße, CPU-Last und Latenz

Abbildung 2.6 setzt den Testparameter Puffergröße mit der gemessenen Latenz und der Systemlast in Beziehung. Die Anzahl der Puffer ist dabei konstant 2. Wie erwartet sinkt mit verringerter Puffergröße auch die Latenz, während die Systemlast steigt.
Abbildung 2.7 zeigt die gemessene Latenz in Abhängigkeit des Testparameters *Anzahl der Puffer*. Der Puffergröße beträgt dabei konstant 128 Samples. Wie auch die Puffergröße weist die Anzahl der Puffer einen signifikanten Einfluss auf die Latenz auf.

Fazit

Der Funktions-und Latenztest bescheinigt der gewählten Plattform bei den für NMP angestrebten Paramtern 128 Samples/Puffer und zwei Puffer insgesamt eine Latenz von 8,6 ms bei einer CPU-Last von 11,15%. Gleichzeitig stellt diese Konfiguration auch die Untergrenze der sinnvoll einsetzbaren dar, denn der Betrieb mit 64 Samples/Puffer zeigte sich schon ohne jegliche weitere Systembelastung als nicht stabil. Diese geringe Latenz attestiert die gute Eignung der Plattform für Echtzeitaudio-Anwendungen und die gemessene Last lässt Spielraum für weitere NMP-relevante Verarbeitungsschritte wie Komprimierung und Netzwerkzugriff.

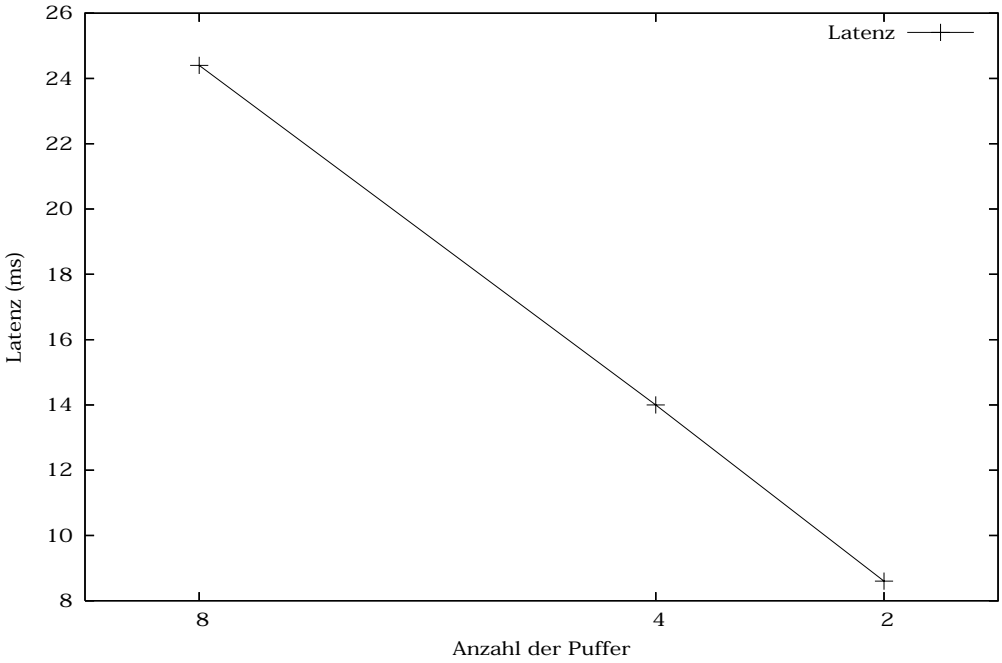

Abbildung 2.7: Zusammenhang zwischen Pufferanzahl und Latenz

2.4.2 Netzwerk

Für die Beurteilung der Netzwerkleistungsfähigkeit der gewählten Plattform ist reine Übertragungsleistung, durch die im Regelfall Computernetzwerksysteme charakterisiert werden, nicht so relevant. Viel mehr ist ein gutes Zusammenspiel aus User-Space-Bibliothek und Kernel-Mode Hardware-Treiber essentiell. Darunter zu Verstehen ist im Besonderen das zügige Versenden von Paketen und Weiterreichen von empfangenen Pakten an die Anwendung. Das vom NMP-System erzeugte Verkehrsmuster ist ungewöhnlich für ein IP-Netzwerk, welches ursprünglich für die zuverlässige Übertagung von nicht-echtzeitkritischen Daten konzipiert wurde. Durch in NMP festgelegten Parameter ergeben sich folgenden Werte:

```
48000 Samples/Sekunde / (128 Samples/Puffer)
          = 375 Puffer/Sekunde

1 Sekunde / 375 Puffer = 2,6 Millisekunden/Puffer

128 Samples/Puffer * 16 Bit/Sample * 2 (Kanäle)
          = 4096 Bit/Puffer = 512 Byte/Puffer
```

Somit wird ca. alle 2,6 ms ein Paket à 512 Byte (plus dem Protokolloverhead) gesendet und ebenso oft eines empfangen. Um die Netzwerkleistung der Plattform unabhängig von deren anderen Komponenten beurteilen zu können, wurde

ein Testprogramm entwickelt, welches ausschließlich die Netzwerkkomponenten betrifft. Es verwendet die *SOCKET*-Schnittstelle des Betriebsystems zur Abstrahierung des Netzwerkzugriffs, so wie auch der vorhandene Prototyp der NMP-Clientkomponente. Folgender Pseudocode beschreibt den Ablauf, der komplette Quellcode befindet sich in Anhang A.3.2:

```
Programm audio_reflector:

Initilisieren_socket_mit_kommandozeitlenargument
WIEDERHOLE
     warte_auf_paket
     kopiere_paket_von_Empfangspuffer_nach_Sendepuffer
     versende_paket
```

Das Kommandozeilenargument bestimmt die maximale Größe eines Paketes und sollte entsprechend der Stimulation durch die Gegentstelle gewählt werden.

Um den für NMP typischen Netzwerkverkehr zu simulieren, wurde ein weiteres Testprogramm entwickelt, welches auf dem Entwicklungs-PC ausgeführt wird (Quelltext in Anhang A.3.3):

```
Programm audio_reflector_client:

initialisiere_audio_und_socket
WIEDERHOLE
     nehme_[Puffergröße]_Samples_auf
     kopiere_Aufnahmepuffer_nach_Sendepuffer
     versende_paket
     empfange_paket
     kopiere_paket_von_Empfangspuffer_nach_
          Wiedergabepuffer
     gebe_[Puffergröße]_Samples_aus
```

Ein Kommandozeilenargument bestimmt den Testparameter Puffergröße, während die Anzahl der Puffer für diesen Test irrelevant ist und daher konstant bei zwei bleibt. Aus der Puffergröße und den anderen konstanten Parametern von NMP wird initial die tatsächliche Größe eines jeden versendeten Pakets berechnet. Dieser Wert ist dem oben beschrieben Programm zu übergeben. Weiterhin anzugeben ist die IP-Adresse des Entwicklungs-Boards.

Testaufbau und Durchführung

Die **LINE-IN**-Buchse des PC ist mit dem Audioabspielgerät verbunden. Im Mi-

xer-Programm des PCs ist die Aufnahme von der Quelle *LINE-IN* aktiviert.

Auf dem Entwicklungs-PC wird die Wiedergabe eines Musikstücks gestartet. Nun wird auf dem PC das Testprogramm *audio_reflector_client* mit den entsprechenden Kommandozeilenparamtern gestartet. Anschließend wird das Testprogramm *audio_reflector* vom Entwicklungs-PC aus per Telnet auf dem Entwicklungs-Board mit dem entsprechenden Kommandozeilenargument gestartet. Für den Zeitraum von 60 Sekunden wird die Musikwiedergabe akustisch auf hörbare Störungen hin überwacht. In einer weiteren Telnet-Sitzung wird durch das aus dem vorigen Test bekannte Skript (Anhang A.3.4) die Systemlast während des Tests aufgezeichnet und ihr Mittelwert berechnet.

Die gesamte Prozedur wird mit geänderten Kommandozeilenparamtern wiederholt.

Beobachtung und Auswertung

Folgende Tabelle zeigt die gegebenen Parameter und das beobachtete Ergebnis.

Puffergröße (Samples) / Paketgröße (Bytes)	CPU-Last	Hörtest
256 / 1024	3,33 %	i. O.
128 / 512	7,72 %	i. O.
64 / 256	15,07 %	i. O.

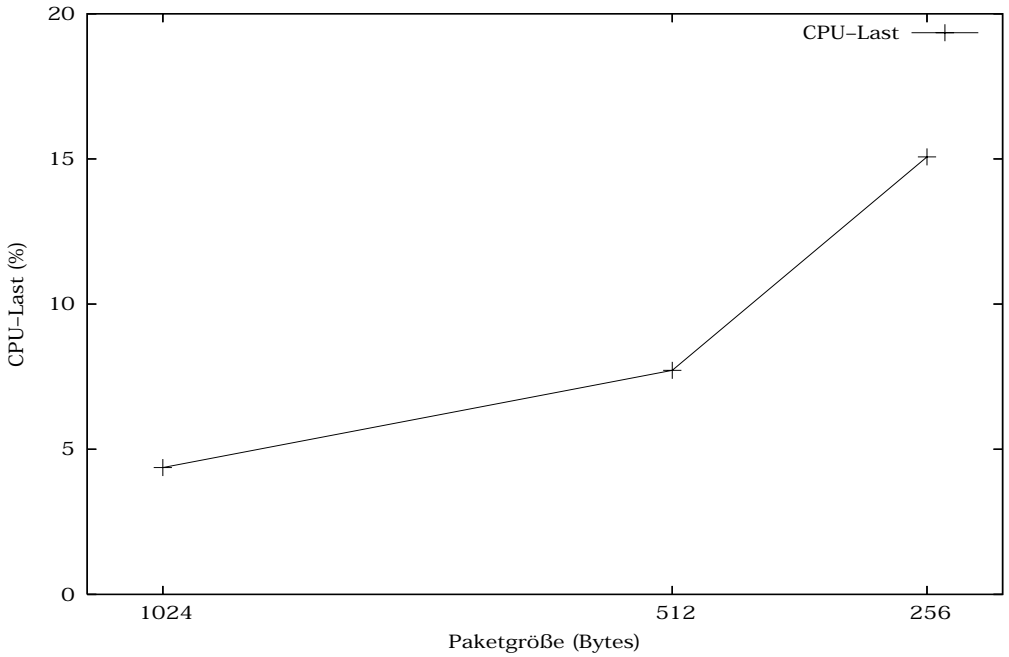

Abbildung 2.8: Zusammenhang zwischen Paketgröße und CPU-Last

Abbildung 2.8 setzt die Audio-Puffergröße bzw. den daraus resultierende Paketgröße mit der Systemlast in Beziehung. Wie erwartet steigt mit verringerter Puffergröße die Systemlast, was durch den erhöhten Overhead der Wechsel zwischen User- und Kernelmodus verursacht wird.

Fazit

Dieser Test zeigt die für NMP vollkommen ausreichende Netzwerkfähigkeit der Plattform auf. Der im realen NMP-Betrieb zusätzliche Verwaltungsverkehr ist aufgrund seiner niedrigen Echtzeitanforderungen und des geringen Datenvolumens zu vernachlässigen.

2.4.3 Audio und Netzwerk kombiniert

Nachdem in den beiden vorigen Tests das Audio-und das Netzwerksystem separat getestet und für ausreichend Leistungsfähig befunden wurden, gilt es nun das selbe für die in NMP so kritische Kombination aus beiden nachzuweisen. Dazu werden die im vorigen Test eingeführten Rollen aus *audio_reflector* und *audio_reflector_client* zwischen PC und Entwicklungs-Board getauscht. Die von der Zielplattform als *audio_reflecotor_client* durchgeführten Schritte aus Audioein- und Ausgabe und Netzwerkzugriff kommen dem Nutzungsmuster des realen NMP-Echtzeitclient schon sehr nahe. Somit sollte dieser Test als guter Indikator für die reale Leistungsfähigkeit der Plattform als NMP-Komponente sein.

Testaufbau und Durchführung

Der Audiozuspieler ist mit der **AUDIO-IN**-Buchse des Entwicklungs-Boards verbunden ist. Der Audiozuspieler wird in den Wiedergabemodus versetzt. Anschließend wird das Testprogramm *audio_reflector_client* vom Entwicklungs-PC aus per Telnet auf dem Entwicklungsboard mit den entsprechenden Kommandozeilenargumenten gestartet. Daraufhin wird das Testprogramm *audio_reflector* auf dem Entwicklungs-PC mit dem entsprechenden Kommandozeilenargument gestartet. Für den Zeitraum von 60 Sekunden wird die Musikwiedergabe akustisch auf hörbare Störungen hin überwacht. In einer weiteren Telnet-Sitzung wird durch ein einfaches Skript (Anhang A.3.4) die Systemlast während des Tests aufgezeichnet und ihr Mittelwert berechnet.

Die gesamte Prozedur wird mit geänderten Kommandozeilenparamtern wiederholt.

Beobachtung und Auswertung

Folgende Tabelle zeigt die gegebenen Parameter und das beobachtete Ergebnis.

Puffergröße (Samples) / Paketgröße (Bytes)	CPU-Last	Hörtest
256 / 1024	13,08 %	i. O.
128 / 512	21,49 %	i. O.
64 / 256	39,24 %	i. O.

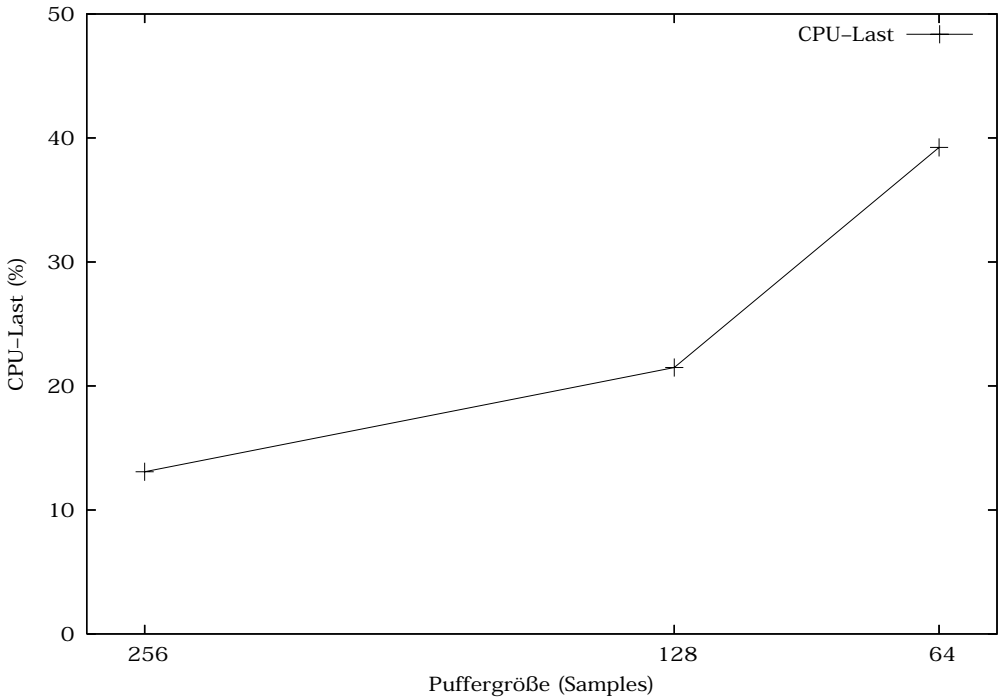

Abbildung 2.9: Zusammenhang zwischen Puffergröße und CPU-Last

Abbildung 2.9 setzt die Audio-Puffergröße bzw. die daraus resultierende Paketgröße mit der gemessenen der Systemlast in Beziehung. Wie erwartet steigt mit verringerter Puffergröße die CPU-Last, was durch den erhöhten Overhead durch die Wechsel zwischen User- und Kernelmodus wie auch die häufigeren Aktivierungen der Applikation verursacht wird.

Fazit

Auch die für NMP typische kombinierte Nutzung von Netzwerk- und Audiosystem ist ausreichend leistungsfähig und lässt noch Spielraum (d. h. CPU-Zeit) für weitere Teilkomponenten wie die Komprimierung.

2.4.4 Komprimierung

Im Gegensatz zu Süd-Korea, wo laut aktuellen Studien bereits über 90 % Prozent der Privathaushalte über eine Netzanbindung mit synchroner Transferleistung von 10 MBit/s und mehr verfügen, sind die Leistungsdaten der Internetanschlüsse von

Privatanwendern hierzulande nicht in der Lage, unkomprimierte Audiodaten in Echtzeit zu transferieren. Somit erscheint eine Reduzierung des unkomprimierten NMP-Datenstroms von *48000 * 2 Byte/Sekunde * 2 Kanäle = 1,5 MBit/s* in beide Richtungen sinnvoll, jedoch sollte diese nur sehr wenig zur ohnehin kritischen Latenz des Gesamtsystems beitragen. Durch diese hohen Latenzanforderungen stellen sich weit verbreitete Standardverfahren wie *MPEG1, Layer 3* (mp3) oder *MPEG4 AAC* als unzureichend heraus. Darüber hinaus gilt der Anspruch professioneller Musiker an die Qualität des Audiosignals als sehr hoch. Somit bietet sich ein verlustlos arbeitendes Verfahren an. Aufgrund dieser Voraussetzungen und seiner patent- und lizenzkostenfreien Verfügbarkeit wurde *FLAC* als möglicher Codec untersucht. Dessen Leitungsfähigkeit auf der Zielplattform soll hier grundlegend untersucht werden. Trotzallem FLAC bereits in anderen (kommerziellen) eingebettente System integriert ist, ist nicht davon auszugehen, dass die frei verfügbare Implementierung auf den Betrieb durch einen ARM-Prozessor optimiert ist.

Die genutzte FLAC-Quellendistribution ist im Internet verfügbar:

```
$ wget http://switch.dl.sourceforge.net/sourceforge/
flac/flac-1.1.4.tar.gz
$ tar xfz flac-1.1.4
$ cd flac-1.1.4
```

Im Sinne der Einfachheit wird das in der Distribution von FLAC enthaltene Kommandzeilenwerkzeug genutzt. Im ersten Arbeitsschritt muss dieses auf dem Entwicklungs-PC cross-kompiliert werden. Da die FLAC-Distribution auf dem Standard-Buildsystem um *configure* und *make* aufbaut, sind einige Umgebungsvariablen entsprechend anzupassen (vorausgesetzt die Cross-Kompilierungswerkzeuge wurden in */usr/local/arm* installiert):

```
$ export PATH=$PATH:/usr/local/arm/4.1.1-920t/bin
$ export LDFLAGS=-L/usr/local/arm/4.1.1-920t/lib/
$ export LIBPATH=/usr/local/arm/4.1.1-920t/lib
$ export CC=arm-linux-gcc
$ export CXX=arm-linux-g++
$ export RANLIB=arm-linux-ranlib
$ export STRIP=arm-linux-strip
$ export LD=arm-linux-ld
$ export AR=arm-linux-ar
```

Nun kann das Makefile generiert werden:

```
$ ./configure --host=arm-gnu-linux --disable-ogg
```

Anschließend wird der Übersetzungsprozess durch

```
$ make
```

gestartet. Nachdem dieser erfolgreich durchgelaufen ist, finden sich die relevanten Dateien im Unterverzeichnis *flac-1.4.4/src/*. Neben der ausführbaren Datei *flac* in Verzeichnis *flac/.lib* ist noch die dynamische Bibliothek *libFLAC.so.8.0.1* so wie die Links auf sie (*libFLAC.so.8, libFLAC.so*) in *libFLAC/.lib* wichtig. Sie alle müssen per TFTP zum Entwicklungs-Board übertragen werden.

```
$ cp flac/.lib/flac /srv/tftp
$ cp libFLAC/.lib/libFLAC.so* /srv/tftp
$ telnet <ip des boards>
# tftp -l flac -g <ip des pc>
# chmod +x flac
# tftp -l libFLAC.so.8.0.1 -g <ip des pc>
# tftp -l libFLAC.so.8 -g <ip des pc>
# tftp -l libFLAC.so -g <ip des pc>
# cp libFLAC.so* /lib
```

Testaufbau und Durchführung

Der Audiozuspieler ist mit der **AUDIO-IN**-Buchse des Entwicklungs-Boards verbunden ist. Der Audiozuspieler wird in den Wiedergabemodus versetzt. Innerhalb eine Telnet-Sitzung wird vom Entwicklungs-PC aus auf dem Board eine *WAV*-Datei von 60 Sekunden Länge (*Semisonic - Secret Smile,* Popmusik, Gitarre, Piano, Bass, Schlagzeug, Gesang) aufgenommen. Alle anderen Einstellungen entsprechen den NMP-Vorgaben (48 kHz Abtastrate, 16 Bit/Sample Auflösung):

```
# arecord -f dat -t wav -d 60 record.wav
```

Nun wird durch folgendes Skript

```
# cat /proc/uptime | awk '{print $1}'; ./flac -N
record.wav; cat /proc/uptime | awk '{print $1}'
```

record.wav komprimiert und die dazu verbrauchte Realzeit gemessen. *N* ist dabei als Parameter zwischen *0* und *8* zu setzen. Die verbrauchte Zeit und der Kompressionsgrad werden abgelesen. Nun wird die eben komprimierte FLAC-Datei wieder dekomprimiert und dazu nötige Zeit gemessen:

```
# cat /proc/uptime | awk '{print $1}'; ./flac -d
record.flac; cat /proc/uptime | awk '{print $1}'
```

Anschließend wird das Kommandozeilenargument *N* variiert, welches die Komprimierungsleistung und damit die Rechenzeit des Verfahrens beeinflußt, und die Komprimierung wiederholt.

Beobachtung und Auswertung

Folgende Tabelle zeigt den gewählten Komprimierungsparameter, den erzielten Kompressionsgrad und die für Komprimierung und Dekomprimierung verbrauchte Zeit.

Komprimierungs-parameter	Komprimierungs-grad	Dauer Komp./Dekomp./Summe (Sekunden)
0	0,597	11,9/8,75/**20,65**
1	0,590	12,33/9,07/**21,4**
2	0,589	14,72/8,94/**23,66**
3	0,537	44,04/10,42/**54,46**
4	0,515	61,16/11,25/**72,41**
5	0,513	94,62/11,21/**105,83**
6	0,513	95,10/11,18/**106,28**
7	0,512	144,87/11,16/**156,03**
8	0,499	205,22/11,67/**216,89**

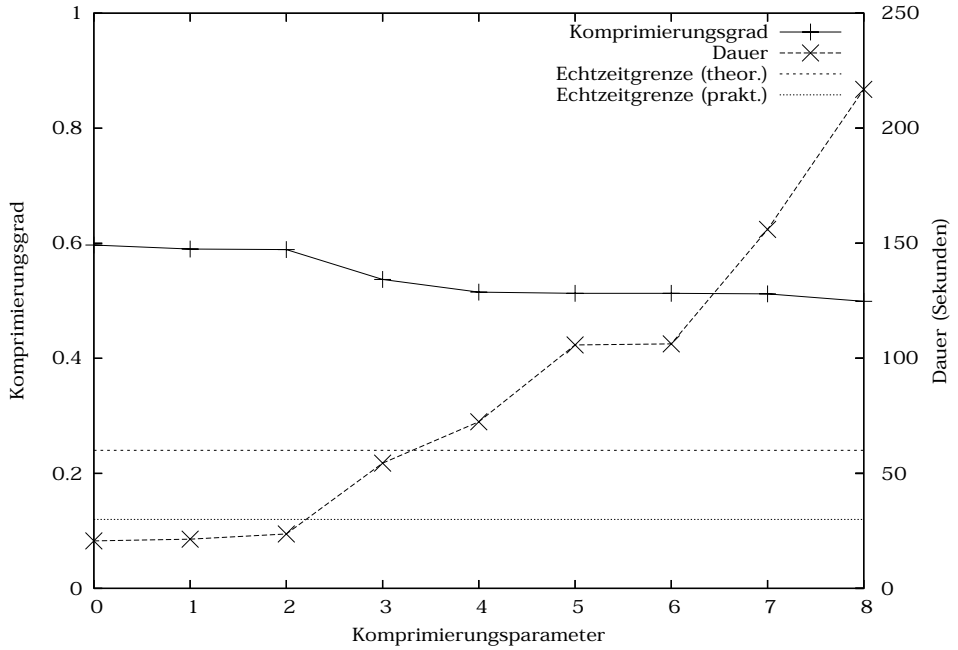

Abbildung 2.10: Zusammenhang zwischen Komprimirungsparameter, -dauer und -grad.

Da der Kodierungsaufwand auch vom Quellmaterial abhängt, ist diese Beobachtungen nicht allgemein gültig. Es zeigt sich jedoch, das der Gewinn an Komprimierung schlecht mit der beanspruchten Zeit skaliert (Abbildung 2.10).

Fazit

Im Hinblick auf den kritischen Faktor der Latenz und den anderen zu bewältigenden Aufgaben eines NMP-Clients ist eine (wenig) schlechtere, dafür wesentlich schnellere Komprimierung einer besseren, dafür viel langsameren, vorzuziehen. Daher sollte der Komprimierungsparamter maximal *2* sein. Weitere Verbesserung beim Umgang mit FLAC ist von der Nutzung der *Math Crunch Engine* des EP9315-EP Prozessors zu erwarten, deren Unterstützung in der genutzten GCC-Version jedoch nicht aktiviert ist.

2.4.5 Fazit

Da alle durchgeführten ein positives Ergebnis zeigen, scheint die Wahl der Hard- und Softwareplattform geglückt zu sein. Ob die auf den Praxisbetrieb als NMP-Client gemachten Rückschlüsse berechtigt sind, wird sich im Verlauf weiterer Tests des Gesamtsystems zeigen.

3. Implementierung einer konfigurationslosen Netzanbindung

Die im vorigen Teil dieser Arbeit eingeführte Architektur des NMP-Systems setzt auf eine zusätzliche, autonome Hardwareeinheit. Zu deren Konfiguration jedoch soll auch weiterhin ein Desktop-Computer eingesetzt werden. Um die Kommunikation zwischen NMP-Hardware, Desktop-Computer und NMP-Server ohne Fachkenntnisse im Bereich Computernetze sicherzustellen, soll die dazu nötige Konfiguration möglichst automatisch erfolgen. Neben der autonomen Einbindung ins bestehende Netz inklusive Internetzugang (3.1) soll sich der NMP-Echtzeitclient auch im lokalen Netzwerk selbst bekanntmachen, sodass der Nutzer vom Desktop-Rechner aus leicht auf ihn zugreifen kann (3.2).

3.1 Automatische Konfiguration der Netzwerkschnittstelle durch DHCP

Der in ersten Teil dieser Arbeit entwickelte autonome Echtzeitclient kommuniziert sowohl mit dem Computer des Anwenders als auch mit dem zentralen NMP-Server über IP-basierte Netzwerke. Dieses setzt eine entsprechende Konfiguration der Netzwerkschnittstelle der Clienthardware voraus. So muss ihr eine eindeutige IP-Adresse zugeordnet werden, welche den Client im lokalen, je nach Netztopologie auch im globalen, Netzwerk eindeutig identifiziert.

Weiterhin ist für den Fall eines kaskadierten Netzwerks zu konfigurieren, wie der Echtzeitclient das Internet erreicht, also in der Regel die IP-Adresse des entsprechenden Gateways.

Zur Nutzung von „natürlichen“ Rechnernamen anstatt von IP-Adressen ist ein *DNS*-Server (*Domain Name System*) [11] vonnöten, dessen IP-Adresse ebenfalls dem Client mitgeteilt werden muss.

Zur automatischen Konfiguration der oben genannten wie auch weiteren Netzwerkparametern wurde das *Dynamic Host Configuration Protocol* (DHCP) [12] definiert. Ein Rechner im Subnetz nimmt dabei die Rolle des DHCP-Servers ein. Dieser reagiert auf Anfragen von sich neu am Netzwerk anmeldenden Geräte und teilt diesen die nötigen Parameter mit.

Die Nutzung von DHCP setzt somit das Vorhandensein eine DHCP-Servers voraus. Dies ist jedoch aufgrund der großen Verbreitung von DHCP in fast allesn größeren Netzwerkinstallation wie in Firmen und Universitäten der Fall. Auch viele

Internet Service Provider lassen die Geräte ihrer Kunden über DHCP automatisch konfigurieren. Durch die große Verbreitung von breitbandigen Internetzugängen ist in immer mehr Haushalten ein IP-Router vorhanden. Diese Geräte haben in der Regel auch einen DHCP-Server integriert.

Somit ist aus infrastruktureller Sicht die automatische Konfiguration der Netzwerkschnittstelle sichergestellt.

Aus Sicht des Echtzeitclients ist nun offensichtlich ebenfalls eine Implementierung eines DHCP-Clients obligatorisch. Die in dieser Arbeit genutzte Linux-Distribution enthält dazu *udhcpc*. Dieses Programm wird standardmäßig beim Systemstart initiiert, um die Netzwerkschnittstelle zu konfigurieren.

3.2 Automatisches Auffinden der Benutzerschnittstelle durch Zeroconf

3.2.1 Anforderungen

Der NMP-Echtzeitclient soll in Zukunft eine HTTP-basierte Benutzerschnittstelle bieten daher einen rudimentären Webserver integrieren. Nach erfolgreicher Konfiguration via DHCP (s. o.) ist dieser im lokalen Netz unter Verwendung der IP-Adresse und des genutzten Ports erreichbar und kann mit einem Webbrowser genutzt werden. Alternativ zu HTTP ist auch ein eigenes Protokoll denkbar, was jedoch einen entsprechenden Client auf dem Rechner des Benutzers erfordert im Weiteren wird von der Nutzung eines Webservers und HTTP ausgegangen). In beiden Fällen ist jedoch die Kenntnis der möglicherweise variablen IP-Adresse nötig, um den Echtzeitclient zu erreichen. Bei Nutzung in einem fremden Netzwerk ist sie vollkommen unbekannt, aber auch im heimatlichen Netzwerk ist es nicht ungewöhnlich, dass ein DHCP-Server dem selben Gerät zu unterschiedlichen Zeitpunkten unterschiedliche IP-Adressen zuweist. Das Ermitteln der zugewiesenen IP-Adresse ist ohne weitreichende Kenntnisse des Netzwerks und entsprechende Befugnisse nicht ohne weiteres möglich und daher dem durchschnittlichen Computernutzer bzw. Musiker nicht zuzumuten.

Wünschenswert ist es daher, dass der Echtzeitclient seine Benutzerschnittstelle, d. h. seine IP-Adresse und den Port, auf dem der integrierte Server Verbindungen entgegen nimmt, im Netzwerk publik macht. Eine für diese Bekanntgabe sensitive Anwendung auf einem Desktopcomputer würde diese Daten dann zum Beispiel in Form eine *URL* (*Uniform Resource Locator*) dem Anwender präsentieren. Das Öffnen dieser URL (z. B. *http//192.168.1.7:8080*) mit einem Webbrowser führt so zur Benutzerschnittstelle des Echtzeitclients.
Das oben beschrieben Szenario setzt zur Identifikation des Echtzeitclients im

Netzwerk auf seine IP-Adresse. Intuitiver für den Anwender wäre jedoch ein „natürlicher“ Name anstelle dieser Zahlenkombination. So sollte es genügen, den wohlbekannten Namen des Echtzeitclients in einen Webbrowser (z. B. *nmpbox.local*) einzugeben, um auf sein Webinterface zu gelangen. Dazu muss der Webserver des Echtzeitclients auf Port 80 laufen, ansonsten ist eine explizite Angabe des Ports erforderlich. Die Domain *.local* sorgt dabei dafür, dass *nmpbox* nicht als Adresse eines Webservers im Internet fehlinterpretiert wird. Damit der Desktopcomputer *nmpbox.local* in eine IP-Adresse auflösen kann, ist ein Mechanismus vonnöten, der, unabhängig von zentralen DNS-Servern und beschränkt auf das lokale Netz, Namen IP-Adressen zuordnet.

3.2.2 Zeroconf

Ein Verfahren zum konfigurationslosen Betrieb von netzwerkfähigen Geräten ist **Zeroconf**. Es fasst drei Techniken zusammen, die mit den in den vorigen Abschnitten beschriebenen Anforderungen korrespondieren:

Automatische Allokation einer IP-Adresse **ohne** DHCP-Server

Die in Zeroconf beschriebene automatische Allokation von IP-Adressen nach RFC3927 (http://www.ietf.org/rfc/rfc3927.txt) beschränkt sich auf *Link-Local* Adressen. Deren Netzwerkverkehr ist laut Spezifikation nicht dazu geeignet, geroutet zu werden, d. h. IP-Pakete, deren IP-Header auf dem Weg zum Ziel verändert werden müssten, sind nicht zulässig. Da aber der Echtzeitclient auch mit einem NMP-Server über das Internet kommunizieren muss, ist dieser Mechanismus für NMP nicht geeignet. Stattdessen sollte wie in 3.1 beschrieben DHCP eingesetzt werden.

Auflösen von Namen in IP-Adressen **ohne** DNS-Server

Zum Auflösen von Namen in IP-Adressen definiert **mDNS** (*Multicast DNS,* http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt) ein Verfahren, dass ohne zentralen DNS-Server auskommt. Dazu werden Anfragen über die Domain *.local* nicht per Unicast an einen DNS-Server, sondern per Multicast-Adresse *224.0.0.251* an den UDP-Port *5353* aller im lokalen Netz vorhandenen Geräte gesendet. Antworten werden daraufhin per Multicast (und somit auch lesbar für alle anderen Geräte, die so ihre Daten aktuell halten können) oder Unicast (zur Vermeidung von unnötigem Netzwerkverkehr) übertragen.

Automatisches Verbreiten und Finden von Diensten im lokalen Netzwerk

DNS-SD (*DNS-Based Service Discovery,* http://files.dns-sd.org/draft-cheshire-

dnsext-dns-sd.txt) nutzt mDNS um im Netzwerk angebotene Dienste zu erfragen. Dazu sendet das an einem oder mehreren Diensten interessierte Gerät eine mDNS-Nachricht an alle vorhandenen Geräte. Sollte ein Gerät einen der gesuchten Dienste anbieten, so antwortet es ebenfalls per mDNS-Nachricht und teilt seine IP-Adresse, seinen Namen (*Hostname*), den Port des angebotenen Dienstes und eine kurze textuelle Beschreibung des Dienstes mit.

Eine Dienstart wird dabei wie folgt definiert:

```
_dienstkürzel._protokoll.local
```

Protokoll ist entweder *_tcp* oder *_udp*, Dienstkürzel z. B. *_http* für einen Webserver oder *_ipp* für einen netzwerkfähigen Drucker. Ein bestimmter Dienst hat entsprechend die Form:

```
Textuelle Beschreibung._dienstkürzel._protokoll.local
```

Also zum Beispiel:

```
NMP Konfiguration._http._tcp.local
```

Die textuelle Beschreibung soll dem Anwender dabei als einziges neben der Dienstart mitgeteilt werden.

Durch eine Implementierung der beiden Zeroconf-Techniken mDNS und DNS-SD auf dem Echtzeitclient wären die beiden verbleibenden Anforderungen aus 3.2.1 erfüllt. Passenderweise bietet Apple mit dem OpenSource-Projekt *mDNS-Responder* eine frei verfügbare, plattformunabhängige Implementierung just dieser Mechanismen an.

Auf Seite des Desktopcomputers, der die per Zeroconf feilgebotene Benutzerschnittstelle anzeigen soll, gibt es ebenfalls für jede relevante Plattform eine frei zugängliche Implementierung. In Apples *Mac OS X* ist *Bonjour* (so Apples Produktname für ihre Zeroconf-Implementierung) bereits enthalten, für Microsoft Windows XP ist ebenfalls eine Implementierung seitens Apple verfügbar. Zur einfachen Adressierung von Webservern ist dabei eine dynamische Liste von Lesezeichen in den Webbrowser integriert, welche per Zeroconf entdeckte Server anzeigt.

Eine komplett unter der *LGPL* stehende Implementierung für Linux-Desktops ist *Avahi*. Zum Beispiel das *service-discovery-applet* für den Gnome-Desktop überwacht die gewünschten Dienste und startet gegebenenfalls die verknüpften Anwendungen.

3.2.3 Inbetriebnahme von mDNSResponder

Der Quellcode für Apples Zeroconf-Implementierung steht als *mDNSResponder-107.6.tar.gz* (bzw. die entsprechend aktuelle Version) zum freien Download bereit. Nach dem entpacken des Archivs mittels

```
$ tar xfz mDNSResponder-107.6.tar.gz
```

befindet sich der relevante Teil im Unterverzeichnis *mDNSPosix* des neu erstellen Verzeichnisses *mDNSResponder-107.6*. Die sich dort befindende Datei *Makefile* steuert den gesamten Kompilierungsvorgang. Nachdem wie in Abschnitt 2.3.1 beschrieben die Umgebungsvariablen entsprechend gesetzt wurden, muss *Makefile* bearbeitet werden, da darin einige relevante Variablen fest (und daher hier falsch) kodiert sind (Anhang A.1.3). So muss Zeile ***CC = @cc*** (in der verwendeten Version Zeile 270) durch # auskommentiert oder gelöscht werden, da sie den durch Umgebungsvariablen vorgegebenen Compiler überschreibt. Das selbe gilt für ***STRIP = strip -S*** (Zeile 296**)**. Die Zeile ***LD = ld -shared*** muss in ***LD += -shared*** geändert werden, damit der richtige Linker genutzt wird (Zeile 271).

Ein

```
$ make os=linux SAResponder
```

startet den Buildvorgang. Nach dessen Vollendung befindet sich im Unterordner *build/prod* die ausführbare Datei *mDNSResponderPosix*, die einzige benötigte.

Ein

```
$ file mDNSResponderPosix
```

sollte durch

```
mDNSResponderPosix: ELF 32-bit LSB executable,
ARM, version 1 (ARM)
dynamically linked (uses shared libs), not stripped
```

bestätigen, dass es sich um eine ausführbare Datei für die ARM-Architektur handelt.

Nach Transfer per TFTP auf das ARM-Board muss die Datei noch ausführbar gemacht werden:

```
# chmod +x mDNSResponderPosix
```

Nun kann per

```
# ./mDNSResponderPosix -n „NMP Config“ -t _http._tcp -p
80
```

der auf dem ARM-Board laufende Webserver im Netzwerk publiziert werden. Dabei folgt dem Parameter *-n* eine die freie textuelle Beschreibung des Dienstes, welche dem Anwender an einem Desktopcomputer präsentiert wird. Die Dienstart bestimmt der Parameter *-t* mit der in 3.2.2 beschriebenen Syntax, *-p* bestimmt den Port auf dem der Service erreichbar ist.

4. Integration von NMP-Echzeitclient und Zeroconf-Implementierung

Nachdem nun sowohl der NMP-Echtzeitclient (zumindest theretisch) und die Zeroconf-Implementierung Bonjour (mDNSResponder) erfolgreich an die Zielplattform angepasst wurde, behandelt folgender Abschnitt die Integration dieser und anderer Applikationen in eine Ramdisk, die sich mit dem passendern Kernel fest im Entwicklungsboard bzw. einer späteren speziell angepassten Hardwarkomponente ablegen lässt. Mit entsprechend angepasstem Bootloader startet das Gerät nach dem Einschalten ohne weitere Eingriffe des Benutzers das Linux-Betriebssystem, bindet sich automatisch in das existierende Netzwerk ein und startet anschließend die gewünschte Applikation, z. B. den NMP-Echtzeitclient. Somit verhält sich das Gesamtsystem wie ein andere Unterhaltungselektronik.

Abhängigkeiten bestimmen

Im ersten Schritt ist zu bestimmen, welche Applikationen, Bibliotheken und sonstigen Datein benötigt werden. Gerade wenn Applikationen auf dynamisch nachladbare Bibliotheken zugreifen, ist nicht ohne weiteres zu erkennen, welche *.so*-Dateien benötigt werden. Ein einfacher, jedoch nicht sonderlich effizienter Weg zur inkrementellen Auflösung der Abhängigkeiten, ist folgender:
Per TFTP kopieren man die betreffende ausführbare Datei auf das Entwicklungsboard, macht sie per *chmod* ausführbar und startet sie. Nun wird bei fehlenden Bibliotheken der dynamische Linker die erste fehlende Bibliothek benennen. Nachdem diese per TFTP transferiert und nach */lib* verschoben wurde, startet man die Applikation erneut und der Linker weist auf die nächste fehlende Bibliothek hin. Diesen Vorgang aus starten, Linkerfehler auslesen und Bibliothek kopieren wiederholt man, bis die Applikation erfolgreich startet.

Ramdisk vorbereiten

Sind alles nötigen Dateien bekannt, gilt es die Ramdisk zur Bearbeitung vorzubereiten. Als Basis dient die in 2.3.4 angepasste Ramdisk, welche genügend freien Speicherplatz bietet. Diese wird an einen beliebigen Ort entpackt.

```
$ cp ramdisk.gz /tmp
$ cd /tmp
$ gunzip ramdisk.gz
```

Nun wird ein Verzeichnis erzeugt und die Ramdisk dort ins Dateisystem eingehangen

```
$ mkdir rdmount
```

```
$ sudo mount ramdisk rdmount -o loop
$ cd rdmount
```

Dateien kopieren

Nun können beliebige Dateien auf die Ramdisk kopiert werden. Zum Betrieb der NMP-Clients ist beispielsweise die FLAC-Laufzeitbibliothek notwendig. Daher wird sie nach *lib* kopiert.

```
$ sudo cp /srv/tftp/libFLAC.so* lib
```

Wie Kapitel 3 zeigt, wird zur Unterstützung des Zeroconf-Protokolls nur die monolithische Anwendung *mDSNReponderPosix* benötigt. Somit kopieren wir sie schlicht nach *usr/bin*.

```
$ sudo cp /srv/tftp/mDNSResponderPosix usr/bin
```

Autostart einrichten

Das verwendete Linuxsystem ist speziell an eingebettete System mit eingeschränktem Speicherplatz angepasst, daher ist keine Ausgewachsenes Startskriptsystem vorhanden. Bei jedem Start des Systems zu startende Programme können einfach ans Ende von *etc/rc.sysinit* eingetragen werden. Soll also sinnvollerweise der Zeroconf-Dienst gestartet werden, ergänzt man

```
mDNSResponderPosix -t http. tcp -n 'NMP Configuration'
-p 80 -b
```

So wird automatisch ein Webserver auf Port 80 im Netzwerk bekannt gemacht. Zu Demonstrationszwecken wird in der aktuellen Konfiguration der in *BusyBox* (http://www.busybox.net/) integrierte Webserver gestartet.

Ramdisk abschließen

Nachdem alle nötigen Veränderungen an der Ramdisk vorgenommen wurden, wird sie wieder ausgehängt, komprimiert und dem TFTP-Server zugeführt.

```
$ cd /tmp
$ sudo umount rdmount
$ gzip ramdisk
$ cp ramdisk.gz /srv/tftp
```

Ramdisk und Kernel im Festspeicher ablegen

In der Eingabeaufforderung von RedBoot kann nun die neue Ramdisk und der zugehörige Kernel per TFTP übertragen und anschließend in den nicht-flüchtigen

Flash-Speicher des Boards geschrieben werden.

Flashspeicher initialisieren:

```
RedBoot> fis init -f
```

Ramdisk übertragen:

```
RedBoot> load -r -b 0x1000000 ramdisk.gz
```

Ramdisk in den Flash-Speicher schreiben:

```
RedBoot> fis create ramdisk
```

ramdisk ist der Name der neu erzeugten Flash-Partition. Den übrigen Parameter werden vom vorigen *load*-Befehl übernommen.

Kernel übertragen:

```
RedBoot> load -r -b 0x80000 zImage
```

Kernel in den Flash-Speicher schreiben:

```
RedBoot> fis create zImage
```

Nun muss noch das Startskript angepasst werden.

```
RedBoot> fconfig
Run script at boot: true
Boot script:
Enter script, terminate with empty line
>> fis load ramdisk
>> fis load zImage
>> exec -r 0x1000000 -s 0x1000000 0xc0080000
>>
Boot script timeout (1000ms resolution): 1
Use BOOTP for network configuration: true
Default server IP address: 192.168.4.2
Set eth0 network hardware address [MAC]: false
GDB connection port: 9000
Force console for special debug messages: false
Network debug at boot time: false
Update RedBoot non-volatile configuration -
```

```
continue (y/n)? y
... Erase from 0x60fe0000-0x61000000: .
... Program from 0x03fe0000-0x04000000 at 0x60fe0000:
.
RedBoot>
```

Nach Betätigen der **/POR**-Taste am Board startet es neu. Statt wie bisher in RedBoot auf Eingaben zu warten, lädt das eben angelegte Bootskript automatisch die Ramdisk und den Kernel in den Arbeitspeicher des Boards und startet den Linux-Kernel. Wie vorher konfiguriert gibt es ein Startverzögerung von einer Sekunde. Während dieser kann per seriellem Kabel und Terminalemulation der Startvorgang unterbrochen werden (**Strg+C**), um eventuelle Fehler mit Hilfe der RedBoot-Eingabeauforderung zu berichtigen.

Nach kurzer Zeit erscheint z. B. im Safari-Browser bei den Bonjour-Bookmarks der Eintrag *NMP Configuration*.

5. Zusammenfassung und Ausblick

Die in Kapitel 2 ausgewählte und getestete Plattform erwies sich als überaus geeignet, um den NMP-Client vom PC in eine kompakteres Gerät zu überführen. Die gewählte Plattform um einen ARM-basierten Prozessor ist weit verbreitet und daher auch kostengünstig in der Beschaffung. Weiterhin garantiert der große Marktanteil ein gute Verfügbarkeit auch in der Zukunft. Die eingesetze Softwareplattform aus Bootloader und Betriebsystem besteht ausschließlich aus freier Open-Source Software. Diese Offenheit ermöglicht eine einfache Weiterentwicklung, Anpassung und Optimierung, ohne Einschränkungen durch NDAs oder andere lizenzrechtliche Querelen.

Die erfolgreiche Umsetzung des echtzeitkritischen Teils des NMP-Clients in ein eingebettes System eröffnet neue Möglichkeiten der Anwendung von NMP. So ist an eine Integration der Benutzeroberfläche in das selbe Gerät zu denken, das verwendete Enticklungs-Board z. B. bringt bereits alle nötigen Hard- und Softwareschnittstellen für eine Touchscreen-Bedienung mit. So könnte dann der komplette NMP-Client in ein Gerät in PDA-Größe integriert werden. Völlig unabhängig von einem PC und somit deutlich flexibler liebe sich NMP auch an unzugänglicheren Orten einsetzen, vorausgesetzt es gibt eine leistungsfähige Netzanbindung. Sollte sich in Zukunft eine schnurlose Übertragunstechnik etablieren, deren Eigenschaften bezüglich Datenrate, Übertragungslatenz und Zuverlässigkeit NMP gerecht wird, ließe sich ein komplett mobiles NMP-Endgerät entwickeln, bzw. die nötige Soft-und Hardware in ein Mobiltelefon integrieren.
Ein anderer Ansatz Musikern die Vorzüge von NMP nahezubringen wäre die Integration ein eingebetteten NMP-Client direkt in Instrumente oder sogar Mikrophone. So könnte wie in der in dieser Arbeit vorgestellten Architektur per PC das Session-Management stattfinden, während das Instrument die Übertragung der Echtzeit-Audiodaten selbstständig übernimmt. Bereits jetzt sind einige Musikgeräte mit integrierter Netzwerkunterstützung am Markt erhältlich, z. B. eine E-Gitarre mit Ethernetanschluß (Gibson *LesPaul HD6x Pro*, hier findet die Übertragung jedoch nur bis zu einer Breakoutbox statt)

In Kapitel 3 wurde die automatische Anbindung des eingebetteten NMP-Clients in ein bestehendes IP-Netzwerk untersucht. Es zeigte sich, das dies auf Basis des Zeroconf-Protokolls bzw. Apples Implementierung recht einfach umsetzen läßt. Diese geringe Komplexität erweckt die Hoffnung, das der Anteil der neu entwickelten Geräte, die Zeroconf unterstützen, weiter wächst. Hätte die Kobination aus Zeroconf, Ethernet-Verkabelung und Power-over-Ethernet (PoE) früher die nötige Marktdurchdringung erreicht, hätte es möglicherweise keinen Bedarf für eine weitere Hochgeschwindigskeitsschnittstelle wie *USB 2.0* gegeben.

Aber auch so kann Zeroconf die Bedienbarkeit vieler Geräte stark vereinfachen, denn gerade im immer wichtiger werdenden Segment der mobilen, schnurlos angebundenen Endgeräte spielt es seine Stärken aus. Da diese Geräte häufig über Wireless-Lan angebunden werden und sich ständig geänderten Netzwerkgegebenheiten ausgesetzt sehen, könnte Zeroconf hier den Nutzer wahrem Plug 'n Pay näher bringen. Beispielsweise könnten mobile Mitarbeiter mit häufig wechselnden Arbeitsplätzen Drucker und andere Peripherie ohne weitere Konfiguration einsetzen. Diese Trend zu einer Steckdosen-Komplexität (*Kabel rein, geht.*) ist auch in anderen Bereichen der Technik zu beobachten, z. B. bei der Kopplung von Bluetooth-Geräten über *NFC*, da man durch stark vereinfachte Bedienung bemüht ist, immer größere Kundenkreise anzusprechen.

Literaturverzeichnis

[1] U. Noyer , L.Wolf, X. Gu, M. Dick. Nmp - a new networked music performance system. In *Proceedings of the 1st ieee international workshop on networking issues in multimedia entertainment (nime'04)*, Dallas, USA, November 2004. IEEE.

[2] X. Gu, M. Dick, Z. Kurtisi, U. Noyer, L.Wolf. Network-centric Music Performance: Practice and Experiments. In *IEEE Communications,* vol. 43, no. 6, pp. 86-93, Juni 2005

[3] Z. Kurtisi, X. Gu, L. Wolf. Enabling Network-centric Music Performance in Wide Area Networks. In *Communications of the ACM, Special Issue on Entertainment Networking: Recreational Use of IP Networks*, vol. 49, no. 11, pp. 52-54, November 2006.

[4] A. Williams. Zero Configuration Networking - Internet-Draft. URL *http://cabernet.levkowetz.com/pdf/draft-ietf-zeroconf-reqts-11.pdf*

[5] J. M. Winett. The Definition of a Socket, RFC147. URL *http://www.isi.edu/in-notes/rfc147.txt.* Mai 1971.

[6] G. P. Scavone. RtAudio: A Cross-Platform C++ Class for Realtime Audio Input/Output. URL *http://ccrma.stanford.edu/~gary/papers/icmc2002c.pdf*

[7] Cirrus Logic. EDB9315A Engineering Development Board - Technical Reference manual. URL *http://arm.cirrus.com/files/schematics/edb9315a/RevB/DOC/EDB9315A_DS638DB1.pdf*. Dezember 2005.

[8] Cirrus Logic. EP9315 User`s Guide. URL *http://www.cirrus.com/en/pubs/manual/EP9315_Users_Guide.pdf.* Februar 2004.

[9] Takashi Iwai. Writing an ALSA Driver. URL *http://alsa-project.org/~iwai/writing-an-alsa-driver.pdf*

[10] Nico Augustijn. System timer runs at 1/2 speed for 2.6.17/20 (+patch). URL *http://arm.cirrus.com/forum/viewtopic.php?t=2858*

[11] P. Mockapetris. Domaind Names. Implementation and Specification, RFC 1035. URL *http://tools.ietf.org/html/rfc1035.* Standard. November 1987.

[12] R.Droms. Dynamic Host Configuration Protocol. RFC2131. URL *ftp://ftp.isi.edu/in-notes/rfc2131.txt*. Draft Standard. März 1997.

Anhang

A.1 Patches

A.1.1 ep93xx_alsadriver_latency.patch

```
--- ep93xx-i2s.c.old     2007-07-28 16:09:20.000000000
+0200
+++ ep93xx-i2s.c    2007-05-22 23:01:57.000000000
+0200
@@ -86,7 +86,7 @@ static const snd_pcm_hardware_t
ep93xx_i
     .channels_max  = 6,
 #endif

-    .period_bytes_min   = 1 * 1024,
+    .period_bytes_min   = 1 * 256,
     .period_bytes_max   = 32 * 1024,
     .periods_min   = 1,
     .periods_max   = 32,
```

A.1.2 ep93xx_rtc.patch

```
--- timex.h.old     2007-07-28 16:23:43.000000000
+0200
+++ timex.h    2007-07-28 16:23:15.000000000 +0200
@@ -2,4 +2,4 @@
  * linux/include/asm-arm/arch-ep93xx/timex.h
  */

-#define CLOCK_TICK_RATE       983040
+#define CLOCK_TICK_RATE       508000
```

A.1.3 mDNSResponder_Makefile.patch

```
--- Makefile.old     2006-06-21 01:07:04.000000000
+0200
+++ Makefile    2007-08-03 15:23:40.000000000 +0200
@@ -267,8 +267,8 @@
 SHAREDDIR = ../mDNSShared
 JDK = /usr/jdk
```

```
-CC = @cc
-LD = ld -shared
+#CC = @cc
+LD += -shared
 CP = cp
 RM = rm
 LN = ln -s -f
@@ -293,7 +293,7 @@
 CFLAGS_DEBUG = -Os -DMDNS_DEBUGMSGS=0
 OBJDIR = objects/prod
 BUILDDIR = build/prod
-STRIP = strip -S
+#STRIP = strip -S
 endif

 # Configure per-OS peculiarities
```

A.2 Konfigurationsdateien

A.2.1 Systemkonfiguration

```
#
# Automatically generated make config: don't edit
#
HAVE_DOT_CONFIG=y

#
# Build options
#
TOOLCHAIN_DIR="/home/andre/Uni/\\
    Studienarbeit/Finale/System/4.1.1-920t"
DOWNLOAD_SITE="http://arm.cirrus.com/files/\\
    linux/packages"
JLEVEL=1
```

```
#
# Linux kernel for EP9XXX
#
# KERNEL_VERSION_2_6_8_1 is not set
KERNEL_VERSION_2_6_17_14=y
DEFAULT_KERNEL_VERSION="2.6.17.14"
PACKAGE_ZIMAGE=y
# KERNEL_EP9301 is not set
# KERNEL_EP9302 is not set
# KERNEL_EP9302A is not set
# KERNEL_EP9307 is not set
# KERNEL_EP9307A is not set
# KERNEL_EP9312 is not set
# KERNEL_EP9315 is not set
KERNEL_EP9315A=y
ZIMAGE_CONFIG_FILE="/home/andre/Uni/Studienarbeit/\\
    Finale/System/kernelconfig"
TARGET_ZIMAGE_COPYTO="/home/andre/Uni/Studienarbeit/\\
    Finale/System/Gebaut"
DEFAULT_PACKAGE_BSP="9315A"

#
# Bootloader Options
#
# PACKAGE_DOWNLOAD is not set
# PACKAGE_REDBOOT is not set
# REDBOOT_ARCH_EDB9301 is not set
# REDBOOT_ARCH_EDB9302 is not set
# REDBOOT_ARCH_EDB9302A is not set
# REDBOOT_ARCH_EDB9307 is not set
# REDBOOT_ARCH_EDB9307A is not set
# REDBOOT_ARCH_EDB9312 is not set
# REDBOOT_ARCH_EDB9315 is not set
# REDBOOT_ARCH_EDB9315A is not set

#
# Individual Package Selection for the target
#

#
# Basic Root Filesystem
#
PACKAGE_BUSYBOX=y
```

```
PACKAGE_BUSYBOX_CONFIG="packages/busybox/\\
    busybox.config"
PACKAGE_UCLIBC=y
# ENABLE_LOCALE is not set
PACKAGE_ZLIB=y

#
# Other
#
# PACKAGE_MODULE_TOOLS is not set
# PACKAGE_GDB_SERVER is not set
# PACKAGE_OPENSSL is not set
# PACKAGE_GREP is not set

#
# Group packages
#
# NETWORK_SUPPORT is not set
AUDIO_SUPPORT=y
PACKAGE_ALSA=y
PACKAGE_ALSA_LIB=y
PACKAGE_ALSA_UTILS=y
# PACKAGE_MADPLAY is not set
# PACKAGE_MPG123 is not set
# PACKAGE_LIBID3TAG is not set
# PACKAGE_LIBMAD is not set
# GRAPHIC_SUPPORT is not set
HW_SUPPORT=y
# PACKAGE_TSLIB is not set
# PACKAGE_IRDA_TOOLS is not set
# PACKAGE_MTD is not set
PACKAGE_SYSFSUTILS=y
# PACKAGE_HOTPLUG is not set
# PACKAGE_FBSET is not set
# PACKAGE_LIRC is not set
# PACKAGE_LIRC_EDB9302A is not set
# PACKAGE_LIRC_EDB9307 is not set
# PACKAGE_LIRC_EDB9307A is not set
# PACKAGE_LIRC_EDB9312 is not set
# PACKAGE_LIRC_EDB9315 is not set
# PACKAGE_LIRC_EDB9315A is not set
# PACKAGE_LIBUSB is not set
# PACKAGE_USBUTILS is not set
```

```
# PACKAGE_DOSFS_TOOLS is not set
# PACKAGE_E2FSPROGS is not set

#
# Example Applications
#
# PACKAGE_FUBAR is not set

#
# Target Options
#
TARGET_ROOTFS_EXT2=y
TARGET_ROOTFS_EXT2_BLOCKS=0
TARGET_ROOTFS_EXT2_INODES=0
TARGET_ROOTFS_EXT2_RESBLKS=0
TARGET_ROOTFS_EXT2_SQUASH=y
TARGET_ROOTFS_EXT2_GZ=y
TARGET_ROOTFS_EXT2_COPYTO=""
# TARGET_ROOTFS_JFFS2 is not set
# TARGET_ROOTFS_CRAMFS is not set
```

A.2.2 Kernelkonfiguration

```
#
# Automatically generated make config: don't edit
# Linux kernel version: 2.6.17.14
# Tue May 22 23:17:12 2007
#
CONFIG_ARM=y
CONFIG_MMU=y
CONFIG_RWSEM_GENERIC_SPINLOCK=y
CONFIG_GENERIC_HWEIGHT=y
CONFIG_GENERIC_CALIBRATE_DELAY=y
CONFIG_VECTORS_BASE=0xffff0000

#
# Code maturity level options
#
CONFIG_EXPERIMENTAL=y
CONFIG_BROKEN_ON_SMP=y
CONFIG_LOCK_KERNEL=y
CONFIG_INIT_ENV_ARG_LIMIT=32
```

```
#
# General setup
#
CONFIG_LOCALVERSION=""
CONFIG_LOCALVERSION_AUTO=y
CONFIG_SWAP=y
CONFIG_SYSVIPC=y
CONFIG_POSIX_MQUEUE=y
# CONFIG_BSD_PROCESS_ACCT is not set
CONFIG_SYSCTL=y
CONFIG_AUDIT=y
# CONFIG_IKCONFIG is not set
# CONFIG_RELAY is not set
CONFIG_INITRAMFS_SOURCE=""
CONFIG_UID16=y
# CONFIG_CC_OPTIMIZE_FOR_SIZE is not set
CONFIG_EMBEDDED=y
CONFIG_KALLSYMS=y
# CONFIG_KALLSYMS_EXTRA_PASS is not set
CONFIG_HOTPLUG=y
CONFIG_PRINTK=y
CONFIG_BUG=y
CONFIG_ELF_CORE=y
CONFIG_BASE_FULL=y
CONFIG_FUTEX=y
CONFIG_EPOLL=y
CONFIG_SHMEM=y
CONFIG_SLAB=y
# CONFIG_TINY_SHMEM is not set
CONFIG_BASE_SMALL=0
# CONFIG_SLOB is not set
CONFIG_OBSOLETE_INTERMODULE=y

#
# Loadable module support
#
# CONFIG_MODULES is not set

#
# Block layer
#
# CONFIG_BLK_DEV_IO_TRACE is not set
```

```
#
# IO Schedulers
#
CONFIG_IOSCHED_NOOP=y
CONFIG_IOSCHED_AS=y
# CONFIG_IOSCHED_DEADLINE is not set
# CONFIG_IOSCHED_CFQ is not set
# CONFIG_DEFAULT_AS is not set
# CONFIG_DEFAULT_DEADLINE is not set
# CONFIG_DEFAULT_CFQ is not set
CONFIG_DEFAULT_NOOP=y
CONFIG_DEFAULT_IOSCHED="noop"

#
# System Type
#
# CONFIG_ARCH_CLPS7500 is not set
# CONFIG_ARCH_CLPS711X is not set
# CONFIG_ARCH_CO285 is not set
# CONFIG_ARCH_EBSA110 is not set
CONFIG_ARCH_EP93XX=y
# CONFIG_ARCH_FOOTBRIDGE is not set
# CONFIG_ARCH_INTEGRATOR is not set
# CONFIG_ARCH_IOP3XX is not set
# CONFIG_ARCH_IXP4XX is not set
# CONFIG_ARCH_IXP2000 is not set
# CONFIG_ARCH_IXP23XX is not set
# CONFIG_ARCH_L7200 is not set
# CONFIG_ARCH_PXA is not set
# CONFIG_ARCH_RPC is not set
# CONFIG_ARCH_SA1100 is not set
# CONFIG_ARCH_S3C2410 is not set
# CONFIG_ARCH_SHARK is not set
# CONFIG_ARCH_LH7A40X is not set
# CONFIG_ARCH_OMAP is not set
# CONFIG_ARCH_VERSATILE is not set
# CONFIG_ARCH_REALVIEW is not set
# CONFIG_ARCH_IMX is not set
# CONFIG_ARCH_H720X is not set
# CONFIG_ARCH_AAEC2000 is not set
# CONFIG_ARCH_AT91RM9200 is not set
```

```
#
# Cirrus EP93xx Implementation Options
#
# CONFIG_MACH_EDB9301 is not set
# CONFIG_MACH_EDB9302 is not set
# CONFIG_MACH_EDB9302A is not set
# CONFIG_MACH_EDB9307 is not set
# CONFIG_MACH_EDB9307A is not set
# CONFIG_MACH_EDB9312 is not set
# CONFIG_MACH_EDB9315 is not set
CONFIG_MACH_EDB9315A=y
CONFIG_EP93XX_SDCS0=y
# CONFIG_MACH_GESBC9312 is not set
# CONFIG_MACH_TS72XX is not set

#
# Processor Type
#
CONFIG_CPU_32=y
CONFIG_CPU_ARM920T=y
CONFIG_CPU_32v4=y
CONFIG_CPU_ABRT_EV4T=y
CONFIG_CPU_CACHE_V4WT=y
CONFIG_CPU_CACHE_VIVT=y
CONFIG_CPU_COPY_V4WB=y
CONFIG_CPU_TLB_V4WBI=y

#
# Processor Features
#
CONFIG_ARM_THUMB=y
# CONFIG_CPU_ICACHE_DISABLE is not set
# CONFIG_CPU_DCACHE_DISABLE is not set
# CONFIG_CPU_DCACHE_WRITETHROUGH is not set
CONFIG_ARM_VIC=y

#
# Bus support
#
CONFIG_ARM_AMBA=y

#
# PCCARD (PCMCIA/CardBus) support
```

```
#
# CONFIG_PCCARD is not set

#
# Kernel Features
#
CONFIG_PREEMPT=y
# CONFIG_NO_IDLE_HZ is not set
CONFIG_HZ=100
# CONFIG_AEABI is not set
# CONFIG_ARCH_DISCONTIGMEM_ENABLE is not set
CONFIG_SELECT_MEMORY_MODEL=y
CONFIG_FLATMEM_MANUAL=y
# CONFIG_DISCONTIGMEM_MANUAL is not set
# CONFIG_SPARSEMEM_MANUAL is not set
CONFIG_FLATMEM=y
CONFIG_FLAT_NODE_MEM_MAP=y
# CONFIG_SPARSEMEM_STATIC is not set
CONFIG_SPLIT_PTLOCK_CPUS=4096
CONFIG_ALIGNMENT_TRAP=y

#
# Boot options
#
CONFIG_ZBOOT_ROM_TEXT=0x0
CONFIG_ZBOOT_ROM_BSS=0x0
CONFIG_CMDLINE="root=/dev/ram console=ttyAM"
# CONFIG_XIP_KERNEL is not set

#
# Floating point emulation
#

#
# At least one emulation must be selected
#
CONFIG_FPE_NWFPE=y
# CONFIG_FPE_NWFPE_XP is not set
# CONFIG_FPE_FASTFPE is not set

#
# Userspace binary formats
#
```

```
CONFIG_BINFMT_ELF=y
CONFIG_BINFMT_AOUT=y
CONFIG_BINFMT_MISC=y
# CONFIG_ARTHUR is not set

#
# Power management options
#
# CONFIG_PM is not set
# CONFIG_APM is not set

#
# Networking
#
CONFIG_NET=y

#
# Networking options
#
# CONFIG_NETDEBUG is not set
CONFIG_PACKET=y
# CONFIG_PACKET_MMAP is not set
CONFIG_UNIX=y
# CONFIG_NET_KEY is not set
CONFIG_INET=y
CONFIG_IP_MULTICAST=y
# CONFIG_IP_ADVANCED_ROUTER is not set
CONFIG_IP_FIB_HASH=y
CONFIG_IP_PNP=y
# CONFIG_IP_PNP_DHCP is not set
# CONFIG_IP_PNP_BOOTP is not set
# CONFIG_IP_PNP_RARP is not set
# CONFIG_NET_IPIP is not set
# CONFIG_NET_IPGRE is not set
# CONFIG_IP_MROUTE is not set
# CONFIG_ARPD is not set
# CONFIG_SYN_COOKIES is not set
# CONFIG_INET_AH is not set
# CONFIG_INET_ESP is not set
# CONFIG_INET_IPCOMP is not set
# CONFIG_INET_XFRM_TUNNEL is not set
# CONFIG_INET_TUNNEL is not set
# CONFIG_INET_DIAG is not set
```

```
# CONFIG_TCP_CONG_ADVANCED is not set
CONFIG_TCP_CONG_BIC=y
# CONFIG_IPV6 is not set
# CONFIG_INET6_XFRM_TUNNEL is not set
# CONFIG_INET6_TUNNEL is not set
# CONFIG_NETFILTER is not set

#
# DCCP Configuration (EXPERIMENTAL)
#
# CONFIG_IP_DCCP is not set

#
# SCTP Configuration (EXPERIMENTAL)
#
# CONFIG_IP_SCTP is not set

#
# TIPC Configuration (EXPERIMENTAL)
#
# CONFIG_TIPC is not set
# CONFIG_ATM is not set
# CONFIG_BRIDGE is not set
# CONFIG_VLAN_8021Q is not set
# CONFIG_DECNET is not set
# CONFIG_LLC2 is not set
# CONFIG_IPX is not set
# CONFIG_ATALK is not set
# CONFIG_X25 is not set
# CONFIG_LAPB is not set
# CONFIG_NET_DIVERT is not set
# CONFIG_ECONET is not set
# CONFIG_WAN_ROUTER is not set

#
# QoS and/or fair queueing
#
# CONFIG_NET_SCHED is not set

#
# Network testing
#
# CONFIG_NET_PKTGEN is not set
```

```
# CONFIG_HAMRADIO is not set
# CONFIG_IRDA is not set
# CONFIG_BT is not set
# CONFIG_IEEE80211 is not set
CONFIG_WIRELESS_EXT=y

#
# Device Drivers
#

#
# Generic Driver Options
#
CONFIG_STANDALONE=y
CONFIG_PREVENT_FIRMWARE_BUILD=y
# CONFIG_FW_LOADER is not set

#
# Connector - unified userspace <-> kernelspace linker
#
# CONFIG_CONNECTOR is not set

#
# Memory Technology Devices (MTD)
#
CONFIG_MTD=y
# CONFIG_MTD_DEBUG is not set
# CONFIG_MTD_CONCAT is not set
CONFIG_MTD_PARTITIONS=y
CONFIG_MTD_REDBOOT_PARTS=y
CONFIG_MTD_REDBOOT_DIRECTORY_BLOCK=-1
CONFIG_MTD_REDBOOT_PARTS_UNALLOCATED=y
# CONFIG_MTD_REDBOOT_PARTS_READONLY is not set
# CONFIG_MTD_CMDLINE_PARTS is not set
# CONFIG_MTD_AFS_PARTS is not set

#
# User Modules And Translation Layers
#
CONFIG_MTD_CHAR=y
CONFIG_MTD_BLOCK=y
# CONFIG_FTL is not set
CONFIG_NFTL=y
```

```
# CONFIG_NFTL_RW is not set
# CONFIG_INFTL is not set
# CONFIG_RFD_FTL is not set

#
# RAM/ROM/Flash chip drivers
#
CONFIG_MTD_CFI=y
# CONFIG_MTD_JEDECPROBE is not set
CONFIG_MTD_GEN_PROBE=y
# CONFIG_MTD_CFI_ADV_OPTIONS is not set
CONFIG_MTD_MAP_BANK_WIDTH_1=y
CONFIG_MTD_MAP_BANK_WIDTH_2=y
CONFIG_MTD_MAP_BANK_WIDTH_4=y
# CONFIG_MTD_MAP_BANK_WIDTH_8 is not set
# CONFIG_MTD_MAP_BANK_WIDTH_16 is not set
# CONFIG_MTD_MAP_BANK_WIDTH_32 is not set
CONFIG_MTD_CFI_I1=y
CONFIG_MTD_CFI_I2=y
# CONFIG_MTD_CFI_I4 is not set
# CONFIG_MTD_CFI_I8 is not set
CONFIG_MTD_CFI_INTELEXT=y
CONFIG_MTD_CFI_AMDSTD=y
CONFIG_MTD_CFI_STAA=y
CONFIG_MTD_CFI_UTIL=y
CONFIG_MTD_RAM=y
# CONFIG_MTD_ROM is not set
# CONFIG_MTD_ABSENT is not set
# CONFIG_MTD_OBSOLETE_CHIPS is not set

#
# Mapping drivers for chip access
#
# CONFIG_MTD_COMPLEX_MAPPINGS is not set
CONFIG_MTD_PHYSMAP=y
CONFIG_MTD_PHYSMAP_START=0x0
CONFIG_MTD_PHYSMAP_LEN=0x0
CONFIG_MTD_PHYSMAP_BANKWIDTH=1
# CONFIG_MTD_ARM_INTEGRATOR is not set
# CONFIG_MTD_EDB93XX is not set
# CONFIG_MTD_PLATRAM is not set

#
```

```
# Self-contained MTD device drivers
#
# CONFIG_MTD_SLRAM is not set
# CONFIG_MTD_PHRAM is not set
# CONFIG_MTD_MTDRAM is not set
# CONFIG_MTD_BLOCK2MTD is not set

#
# Disk-On-Chip Device Drivers
#
# CONFIG_MTD_DOC2000 is not set
# CONFIG_MTD_DOC2001 is not set
# CONFIG_MTD_DOC2001PLUS is not set

#
# NAND Flash Device Drivers
#
CONFIG_MTD_NAND=y
CONFIG_MTD_NAND_VERIFY_WRITE=y
CONFIG_MTD_NAND_EDB93xx=y
CONFIG_MTD_NAND_IDS=y
# CONFIG_MTD_NAND_DISKONCHIP is not set
# CONFIG_MTD_NAND_NANDSIM is not set

#
# OneNAND Flash Device Drivers
#
# CONFIG_MTD_ONENAND is not set

#
# Parallel port support
#
# CONFIG_PARPORT is not set

#
# Plug and Play support
#

#
# Block devices
#
# CONFIG_BLK_DEV_COW_COMMON is not set
CONFIG_BLK_DEV_LOOP=y
```

```
CONFIG_BLK_DEV_CRYPTOLOOP=y
# CONFIG_BLK_DEV_NBD is not set
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_RAM_COUNT=2
CONFIG_BLK_DEV_RAM_SIZE=32768
CONFIG_BLK_DEV_INITRD=y
# CONFIG_CDROM_PKTCDVD is not set
# CONFIG_ATA_OVER_ETH is not set

#
# ATA/ATAPI/MFM/RLL support
#
# CONFIG_IDE is not set

#
# SCSI device support
#
# CONFIG_RAID_ATTRS is not set
# CONFIG_SCSI is not set

#
# Multi-device support (RAID and LVM)
#
# CONFIG_MD is not set

#
# Fusion MPT device support
#
# CONFIG_FUSION is not set

#
# IEEE 1394 (FireWire) support
#

#
# I2O device support
#

#
# Network device support
#
CONFIG_NETDEVICES=y
# CONFIG_DUMMY is not set
```

```
# CONFIG_BONDING is not set
# CONFIG_EQUALIZER is not set
# CONFIG_TUN is not set

#
# PHY device support
#
# CONFIG_PHYLIB is not set

#
# Ethernet (10 or 100Mbit)
#
CONFIG_NET_ETHERNET=y
# CONFIG_MII is not set
CONFIG_EP93XX_ETH=y
# CONFIG_SMC91X is not set
# CONFIG_DM9000 is not set

#
# Ethernet (1000 Mbit)
#

#
# Ethernet (10000 Mbit)
#

#
# Token Ring devices
#

#
# Wireless LAN (non-hamradio)
#
CONFIG_NET_RADIO=y
# CONFIG_NET_WIRELESS_RTNETLINK is not set

#
# Obsolete Wireless cards support (pre-802.11)
#
# CONFIG_STRIP is not set
# CONFIG_HOSTAP is not set

#
```

```
# Wan interfaces
#
# CONFIG_WAN is not set
# CONFIG_PPP is not set
# CONFIG_SLIP is not set
# CONFIG_SHAPER is not set
# CONFIG_NETCONSOLE is not set
# CONFIG_NETPOLL is not set
# CONFIG_NET_POLL_CONTROLLER is not set

#
# ISDN subsystem
#
# CONFIG_ISDN is not set

#
# Input device support
#
CONFIG_INPUT=y

#
# Userland interfaces
#
# CONFIG_INPUT_MOUSEDEV is not set
# CONFIG_INPUT_JOYDEV is not set
# CONFIG_INPUT_TSDEV is not set
# CONFIG_INPUT_EVDEV is not set
# CONFIG_INPUT_EVBUG is not set

#
# Input Device Drivers
#
# CONFIG_INPUT_KEYBOARD is not set
# CONFIG_INPUT_MOUSE is not set
# CONFIG_INPUT_JOYSTICK is not set
# CONFIG_INPUT_TOUCHSCREEN is not set
# CONFIG_INPUT_MISC is not set

#
# Hardware I/O ports
#
CONFIG_SERIO=y
# CONFIG_SERIO_SERPORT is not set
```

```
# CONFIG_SERIO_AMBAKMI is not set
CONFIG_SERIO_LIBPS2=y
# CONFIG_SERIO_RAW is not set
# CONFIG_GAMEPORT is not set

#
# Character devices
#
CONFIG_VT=y
CONFIG_VT_CONSOLE=y
CONFIG_HW_CONSOLE=y
# CONFIG_SERIAL_NONSTANDARD is not set

#
# Serial drivers
#
# CONFIG_SERIAL_8250 is not set

#
# Non-8250 serial port support
#
CONFIG_SERIAL_AMBA_PL010=y
CONFIG_SERIAL_AMBA_PL010_CONSOLE=y
# CONFIG_SERIAL_AMBA_PL011 is not set
# CONFIG_EP93XX_SERIAL_FLASH is not set
CONFIG_SERIAL_CORE=y
CONFIG_SERIAL_CORE_CONSOLE=y
CONFIG_UNIX98_PTYS=y
CONFIG_LEGACY_PTYS=y
CONFIG_LEGACY_PTY_COUNT=256

#
# IPMI
#
# CONFIG_IPMI_HANDLER is not set

#
# Watchdog Cards
#
CONFIG_WATCHDOG=y
# CONFIG_WATCHDOG_NOWAYOUT is not set

#
```

```
# Watchdog Device Drivers
#
# CONFIG_SOFT_WATCHDOG is not set
CONFIG_EP93XX_WATCHDOG=y
# CONFIG_NVRAM is not set
# CONFIG_DTLK is not set
# CONFIG_R3964 is not set

#
# Ftape, the floppy tape device driver
#
# CONFIG_RAW_DRIVER is not set

#
# TPM devices
#
# CONFIG_TCG_TPM is not set
# CONFIG_TELCLOCK is not set

#
# I2C support
#
CONFIG_I2C=y
CONFIG_I2C_CHARDEV=y

#
# I2C Algorithms
#
CONFIG_I2C_ALGOBIT=y
# CONFIG_I2C_ALGOPCF is not set
# CONFIG_I2C_ALGOPCA is not set

#
# I2C Hardware Bus support
#
CONFIG_I2C_EP93XX=y
# CONFIG_I2C_PARPORT_LIGHT is not set
# CONFIG_I2C_PCA_ISA is not set

#
# Miscellaneous I2C Chip support
#
# CONFIG_SENSORS_DS1337 is not set
```

```
# CONFIG_SENSORS_DS1374 is not set
# CONFIG_SENSORS_EEPROM is not set
# CONFIG_SENSORS_PCF8574 is not set
# CONFIG_SENSORS_PCA9539 is not set
# CONFIG_SENSORS_PCF8591 is not set
# CONFIG_SENSORS_MAX6875 is not set
# CONFIG_I2C_DEBUG_CORE is not set
# CONFIG_I2C_DEBUG_ALGO is not set
# CONFIG_I2C_DEBUG_BUS is not set
# CONFIG_I2C_DEBUG_CHIP is not set

#
# SPI support
#
# CONFIG_SPI is not set
# CONFIG_SPI_MASTER is not set

#
# Dallas's 1-wire bus
#
# CONFIG_W1 is not set

#
# Hardware Monitoring support
#
# CONFIG_HWMON is not set
# CONFIG_HWMON_VID is not set

#
# Misc devices
#

#
# LED devices
#
# CONFIG_NEW_LEDS is not set

#
# LED drivers
#

#
# LED Triggers
```

```
#

#
# Multimedia devices
#
# CONFIG_VIDEO_DEV is not set
CONFIG_VIDEO_V4L2=y

#
# Digital Video Broadcasting Devices
#
# CONFIG_DVB is not set

#
# Graphics support
#
# CONFIG_FB is not set

#
# Console display driver support
#
# CONFIG_VGA_CONSOLE is not set
CONFIG_DUMMY_CONSOLE=y

#
# Sound
#
CONFIG_SOUND=y

#
# Advanced Linux Sound Architecture
#
CONFIG_SND=y
CONFIG_SND_TIMER=y
CONFIG_SND_PCM=y
# CONFIG_SND_SEQUENCER is not set
CONFIG_SND_OSSEMUL=y
CONFIG_SND_MIXER_OSS=y
CONFIG_SND_PCM_OSS=y
CONFIG_SND_PCM_OSS_PLUGINS=y
# CONFIG_SND_DYNAMIC_MINORS is not set
CONFIG_SND_SUPPORT_OLD_API=y
# CONFIG_SND_VERBOSE_PROCFS is not set
```

```
# CONFIG_SND_VERBOSE_PRINTK is not set
# CONFIG_SND_DEBUG is not set

#
# Generic devices
#
# CONFIG_SND_DUMMY is not set
# CONFIG_SND_MTPAV is not set
# CONFIG_SND_SERIAL_U16550 is not set
# CONFIG_SND_MPU401 is not set

#
# ALSA ARM devices
#
CONFIG_SND_EP93XX_IIS=y
CONFIG_CODEC_CS4271=y
# CONFIG_SND_EP93XX_AC97 is not set
# CONFIG_SND_ARMAACI is not set

#
# Open Sound System
#
# CONFIG_SOUND_PRIME is not set

#
# USB support
#
CONFIG_USB_ARCH_HAS_HCD=y
CONFIG_USB_ARCH_HAS_OHCI=y
# CONFIG_USB_ARCH_HAS_EHCI is not set
# CONFIG_USB is not set

#
# NOTE: USB_STORAGE enables SCSI, and ,SCSI disk
support‘
#

#
# USB Gadget Support
#
# CONFIG_USB_GADGET is not set
# CONFIG_USB_GADGET_NET2280 is not set
# CONFIG_USB_GADGET_PXA2XX is not set
```

```
# CONFIG_USB_GADGET_EP931X is not set
# CONFIG_USB_GADGET_GOKU is not set
# CONFIG_USB_GADGET_LH7A40X is not set
# CONFIG_USB_GADGET_OMAP is not set
# CONFIG_USB_GADGET_AT91 is not set
# CONFIG_USB_GADGET_DUMMY_HCD is not set
# CONFIG_USB_ZERO is not set
# CONFIG_USB_ETH is not set
# CONFIG_USB_GADGETFS is not set
# CONFIG_USB_FILE_STORAGE is not set
# CONFIG_USB_G_SERIAL is not set

#
# MMC/SD Card support
#
# CONFIG_MMC is not set

#
# Real Time Clock
#
CONFIG_RTC_LIB=y
CONFIG_RTC_CLASS=y
CONFIG_RTC_HCTOSYS=y
CONFIG_RTC_HCTOSYS_DEVICE="rtc0"

#
# RTC interfaces
#
CONFIG_RTC_INTF_SYSFS=y
CONFIG_RTC_INTF_PROC=y
CONFIG_RTC_INTF_DEV=y

#
# RTC drivers
#
# CONFIG_RTC_DRV_X1205 is not set
# CONFIG_RTC_DRV_DS1672 is not set
# CONFIG_RTC_DRV_PCF8563 is not set
# CONFIG_RTC_DRV_RS5C372 is not set
# CONFIG_RTC_DRV_M48T86 is not set
CONFIG_RTC_DRV_EP93XX=y
# CONFIG_RTC_DRV_EDB93XX_EMBEDDED is not set
CONFIG_RTC_DRV_ISL1208=y
```

```
# CONFIG_RTC_DRV_TEST is not set

#
# File systems
#
CONFIG_EXT2_FS=y
CONFIG_EXT2_FS_XATTR=y
CONFIG_EXT2_FS_POSIX_ACL=y
CONFIG_EXT2_FS_SECURITY=y
# CONFIG_EXT2_FS_XIP is not set
# CONFIG_EXT3_FS is not set
CONFIG_FS_MBCACHE=y
# CONFIG_REISERFS_FS is not set
# CONFIG_JFS_FS is not set
CONFIG_FS_POSIX_ACL=y
# CONFIG_XFS_FS is not set
# CONFIG_OCFS2_FS is not set
# CONFIG_MINIX_FS is not set
CONFIG_ROMFS_FS=y
# CONFIG_INOTIFY is not set
# CONFIG_QUOTA is not set
CONFIG_DNOTIFY=y
# CONFIG_AUTOFS_FS is not set
CONFIG_AUTOFS4_FS=y
# CONFIG_FUSE_FS is not set

#
# CD-ROM/DVD Filesystems
#
# CONFIG_ISO9660_FS is not set
# CONFIG_UDF_FS is not set

#
# DOS/FAT/NT Filesystems
#
# CONFIG_MSDOS_FS is not set
# CONFIG_VFAT_FS is not set
# CONFIG_NTFS_FS is not set

#
# Pseudo filesystems
#
CONFIG_PROC_FS=y
```

```
CONFIG_SYSFS=y
# CONFIG_TMPFS is not set
# CONFIG_HUGETLB_PAGE is not set
CONFIG_RAMFS=y
# CONFIG_CONFIGFS_FS is not set

#
# Miscellaneous filesystems
#
# CONFIG_ADFS_FS is not set
# CONFIG_AFFS_FS is not set
# CONFIG_HFS_FS is not set
# CONFIG_HFSPLUS_FS is not set
# CONFIG_BEFS_FS is not set
# CONFIG_BFS_FS is not set
# CONFIG_EFS_FS is not set
# CONFIG_JFFS_FS is not set
# CONFIG_JFFS2_FS is not set
# CONFIG_CRAMFS is not set
# CONFIG_VXFS_FS is not set
# CONFIG_HPFS_FS is not set
# CONFIG_QNX4FS_FS is not set
# CONFIG_SYSV_FS is not set
# CONFIG_UFS_FS is not set

#
# Network File Systems
#
# CONFIG_NFS_FS is not set
# CONFIG_NFSD is not set
# CONFIG_SMB_FS is not set
# CONFIG_CIFS is not set
# CONFIG_NCP_FS is not set
# CONFIG_CODA_FS is not set
# CONFIG_AFS_FS is not set
# CONFIG_9P_FS is not set

#
# Partition Types
#
CONFIG_PARTITION_ADVANCED=y
# CONFIG_ACORN_PARTITION is not set
# CONFIG_OSF_PARTITION is not set
```

```
# CONFIG_AMIGA_PARTITION is not set
# CONFIG_ATARI_PARTITION is not set
# CONFIG_MAC_PARTITION is not set
CONFIG_MSDOS_PARTITION=y
# CONFIG_BSD_DISKLABEL is not set
# CONFIG_MINIX_SUBPARTITION is not set
# CONFIG_SOLARIS_X86_PARTITION is not set
# CONFIG_UNIXWARE_DISKLABEL is not set
# CONFIG_LDM_PARTITION is not set
# CONFIG_SGI_PARTITION is not set
# CONFIG_ULTRIX_PARTITION is not set
# CONFIG_SUN_PARTITION is not set
# CONFIG_KARMA_PARTITION is not set
# CONFIG_EFI_PARTITION is not set

#
# Native Language Support
#
CONFIG_NLS=y
CONFIG_NLS_DEFAULT="iso8859-1"
CONFIG_NLS_CODEPAGE_437=y
# CONFIG_NLS_CODEPAGE_737 is not set
# CONFIG_NLS_CODEPAGE_775 is not set
# CONFIG_NLS_CODEPAGE_850 is not set
# CONFIG_NLS_CODEPAGE_852 is not set
# CONFIG_NLS_CODEPAGE_855 is not set
# CONFIG_NLS_CODEPAGE_857 is not set
# CONFIG_NLS_CODEPAGE_860 is not set
# CONFIG_NLS_CODEPAGE_861 is not set
# CONFIG_NLS_CODEPAGE_862 is not set
# CONFIG_NLS_CODEPAGE_863 is not set
# CONFIG_NLS_CODEPAGE_864 is not set
# CONFIG_NLS_CODEPAGE_865 is not set
# CONFIG_NLS_CODEPAGE_866 is not set
# CONFIG_NLS_CODEPAGE_869 is not set
# CONFIG_NLS_CODEPAGE_936 is not set
# CONFIG_NLS_CODEPAGE_950 is not set
# CONFIG_NLS_CODEPAGE_932 is not set
# CONFIG_NLS_CODEPAGE_949 is not set
# CONFIG_NLS_CODEPAGE_874 is not set
# CONFIG_NLS_ISO8859_8 is not set
# CONFIG_NLS_CODEPAGE_1250 is not set
# CONFIG_NLS_CODEPAGE_1251 is not set
```

```
# CONFIG_NLS_ASCII is not set
CONFIG_NLS_ISO8859_1=y
# CONFIG_NLS_ISO8859_2 is not set
# CONFIG_NLS_ISO8859_3 is not set
# CONFIG_NLS_ISO8859_4 is not set
# CONFIG_NLS_ISO8859_5 is not set
# CONFIG_NLS_ISO8859_6 is not set
# CONFIG_NLS_ISO8859_7 is not set
# CONFIG_NLS_ISO8859_9 is not set
# CONFIG_NLS_ISO8859_13 is not set
# CONFIG_NLS_ISO8859_14 is not set
# CONFIG_NLS_ISO8859_15 is not set
# CONFIG_NLS_KOI8_R is not set
# CONFIG_NLS_KOI8_U is not set
CONFIG_NLS_UTF8=y

#
# Profiling support
#
# CONFIG_PROFILING is not set

#
# Kernel hacking
#
# CONFIG_PRINTK_TIME is not set
# CONFIG_MAGIC_SYSRQ is not set
# CONFIG_DEBUG_KERNEL is not set
CONFIG_LOG_BUF_SHIFT=14
# CONFIG_DEBUG_BUGVERBOSE is not set
# CONFIG_DEBUG_FS is not set
CONFIG_FRAME_POINTER=y
# CONFIG_UNWIND_INFO is not set
# CONFIG_DEBUG_USER is not set

#
# Security options
#
# CONFIG_KEYS is not set
# CONFIG_SECURITY is not set

#
# Cryptographic options
#
```

```
CONFIG_CRYPTO=y
# CONFIG_CRYPTO_HMAC is not set
# CONFIG_CRYPTO_NULL is not set
# CONFIG_CRYPTO_MD4 is not set
CONFIG_CRYPTO_MD5=y
CONFIG_CRYPTO_SHA1=y
# CONFIG_CRYPTO_SHA256 is not set
# CONFIG_CRYPTO_SHA512 is not set
# CONFIG_CRYPTO_WP512 is not set
# CONFIG_CRYPTO_TGR192 is not set
CONFIG_CRYPTO_DES=y
# CONFIG_CRYPTO_BLOWFISH is not set
# CONFIG_CRYPTO_TWOFISH is not set
# CONFIG_CRYPTO_SERPENT is not set
CONFIG_CRYPTO_AES=y
# CONFIG_CRYPTO_CAST5 is not set
# CONFIG_CRYPTO_CAST6 is not set
# CONFIG_CRYPTO_TEA is not set
CONFIG_CRYPTO_ARC4=y
# CONFIG_CRYPTO_KHAZAD is not set
# CONFIG_CRYPTO_ANUBIS is not set
# CONFIG_CRYPTO_DEFLATE is not set
# CONFIG_CRYPTO_MICHAEL_MIC is not set
# CONFIG_CRYPTO_CRC32C is not set
# CONFIG_CRYPTO_TEST is not set

#
# Hardware crypto devices
#

#
# Library routines
#
CONFIG_CRC_CCITT=y
# CONFIG_CRC16 is not set
CONFIG_CRC32=y
CONFIG_LIBCRC32C=y
```

A.3 Quelltexte der Testprogramme

A.3.1 inout.cpp

```
#include „RtAudio.h“
#include <iostream>

void usage(void) {
  std::cout << „\nuseage: call_inout buffersize
#buffers \n“;
  exit(0);
}

int inout(char *buffer, int buffer_size, void *)
{
   return 0;
}

int main(int argc, char *argv[])
{
  int buffer_size, num_buffers;
  RtAudio *audio = 0;
  char input;

  if (argc != 3 ) usage();

  buffer_size = (int) atoi(argv[1]);
  num_buffers = (int) atoi(argv[2]);

  try {
    audio = new RtAudio(0, 2, 0, 2,  RTAUDIO_SINT16,
48000, &buffer_size, num_buffers);
  }
  catch (RtError &error) {
    error.printMessage();
    exit(EXIT_FAILURE);
  }

  try {
    audio->setStreamCallback(&inout, NULL);
    audio->startStream();
  }
  catch (RtError &error) {
```

```
    error.printMessage();
    goto cleanup;
  }

  std::cout << „\nRunning ... press <enter> to quit
(buffer size = „ << buffer_size << „).\n“;
  std::cin.get(input);

  try {
    audio->stopStream();
  }
  catch (RtError &error) {
    error.printMessage();
  }

 cleanup:
  audio->closeStream();
  delete audio;

  return 0;
}
```

A.3.2 audio_reflector.c

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>


int main(int argc, char **argv) {
  if (argc < 2) {
    printf(„Usage: reflector <buffersize>\n“);
    return 1;
  }

  int s, rs, retval, bytes;
```

```
  struct sockaddr_in my_addr;
  struct sockaddr from;
  unsigned int size;
  fd_set rfds;

  int buffersize = atoi(argv[1]);
  char *in_buffer = (char *) malloc(buffersize);
  char *out_buffer = (char *) malloc(buffersize);

  printf(„Buffersize: %i \n", buffersize);

  s = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
  if (s < 0) {
    perror(„Creating socket");
    return -1;
  }

  my_addr.sin_family = AF_INET;
  my_addr.sin_port = htons(35000);
  my_addr.sin_addr.s_addr = INADDR_ANY;
  memset(&(my_addr.sin_zero), 0, 8);

  rs = bind(s, (struct sockaddr *) &my_addr,
sizeof(struct sockaddr));
  if (rs < 0) {
    perror(„Binding socket");
    return -1;
  }

  while(1) {
    FD_ZERO(&rfds);
    FD_SET(s, &rfds);

    retval = select(s + 1, &rfds, NULL, NULL, NULL);

    if (retval >  0) {
      size = sizeof(from);
      bytes = recvfrom(s, in_buffer, buffersize, 0,
&from, &size);

      if (bytes < 0) {
        perror(„Receiving");
      } else {
```

```
        memcpy(out_buffer, in_buffer, bytes);
        bytes = sendto(s, out_buffer, bytes, 0, &from,
size);
        if (bytes < 0) {
          perror(„Sending“);
        }
      }
    } else {
        printf(„No data arrived! \n“);
      }
    }

  close(s);
  return 0;
}
```

A.3.3 audio_reflector_client.cpp

```
#include <cstdio>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <unistd.h>

#include „RtAudio.h“


int s, channels;

int audio_handler(char *buffer, int buffer_size, void
*)
{
  send(s, buffer, buffer_size * channels * 2, 0);
  memset(buffer, 0, buffer_size * channels * 2);
  recv(s, buffer, buffer_size * channels * 2, MSG_
DONTWAIT);
  return 0;
}


int main(int argc, char **argv) {
```

```
  if (argc < 5) {
    printf(„Usage: reflector_client <destination>
<buffersize> <channels> <buffers>\n“);
    return 1;
  }

  struct sockaddr_in rem_addr;
  struct hostent *h;

  int buffers = atoi(argv[4]);
  int buffer_size = atoi(argv[2]);
  channels = atoi(argv[3]);
  printf(„Buffersize: %i \n“, channels * buffer_size *
sizeof(short));

  RtAudio *audio = 0;

  s = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
  if (s < 0) {
    perror(„Creating socket“);
    return -1;
  }
  printf(„Socket created\n“);

  h = gethostbyname(argv[1]);
  if (!h) {
    perror(„Unknown host“);
    return -3;
  }

  rem_addr.sin_family = AF_INET;
  rem_addr.sin_port = htons(35000);
  memcpy(&rem_addr.sin_addr.s_addr, h->h_addr, h->h_
length);
  memset(&(rem_addr.sin_zero), 0, 8);

  int rs = connect(s, (struct sockaddr *) &rem_addr,
sizeof(struct sockaddr));
  if (rs < 0) {
    perror(„Connecting failed“);
    return -2;
  }
  printf(„Connection prepared\n“);
```

```
  try {
    audio = new RtAudio(0, channels, 0, channels,
RTAUDIO_SINT16, 48000, &buffer_size, buffers);
    printf(„Actual Buffersize: %d\n“, buffer_size);
  } catch (RtError &error) {
    error.printMessage();
    exit(EXIT_FAILURE);
  }

  try {
    audio->setStreamCallback(audio_handler, NULL);
    audio->startStream();
  } catch(RtError &error) {
    error.printMessage();
    delete audio;
    close(s);
    return 0;
  }
  printf(„Audio initialized\n“);

  std::cout << „Hallo!“ << std::endl;
  std::cin.get();
  std::cout << „Tschau!“ << std::endl;

  try {
    audio->stopStream();
    audio->closeStream();
  } catch (RtError &error) {
    error.printMessage();
  }

  delete audio;
  close(s);

  return 0;
}
```

A.3.4 Messung der CPU-Last

Die Aufzeichnung der CPU-Last erfolgte durch folgendes BASH-Skript

```
# top | grep pid | awk '{print $6}' > logfile
```

Die so erzeugte Datei enthält zeilenweise aufgelistet die CPU-Belastung durch den Prozess mit der ID *pid*. Diese Datei wird auf dem Entwicklungs-PC durch folgendes Skript ausgewertet, d. h. die durchschnittliche CPU-Belastung wird bestimmt.

```
#!/usr/bin/python

import sys

def cpulast(fn):
     f = file(fn)
     i = 0
     summe = 0.0
     while 1:
          try:
               val = float(f.readline())
               i += 1
               summe += val
          except:
               break
     print "CPU-Last: " + str(summe / i)
     f.close()

cpulast(sys.argv[1])
```