Fachhochschule
Münster University of
Applied Sciences

Studiengang Systemtechnik

Vertiefungsrichtung Infotronics

DIPLOM 2011

BAPTISTE SOLIOZ

Ethernet Soundkarte

Dozent : Pierre-André Mudry

Experte : Peter Glösseköttter

Steinfurt, den 16. September 2011

HES-SO Valais

SI	TV
X	X

Données du travail de diplôme
Daten der Diplomarbeit

FO 1.2.02.07.AB
pof/31/01/2009

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2010/2011**	No TD / *Nr. DA* **it/2011/56**
Mandant / *Auftraggeber* ☐ HES—SO Valais ☐ Industrie ☒ Etablissement partenaire *Partnerinstitution* **Fachhochschule Münster**	Etudiants / *Studenten* **Baptiste Solioz** Professeur / *Dozent* **Pierre-André Mudry** **Joseph Moerschell**	Lieu d'exécution / *Ausführungsort* ☐ HES—SO Valais ☐ Industrie ☒ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte* *(données complètes)* **Peter Glösekötter** | Dozent der Parterninstitution)	

Titre / *Titel*

Digitale Soundkarte

Description et Objectifs / *Beschreibung und Ziele*

Es handelt sich um die Entwicklung einer rein digitalen Soundkarte. Dazu soll ein Mikrocontroller vom Typ STM32 mit modernem ARM Cortex M3 Kern verwendet werden.

Die Aufgabe besteht darin im uC eine USB Soundkarte zu implementieren die mit einer gängigen Soundkarte (z.B. SoundBlaster 16) kompatibel ist. Die Audioausgabe soll rein digital in Form eines SPDIF Signals erfolgen.

Dieses wird in einer zu diesem Projeckt parallel entwickelten digitalen Endstufe weiterverarbeitet, welche ein eingehendes SPDIF Signal mittels FPGA in ein PWM-Ausgangssignal wandelt.

Das Ziel ist eine rein digitale USB Soundkarte welchevmit niedrigen Verlusten direkt Ausgangsleistung erzeugt ohne dabei ein analoges Zwischensignal zu erzeugen. Ein heute typisches "Verstärkerbrummen" durch die Verstärkung von thermischem Rauschen soll bei dieser neuen Soundkarte nicht auftreten.

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:* 16.05.2011	Exposition publique / *Ausstellung Diplomarbeiten:* 02.09.2011
Remise du rapport / *Abgabe des Schlussberichts:* Spätestens am 23.09.2011	Défense orale / *Mündliche Verteidigung:* Woche 39

Signature ou visa / *Unterschrift oder Visum*

Responsable de l'orientation
Leiter der Vertiefungsrichtung: .....

[1] Etudiant/*Student*: .....

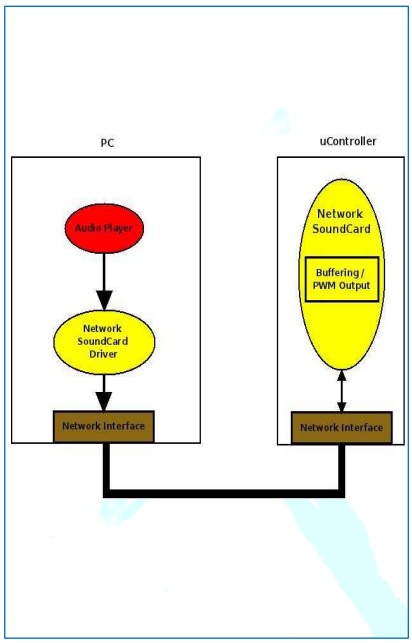
[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive et le caractère confidentiel du travail de diplôme qui lui est confié et des informations mises à sa disposition.*
Durch seine Unterschrift verpflichtet sich der Student, die Richtlinie einzuhalten sowie die Vertraulichkeit der Diplomarbeit und der dafür zur Verfügung gestellten Informationen zu wahren.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Ethernet Soundkarte

Diplomand/in Baptiste Solioz



Diplomarbeit
| 2011 |

Ziel des Projekts

Das Ziel dieses Projekts ist es, eine Netzwerksoundkarte zu entwickeln. Eine Soundkarte ist Teil der Hardware eines Rechnersystems und verarbeitet analoge und digitale Audiosignale.

Methoden | Experimente | Resultate

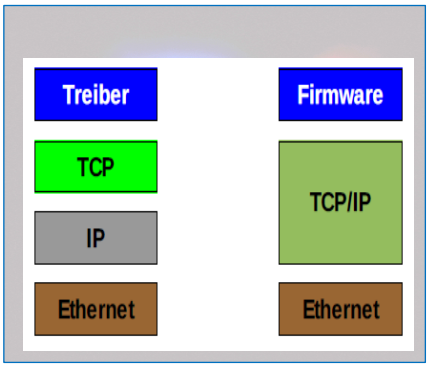
Der Rechner läuft unter dem Betriebsystem Linux. Am Anfang muss man eine Kernelmodul (Treiber) entwickeln, welches eine Soundkarte implementiert und Audiodaten an eine voreingestellte IP-Adresse streamt.
Dann entwickelt man einer Firmware für eine Mikrocontrollerplatine, welche eine Netzwerksoundkarte darstellt. Die Soundkarte bekommt mehrere Audiokanäle von der Netzwerkschnittstelle und macht eine Trennung um alle die Kanäle einzeln zu verarbeiten.
Die Musik muss jetzt gespielt werden. Es ist das Ziel, einen digitalen Ausgang zu benutzen, also benutzt man PWM-Ausgänge um den Sound zu spielen.
Zum Schluss entwickelt man einer eigenen Platine für eine autarke Ethernet Soundkarte. Die Platine muss mit einer einzelnen Spannung versorgt werden. Ein Teil mit Leistungselektronik erlaubt die Erzeugung von leistungsstarken PWMs der verschiedenen Spannungsniveaus. Diese PWMs werden an zwei Lautsprecheranschlüssen bereitgestellt.

Studiengang
Systemtechnik

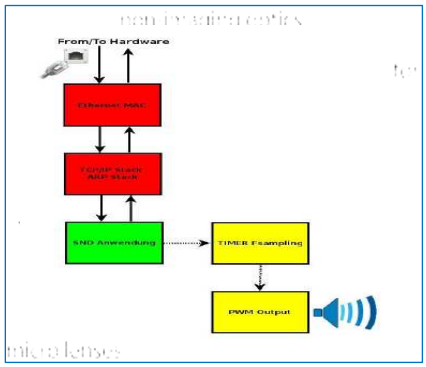
Anwendungsbereich
Infotronics

Verantwortliche/r Dozent/in
Dr Pierre-André Mudry
pandre.mudry@hevs.ch

Partner
Fachhochschule Münster



Beschreibung der Schichten. Auf dem Rechner sind die Schichten gespalten und es gibt eine Kommunikation zwischen allen Schichten.



Struktur der Soundkarte Firmware. Es gibt eine Ethernet Schnittstelle und die Implementierung der Anwendung.

HES-SO Wallis
Route du Rawyl 47
1950 Sitten

Tel. 027 606 85 11
Web www.hevs.ch

1 INHALTSÜBERSICHT

2 ABBILDUNGSVERZEICHNIS

3 EINLEITUNG

Das Ziel dieses Projekts ist es, eine Netzwerksoundkarte zu entwickeln. Eine Soundkarte ist Teil der Hardware eines Rechnersystems und verarbeitet analoge und digitale Audiosignale.

Eine Soundkarte kann intern über den PCI-Bus oder extern über eine USB-Schnittstelle Daten austauschen.

Der Aufgabenbereich einer Soundkarte erstreckt sich über die Aufzeichnung, Synthese, Mischung und Bearbeitung bis hin zur Wiedergabe von Audiosignalen.

Zurzeit gibt es eine große Auswahl von Soundkarten auf dem Markt. Vor allem die USB-Soundkarten sind weit verbreitet, weil sie eine einfache Lösung für die Nutzung eines externen Geräts bieten.

Auf dem Markt existiert bis jetzt keine Ethernet (Netzwerk) Soundkarte. Das Ziel dieses Projekts ist es, eine Netzwerksoundkarte für ein Linux Betriebssystem zu implementieren. Heutzutage ist Ethernet das am häufigsten verwendete Bussystem, um eine Kommunikation zwischen verschiedenen Geräten zu aufzubauen oder eine Verbindung mit dem Internet herzustellen. Die Grundidee ist, eine Soundkarte auf Grundlage der Ethernet-Technologie zu entwickeln, die das TCP/I-Protokoll verwendet und somit eine Client-Server Architektur realisiert.

Desweiteren verarbeitet die Soundkarte ausschließlich digitale Signale vom Rechner bis hin zum Lautsprecher. Dadurch ergeben sich im Wesentlichen zwei Vorteile. Zum einen werden in der digitalen Verarbeitung keine analogen Störsignale mitverstärkt. Zum anderen weist die Erzeugung von Leistungssignalen mittels Pulsweitenmodulation wesentlich höhere Wirkungsgrade auf als klassische Typ A oder Typ A/B Verstärker.

4 SPEZIFIKATION

4.1 DATENFLUSS

Das Ziel dieser Bachelorarbeit ist die Implementierung einer Soundkarte, die nur mit einer Netzwerkschnittstelle kommuniziert. Die Soundkarte erhält alle Daten für ein oder zwei Audiokanäle von einem gewöhnlichen Rechner.
Die Firmware der Soundkarte speichert alle Daten zwischen und überprüfen. Anschließend muss eine Verwaltung des Puffers alle Samples unter Verwendung eines PWM-Ausgangs zum Lautsprecher weiterleiten, wo sie dann abgespielt werden.

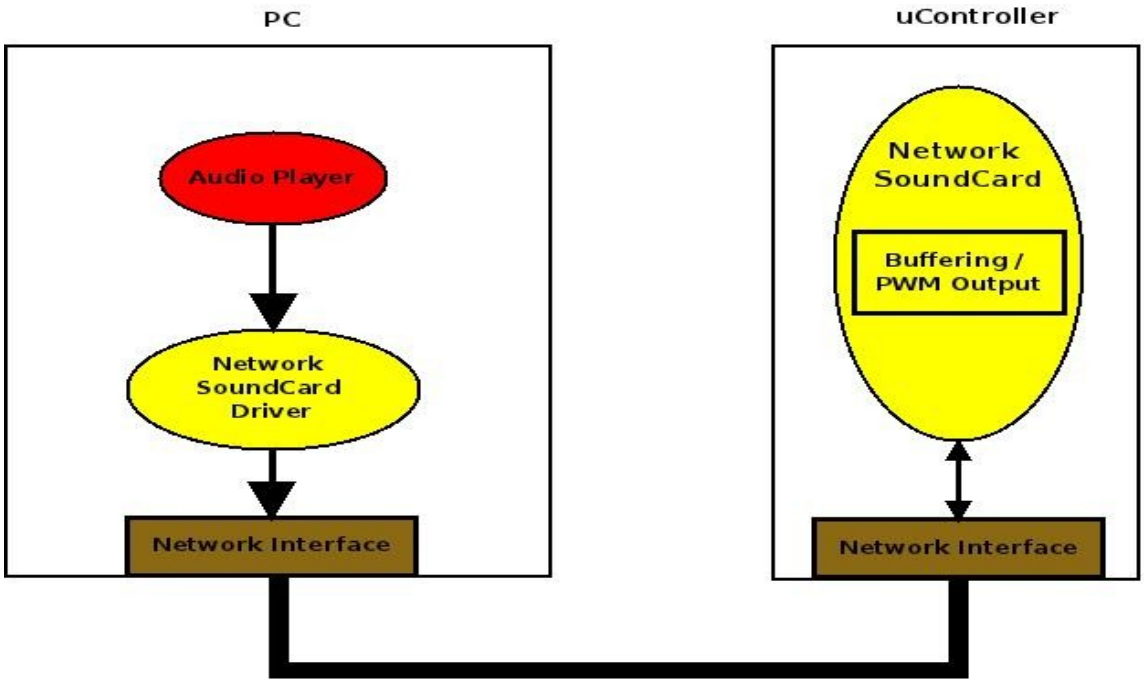


Abbildung 1: Datenfluss

Auf der Softwareseite gibt es drei Pakete die implementiert werden müssen, um dieses Projekt realisieren zu können:

- Erster Teil: Der Rechner läuft unter dem Betriebssystem Linux (openSuSE 11.4 x86). Es existieren viele Soundkartentreiber für verschiedene Karte, jedoch gibt es keinen Treiber für eine Soundkarte die mit Ethernet funktioniert. Man muss ein Kernelmodul entwickeln, welches eine Soundkarte implementiert und Audiodaten an eine voreingestellte IP-Adresse streamt.
- Zweiter Teil: Entwicklung einer Firmware für eine Mikrocontrollerplatine, welche eine Netzwerksoundkarte darstellt. Die Soundkarte bekommt mehrere Audiokanäle von der Netzwerkschnittstelle und macht eine Trennung (Demultiplexing) um alle die Kanäle einzeln zu verarbeiten.

- Die Musik muss jetzt gespielt werden. Es ist das Ziel, einen digitalen Ausgang zu benutzen. Der Mikrocontroller hat viele PWM-kompatible Ausgänge. Eine Testplatine steht zur Verfügung um die Firmware zu testen.
- Dritter Teil: Entwicklung einer eigenen Platine für eine autarke Ethernet Soundkarte. Die Platine soll mit einer einzelnen Spannung versorgt werden. Ein Ethernet Controller ermöglicht den bidirektionalen Datenaustausch mit dem Rechner. Drei DC-DC Converter stellen verschiedene Spannungsniveaus energieeffizient zur Verfügung. Ein Teil mit Leistungselektronik erlaubt die Erzeugung von leistungsstarken PWMs der verschiedenen Spannungsniveaus. Diese PWMs werden an zwei Lautsprecheranschlüssen bereitgestellt.

4.2 BLOCK SCHEMA

Nachfolgend ein Schema mit alle Komponenten des Projekts :

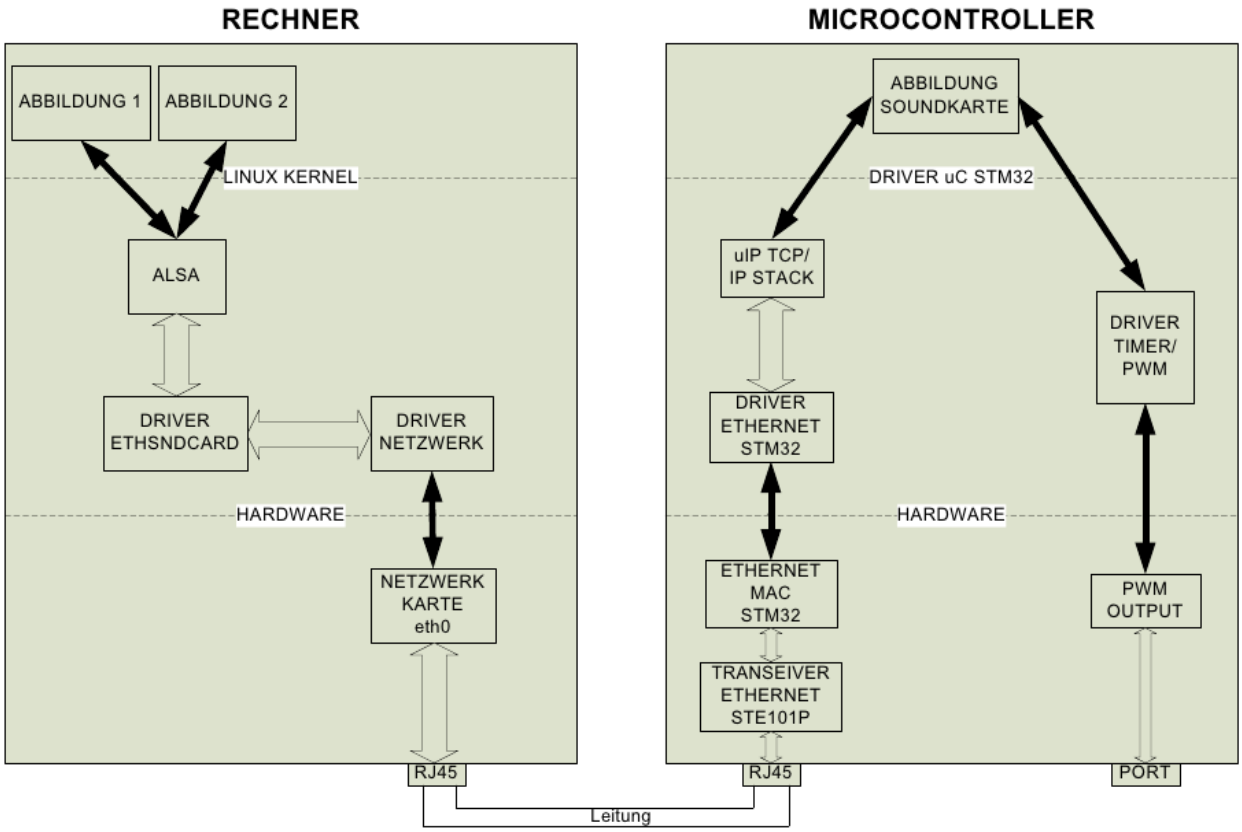


Abbildung 2: Block Schema

Auf einem Rechner gibt es viele Applikationen, die eine Audio-Schnittstelle verwenden, wie etwa die Wiedergabe von Audiosignalen. Auf einem Linux Betriebssystem wenden alle Applikationen eine standardisierte Architektur an. Diese Architektur heißt ALSA für Advanced Linux Sound Architektur. Alle Applikationen verwenden Funktionen von ALSA um Sounds wieder zu geben.

Danach kann ALSA den Sound zu dem richtigen Treiber durchschalten. ALSA erkennt, welche Soundkarte zurzeit benutzt wird und kann den passenden Treiber auswählen.

Es existieren keine Ethernet Treiber, welche mit ALSA funktionieren. Somit besteht der erste Teil dieses Projekts darin, einen ALSA Treiber zu entwickeln, welcher per TCP/IP kommunizieren kann.

Dieser Treiber wird einen anderen Treiber benutzten. Dies ist ein universeller Netzwerktreiber in der TCP/IP Schicht. So funktioniert diese Soundkarte mit einer beliebigen Netzwerkkarte.

Desweiteren benötigt das System ein externes Gerät mit einer Netzwerkschnittstelle. Der Treiber schickt die rohen Audiodaten an das externe Gerät, welches dann das Audiosignal über einen Lautsprecher ausgibt. Dieses externe Gerät ist die zu entwickelnde Ethernet Soundkarte

Auf der Platine der Soundkarten gibt es verschiedene Bauteile. Für die Ethernet Verbindung enthält die Platine einen Transceiver mit einem RJ45 Stecker. Im Mikrocontroller befindet sich eine Funktionseinheit namens MAC (Medium Access Control). Der MAC ist eine Schnittstelle zwischen Software und dem Transceiver. In unserem Fall ist die Hardware der Transceiver und die Software die Firmware Anwendung.

Viele Mikrocontroller haben ihren eigenen Treiber schon entwickelt. Also braucht dieser nicht mehr erstellt zu werden.

Die Soundkarten Anwendung muss die Kommunikation mit dem Rechner verwalten und den Puffer mit allen Samples kontrollieren. Das ist der zweite Teil dieses Projekts. Dieser Teil ist sehr wichtig, weil der Mikrocontroller sicherstellen muss das die Übertragungsrate eingehalten wird und alle Samples gespielt werden.

Die dritte Teilaufgabe ist das Ausgeben der digitalen Audiodaten auf einen an die Soundkarte angeschlossenen Lautsprecher. Die zu entwickelnde Audioendstufe soll klein sein und eine geringe Verlustleistung aufweisen. Die Lösung ist im Mikrocontroller für jeden Audiokanal ein PWM-Signal zu erzeugen. PWM bedeutet Pulse With Modulation bzw. Pulsweitenmodulation. Diese Technik ist sehr einfach. Mit einer festen Frequenz modifiziert man die Impulsbreite, welche ein Sample in ein oder zwei Frequenzperiode abbildet. Die Frequenze der PWM ist deutlich höher gewählt als die Zeitkonstante der Lautsprecherspule. Das analoge Audiosignal ergibt sich dann als Strom durch die Spule des angeschlossenen Lautsprechers.

Beim aktuell vervendeten Mikrocontroller werden für diese Aufgaben sämtliche vorhandenen Hardwaretimer benötigt.

Das PWM-Signal wird auf eine andere Platine via einer einfachen Leitungsverbindung geschickt. Auf dieser Platine gibt es einen Lautsprecher, welcher die Musik abspielt und somit als Testvorrichtung für die Soundkarte fungiert.

Ein letzter optionaler Teil wird sein, eine spezielle Hardwareplatine zu entwickeln. Diese Platine wird alle Bauelement für die Soundkarte enthalten. Zum Beispiel braucht die Soundkarte einen Ethernet Transceiver mit einem Mikrocontroller, eine Energie Verwaltung und einen Ausgang mit mehr Leistung für den Lautsprecher. Eine weitere Herausforderung ist das die Platine so klein wie möglich sein soll.

5 WERKZEUG

5.1 LINUX UND LINUX KERNEL

Linux oder GNU/Linux ist ein Opensource Betriebssystem. Es basiert auf dem Linux-Kernel und im Wesentlichen auf der GNU-Software. Die kommerzielle Verbreitung wurde ab 1992 durch die Lizenzierung des Linux-Kernels unter der GPL (General Public License) ermöglicht.

Die Besonderheit dieses Betriebssystems ist, das es von Softwareentwicklern auf der ganzen Welt weiterentwickelt wird, welche an den verschiedenen Projekten arbeiten. Es gibt verschiedene Linux-Distributionen, die den Linux-Kernel verwenden.

Einige Beispiele von Distributionen sind etwa: Ubuntu, Debian, Mandivia, OpenSuse usw.

Hinweis: Für dieses Projekt habe ich die Distribution OpenSuse angewendet.

Nachfolgend ein Schema der Linux-Struktur abgebildet:

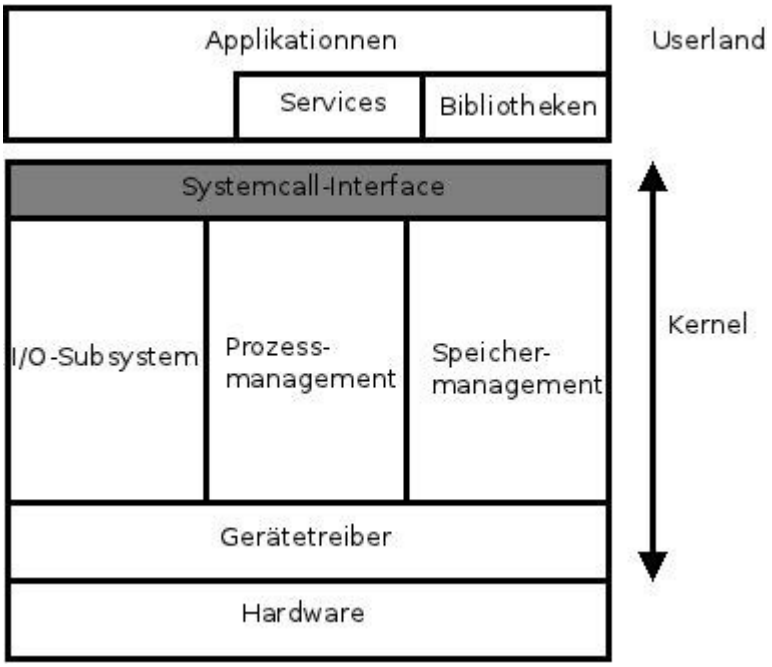


Abbildung 3: Linux Struktur (cf. Ref. 2)

- Die Benutzerschicht (userland): Alle Anwendungen laufen in dieser Schicht. Es gibt auch viele System-Dienste die hier laufen.
- Linux-Kernel: Dies ist der Kern des Betriebssystems oder Systemkern. Der Kernel ist der zentrale Bestandteil eines Betriebssystems. In ihm ist die Prozess- und Datenorganisation festgelegt, auf der alle weiteren Softwarebestandteile des Betriebssystems aufbauen. Er bildet die Schicht des Systems und hat direkten Zugriff auf die Hardware. Zum Beispiel enthält ein Kernel alle Treiber für Hardwaregeräte.
- Hardware: Zum Beispiel gibt es die CPU (Hauptprozessor), verschiedene Verarbeitungseinheiten wie etwa RAM, Datenträger oder Netzwerk (um auf das Internet zu vernetzen) und externe Peripheriegeräte wie z.B. Tastatur und Maus.

Komponenten des Kernels :

(cf. Ref. 2)

Systemcall-Interface :

Applikationen können die Dienste, die ein Betriebssystem zur Verfügung stellt über das Systemcall-Interface verwenden. Technisch ist diese Schnittstelle über eine Art Sofware-Interrupts realisiert. Möchte eine Applikation einen Dienst (zum Beispiel das Lesen von Daten aus einer Datei) nutzen, löst sie einen Software-Interrupt aus und übergibt dabei Parameter, die den vom Kernel auszuführenden Dienst hinreichend charakterisieren. Der Kernel selbst führt das Auslösen des Software-Interrupt und die zugehörige Interrupt Service Routine (ISR) aus und gibt der aufrufenden Applikation einen Rückgabewert zurück.

Das Auslösen des Software-Interrupts wird im Regelfall nicht durch die Applikationentwickler selbst programmiert. Vielmehr sind die Aufrufe der Systemcalls in den Standardbibliotheken versteckt, und eine Applikation nutzt eine dem Systemcall entsprechende Funktion in der Bibliothek.

Prozessmanagement

Eine zweite Komponente des Betriebssystemkerns stellt das Prozess-Subsystem dar. Im Wesentlichen verhilft es Einprozessorsystemen dazu, mehrere Applikationen quasi parallel auf dem einen Mikroprozessor (CPU) abarbeiten zu können. Bei einem Mehrprozessorsystem werden die Applikationen auf die unterschiedlichen Prozessoren verteilt. Aus Sicht des Betriebssystems werden Applikationen als Rechenprozesse (genauer Tasks oder Threads) bezeichnet.

Jeder Rechenprozess besteht aus Code und Daten. Dafür wird im Rechner jeweils ein eigener Speicherblock reserviert. Ein weiterer Speicherblock kommt hinzu, um die während der Abarbeitung des Prozesses abzulegenden Daten zu speichern, der sogenannte Stack. Damit belegt jeder Prozess mindestens drei Speicherblöcke: ein Codesegment, ein Datensegment und ein Stacksegment.

Speichermanagement

Die dritte Komponente moderner Betriebssysteme ist die Speicherverwaltung. Hardware und Software machen es möglich, daß, daß jede Anwendung in der Benutzerschicht scheinbar über einen linearen Adressbereich verfügen kann. Der Entwickler kann Speicherbereiche (Segmente) definieren, die er dann, durch die Hardware unterstützt, bezüglich lesender und schreibender Zugriffe überwachen kann. Darüber hinaus kann sichergestellt werden, daß aus einem Datensegment kein Code gelesen wird beziehungsweise in ein Codesegment keine Daten abgelegt werden.

IO-Management

Ein vierter großer Block des Betriebssystemkerns ist das IO-Management. Dieses ist für den Datenaustausch der Programme mit der Peripherie, den Geräten, zuständig.

Das IO-Management hat im Wesentlichen drei Aufgaben:

- Ein Interface zur systemkonformen Integration von Hardware anzubieten
- Eine einheitliche Programmierschnittstelle für den Zugriff auf die Peripherie zur Verfügung zu stellen
- Ordnungsstrukturen für Daten in Form von Verzeichnissen und Dateien über das sogenannte Filesystem zu realisieren

Geräte Treiber

Die fünfte Komponente eines Betriebssystems sind die Gerätetreiber. Als Softwarekomponente erfüllen sie eine überaus wichtige Funktion. Sie steuern den Zugriff auf alle Geräte. Erst der Treiber macht es einer Applikation möglich über ein bekanntes Interface die Funktionalität eines Gerätes zu nutzen.

Weitere Beschreibungen im Kapitel «Treiber Entwicklung»

5.2 LINUX ALSA

Die Advanced Linux Sound Architecture (ALSA) ist eine freie Soundarchitektur für Linux-Systeme, die PCM-Audio- und MIDI-Funktionalität anbietet. ALSA wie Linux stehen unter der GPL-Lizenzierung. ALSA enthält drei Komponenten: Treiber, Hilfsprogramme und Anwendungsbibliotheken zum Testen.

ALSA besteht aus Linux-Kernelmodulen, die verschiedene Kerneltreiber für Soundkarten bereitstellen. Unterschiedliche Aufgaben (allgemeiner Sound; Midi, Wave, Synthesizer; Hardware) werden durch einzelne Gerätetreiber im Soundstack abstrahiert. Wiedergabe von Dolby Digital ist möglich. Die Ziele des ALSA-Projektes waren insbesondere die Unterstützung einer automatischen Konfiguration der Soundkarten und eine elegante Handhabung mehrerer Soundgeräte in einem System. Diese Ziele wurden größtenteils erreicht. Verschiedene Frameworks wie JACK und PulseAudio nutzen ALSA für Audiobearbeitung und -abmischung auf professionellem Niveau mit niedriger Latenz.

Die wenig gepflegten Treiber für die OSS3-Architektur werden in aktuellen Kernel-Versionen zugunsten von ALSA als veraltet markiert.

ALSA hat viele Werkzeuge, die es ermöglichen verschiedene Aufgabe auszuführen:

- Alsactl: Alsa Control ist ein Programm, um die Soundkarten zu kontrollieren und die Konfiguration zu manipulieren. Alsactl kann auch viele Konfigurationen für verschiedene Soundkarten speichern oder laden.
- Alsaconf: Diese Firmware kann die Soundkarte, welche auf dem Rechner ist identifizieren und konfigurieren.
- Alsamixer: Diese Firmware kann die Lautstärke von verschiedene Kanälen abgleichen. Es verwendet Ncurses (Teilgrafisches Text-Userinterface) für die Anzeige.
- Aplay: Diese Software kann eine .wav Datei abspielen.
- Arecord: Diese Software kann eine .wav Datei ab einem Schallausgang speichern.
- Aconnect: Diese Software kann zwei bestehende Ports auf einen Sequenzgeber aufschalten.

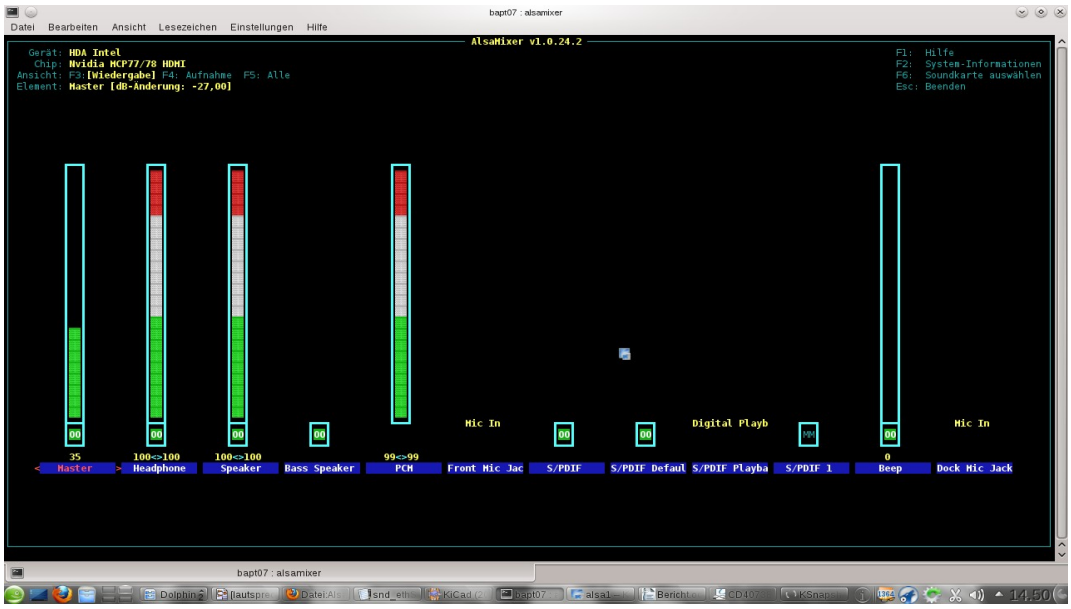


Abbildung 4: ALSA -> ALSAMIXER

Hier ein Beispiel einer Alsamixer Anzeige für eine HDA Intel Soundkarte . Die verschiedenen Lautstärken, die modifiziert werden können sind der Master, der Lautsprecher, das Mikrophon, der Systembeep und die PCM (Pulse Code Modulation) Lautstärke.

Nachfolgend die ALSA Architektur eines Rechners:

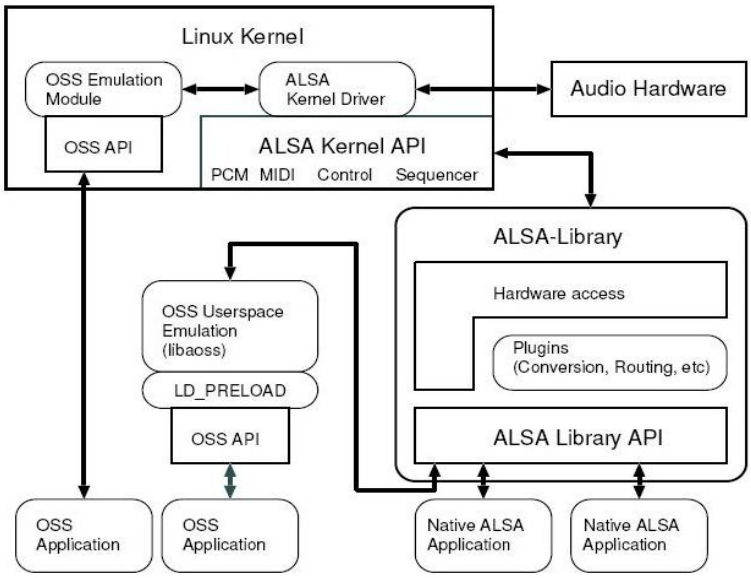


Figure 1: Basic Structure and Flow of ALSA System

Abbildung 5: ALSA System (cf. Ref. 3)

Im ALSA System gibt es zwei wichtige Teile, die ALSA-Library und ALSA-Kernel. In der ALSA-Library gibt es alle Werkzeuge, die der Benutzer direkt aufrufen kann. In ALSA-Kernel findet man sämtliche Treiber für alle von ALSA unterstützten Soundkarten. Wenn ein Soundkartenhersteller einen Treiber für seine Hardware entwickelt hat, kann er seinen Quelltext an die ALSA-Gemeindeschicken. Diese fügt dann den Treiber in zum ALSA-Tree hinzu.

ALSA ist in der Linux Kernel-Version 2.5 erstmals eingeführt worden. Früher war es ein anderes System. Dieses hieß OSS (Open Sound System). Für alle OSS-Anwendungen hat ALSA eine Kompatibilitätsschicht implementiert, die OSS Userspace Emulation.

5.3 SOCAT

Socat ist ein Werkzeug für TCP-IP Netzwerke. Dies ist ein Werkzeug für den bidirektionalen Datentransfer zwischen zwei Datenkanälen. Diese Kanäle können eine Datei, ein Pipe, ein Gerät (Z.B. seriell Linie), ein Socket (Z.B. IP4, IP6, UDP, TCP), ein SSL Socket, eine Proxy Verbindung, ...

Es gibt viele Anwendungen von Socat. Nachfolgend sind einige Beispiele mit Socket-Befehlszeilen aufgeführt.

```
# socat TCP4-LISTEN:6666 TCP4:www.company.com:80
```

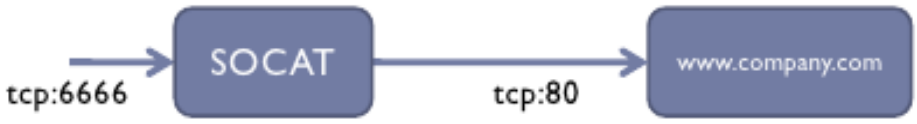


Abbildung 6: Socat Beispiel 1 (cf. Ref. 4)

Das erste Beispiel zeigt ein Relais zwischen zwei Datenkanälen. Die Befehlszeile ist leicht zu verstehen. Das Stichwort ist natürlich SOCAT mit zwei Parametern. Der erste Parameter ist die Quelle der Daten und der Zweite ist das Ziel.

```
# socat TCP4-LISTEN:6666 OPENSSL:192.168.123.50:443
```

Abbildung 7: Socat Beispiel 2 (cf. Ref. 4)

Hier ist die Quelle ein TCP-IP (IPv4) DatenStrom auf dem TCP-Port 6666 (Socat ist hier der Server). Das Ziel ist die IP-Adresse von www.compagny.com auf dem TCP-Port 80 (der gewöhnliche Port für eine Internet-Verbindung) und Socat ist hier der Client.

Socat kann auch verschlüsselte Verbindungen via SSL erstellen. Dazu wird als Protokoll openssl gewählt, sowie evtl ein anderer Port. Daten, welche über diese Verbdingung laufen, werden vor der Übertragung verschlüsselt. Am anderen Ende der Verbindung muss dann natürlich ein entsprechender Server auf verschlüsselte Daten warten (hier an Port 443.

Abbildung 8: Socat Beispiel 3 (cf. Ref. 4)

Allgemein wurde Socat verwendet um während der Entwicklung des ALSA Treibers einen Netzwerkport bereitzustellen, welcher die rohen Audiodaten entgegennimmt. Zu diesem Zeitpunkt existierte die eigentliche Soundkarte noch nicht. Zu Testzwecken wurde der Port auf demselben Rechner bereitgestellt, auf welchem auch der ALSA-Treiber geladen wurde. Die rohen Audiodaten wurden in eine Datei ausgegeben und konnten anschließend auf korrekte Übertragung analysiert werden.

6 TREIBER ENTWICKLUNG

6.1 ALLGEMEIN

6.1.1 Was ist ein Treiber?

Ein Treiber (Driver) ist eine kleines Programm, welches einem Programm (oft ein Dienst des Betriebssystems oder Eine Anwendung) ermöglicht, über eine einheitliche Schnittstelle mit dem Gerät zu Interagieren. Jedes Gerät verfügt über seinen eigenen Treiber.

Die Aufgabe eines Treibers ist es, eine Schnittstelle zwischen dem System und einem externen Gerät oder einer Erweiterungskarte herzustellen.

Zum Beispiel kann ein Drucker oder eine Netzwerkkarte nicht ohne Treiber funktionieren.

Gewisse Betriebssysteme wie Windows haben ihre eigenen Treiber, die für mehrere ähnliche Geräte funktionieren, jedoch nicht immer mit allen Funktionalitäten. Diese Treiber haben nicht alle Fähigkeit eines Herstellertreibers.

Ganz verschiedene Arten von Hardware werden über Gerätetreiber in ein Betriebssystem integriert: Drucker, Kameras, Tastaturen, Bildschirme, Netzwerkkarten, Scanner, um nur einige Beispiel aufzuführen.

Treiber von den Geräteherstellern entwickelt. Für ein Gerät kann es verschiedene Treiberversionen geben, um z.B. Fehler in der Firmware zu korrigieren. Ein anderer Grund ist, daß ein Gerät nicht unbedingt kompatibel mit einem anderen Gerät ist. Derartige Inkompatibilitäten können durch Treiberupdates nachträglich behoben werden.

6.1.2 **Treiber für Linux**

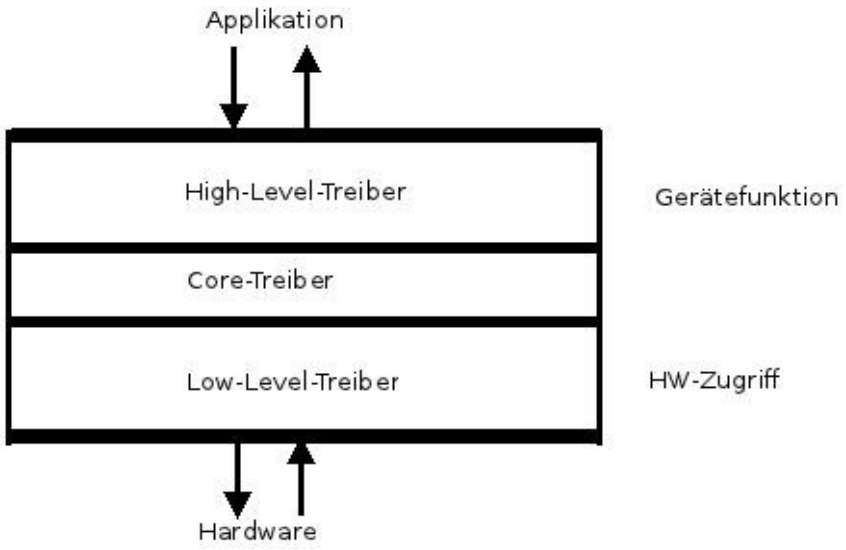


Abbildung 9: Treiber Linux (cf. Ref. 2)

Der Treiber muss die IO-Controls auswerten. Viele Treiber bestehen dabei aus mehreren Schichten (Low-Level, Core und High-Level) mit jeweils spezifischen Aufgaben. Man nennt sie deshalb auch geschichtete Treiber (stacked driver).

Der Low-Level-Treiber ist für die Ansteuerung der internen Hardwareschnittstelle, also beispielsweise eines ganz spezifischen USB-Controllers zuständig.

Da die Anzahl beziehungsweise Auswahl der USB-Komponenten für den direkten Hardwarezugriff gering ist, kommt man hier mit einer geringen Anzahl von Treibern aus. Der Low-Level-Treiber greift direkt auf die Register der Hardware zu.

Der High-Level-Treiber dagegen ist für einen Gerätetyp, zum Beispiel eine Webcam zuständig. Der notwendige Datentransfer zwischen dem Gerät (der Webcam) und dem Treiber wird durch den Low-Level-Treiber durchgeführt; der High-Level-Treiber greift also nicht direkt auf die Register der Hardware zu.

Bei USB werden beispielsweise zwischen Gerät und Treiber Kommandopakete verschickt. Damit ist der High-Level-Treiber für die Zusammenstellung der richtigen Pakete und die Auswertung der Antworten verantwortlich. Der eigentliche Pakettransport wird aber durch den Low-Level-Treiber initiiert.

Zwischen Low-Level-Treiber und High-Level-Treiber liegt die Core-Treiberschicht. Diese erweitert die interne Treiberschnittstelle um Geräte-typspezifische Funktionen. So stellt im Fall von USB der Core-Treiber Funktionen zum Geräte/Treiber-Management zur Verfügung.

In dieser Zwischenschicht ist beispielsweise abgelegt, welche Geräte am USB angeschlossen sind. Der Core-Treiber versucht zudem, den zu einem USB-Gerät passenden Treiber zu finden und zu laden, beziehungsweise wenn ein Treiber geladen wird, ein zugehöriges Gerät ausfindig zu machen und dem Treiber zuzuweisen.

6.1.3 Erstellung eines Kernelmoduls

Zum einfachen Start kann ein„Hallo“ Modul erstellt werden. Um so ein einfaches Modul zu erschaffen, muss man mindestens zwei Funktionen implementieren.

- Init: Eine Initialisierungsfunktion um das Module zu initialisieren. Diese Funktion benutzt einen spezifischen Kernel-Makro der « module_init() » heißt. Dieser Makro ermöglicht es, einen Link mit dem Kernel herzustellen.
- Exit: Eine Ausgangsfunktion um das Modul zu löschen. Diese Funktion benutzt ebenfalls einen Makro « module_exit() » um den Link zu löschen.

Für Kernelmodule werden verschiedene Programmiermakros bereitgestellt um häufig auftretende Konstrukte einfacher implementieren zu können. Die benennung des Modulauthors kann z.B. mittels MODULE_AUTHOR angegeben werden.

Nachfolgend das Beispiel „Hallo“ mit dem Resultat:

hello.c

```
1 #include <linux/init.h>
2 #include <linux/module.h>
3 #include <linux/kernel.h>
4
5 int test_init(void)
6 {
7   printk(KERN_INFO "Hallo !!\n");
8   return 0;
9 }
10
11 static void test_exit(void)
12 {
13   printk(KERN_INFO "Tchuss Bis bald!!\n");
14 }
15
16 module_init(test_init);
17 module_exit(test_exit);
18
19 MODULE_AUTHOR("Baptiste Solioz");
20 MODULE_DESCRIPTION("Test Module Hello");
21 MODULE_LICENSE("GPL");
22
```

C Tabulatorbreite: 4 Z. 1, Sp. 1 EINF

Abbildung 10: Module Beispiel "Hallo"

Die „printk()“ Funktion ist im Linux-Kernel implementiert und für Module erstellt worden. Das funktioniert wie bei der Funktion „printf()“ in C. Der Kernel braucht diese spezifische Funktion, weil er selbst ohne Hilfe der C-Bibliothek funktionieren muss. Das Modul kann printk() aufrufen, da es nach der Einfügung in den Kernel spezifische Kernel-Funktionen und Variablen nutzen kann.

Das Stichwort „KERN_INFO“ gibt der Nachricht Priorität. Nachfolgend gibt es alle Nachrichten-Typen mit einer Beschreibung:

```
#define KERN_EMERG      "<0>"   /* system is unusable                   */
#define KERN_ALERT      "<1>"   /* action must be taken immediately     */
#define KERN_CRIT       "<2>"   /* critical conditions                  */
#define KERN_ERR        "<3>"   /* error conditions                     */
#define KERN_WARNING    "<4>"   /* warning conditions                   */
#define KERN_NOTICE     "<5>"   /* normal but significant condition     */
#define KERN_INFO       "<6>"   /* informational                        */
#define KERN_DEBUG      "<7>"   /* debug-level messages                 */
```

Es gibt verschiedene Prioritäten, da nicht alle Nachrichten dargestellt sind. Zum Beispiel kann man eine „Warning“-Nachricht missachten. In diesem Beispiel braucht man nur die Informationsnachricht.

Wenn wir das Modul laden, sehen wir die Nachricht „Hallo!!“. Und wenn wir das Modul löschen, erhalten wir die Nachricht „Tschüss Bis Bald!!“.

Nachfolgend der Test auf der Konsole:

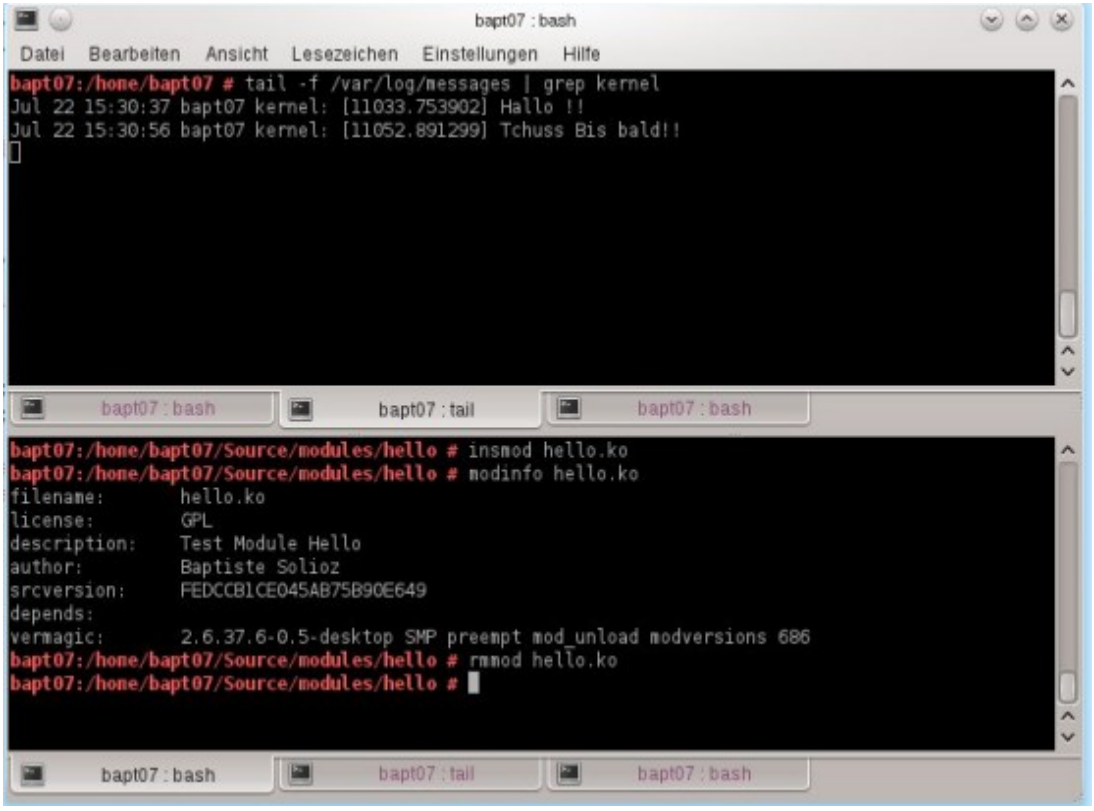


Abbildung 11: INSMOD / RMMOD

Oben kann man alle Kernel-Nachrichten sehen. Das Kommando, welches benutzt wird ist:

„tail -f /var/log/messages | grep kernel“.

Tail ist das Kommando um die letzten 10 Linien einer Datei darzustellen. Der Parameter "-f" sagt aus, daß endlos neu an die Datei angehängter Text ausgegeben wird. Da in die messages Datei sämtliche Systemmeldungen ausgegeben werden, wird die Ausgabe von tail per Pipe (| Symbol) an den Befehl grep geschickt. Grep unterdrückt hier sämtliche Zeilen, welche nicht das Schlüsselwort "kernel" enthalten.

Um ein Modul zu laden muss man das Kommando „insmod“ mit dem Modulnamen benutzen.

Dann gibt es das Kommando „modinfo“ mit welchem der Benutzer den Modulparameter sehen kann.

Und letztendlich muss man das Modul mit „rmmod“ löschen.

6.1.4 Wie kann man ein Modul kompilieren.

6.1.4.1 MakeFile

„make“ ist ein Computerprogramm, das Kommandos in Abhängigkeit von Bedingungen ausführt. Es wird hauptsächlich bei der Softwareentwicklung als Programmierwerkzeug eingesetzt.

Beispielsweise in großen Projekten, die viele verschiedenen Dateien mit Quelltextenthalten, kann man alle Arbeitsschritte (Übersetzung, Linken, Dateien kopieren, …) automatisieren.

„Make“ ist so mächtig, daß es nicht nur zur automatisierten Steuerung der Übersetzung geeignet ist, sondern beliebige Aufgaben übernehmen kann, bei denen die zeitliche Abhängigkeit von Dateien eine Rolle spielt.

Nachfolgend gibt es verschiedene andere Gründe „make“ zu benutzen:

- Ein Projekten mit vielen Dateien mit Quelltextkann viele Compiler-Kommandos die sehr komplex und lang sind enthalten. Mit Make kann man diese Kommandos reduzieren
- Make reduziert auch die spezifischen Optionen zum Kompilieren
- Die Notwendigkeit eine saubere Umgebung zu erhalten
- Man kann Make sehr leicht mit einer Konsole anrufen

Die Anweisungen sind in eine Textdatei geschrieben. Gewöhnlich werden diese Dateien Makefile genannt und enthalten das Kommando, daß Make bearbeiten soll.

Die Struktur einer Makefile ist nachfolgend zu sehen.Wesentlich sind die Angaben für das zu erstellende Ziel (target), die dafür aufzulösenden Abhängigkeiten (dependencies) sowie die auszuführenden Befehle (command). Die Abhängigkeiten sind alle anderen Ziele, welche vorher erstellt werden müssen. Nachdem die Abhängigkeiten erstellt wurden, werden die aufgelisteten Befehler nacheinander ausgeführt. Es ist also wichtig festzustellen, daß ein Makefile nicht einfach von oben nach unten ausgeführt wird. Stattdessen werden zunächst die Ziele erstellt, welche keine unaufgelösten Abhängigkeiten enthalten. Zum Schluss wird dann das Ziel erstellt, von dem kein weiteres Ziel mehr abhängt.

Grundstruktur eines Makefiles:

```
target: dependencies
        command
        command
        ...
```

6.1.4.2 Module MakeFile

Um ein Modul zu erschaffen und zu kompilieren muss man eine kleine Makefile schreiben.

Diese Makefile ist nachfolgend beschrieben:

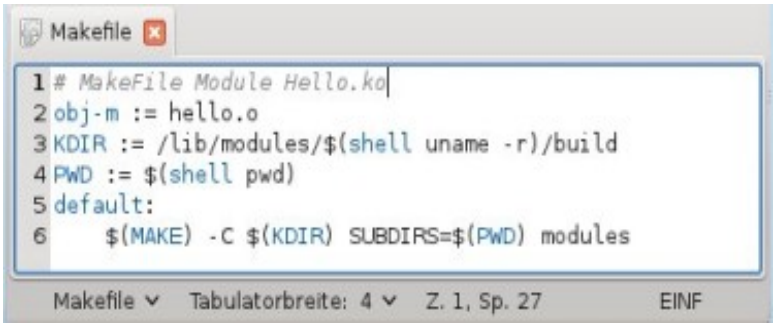


Abbildung 12: Kode Makefile

- **obj-m := hello.o** : Die hello.o ist eine Objektdatei, die zu hello.c gehört. Sie wird automatisch kompiliert. Dann kommt die obj-m. Sie bedeutet Objekt (obj) und Module oder Treiber (m). Dies ist eine Liste welche der Kernel bauen muss. Diese Zeile erschafft eine Datei hello.ko, die das Modul darstellt.
- **KDIR := /lib/modules/$(shell uname -r)/build** : Diese Zeile erschafft eine Variable KDIR (Kernel-Directories). Die Variable enthält den absoluten Dateipfad zu den Linux Kernelquellen auf dem aktuellen Rechner. Der Text "$(shell uname -r)" wird dabei durch das Resultat des Shellbefehls "uname -r", dem aktuellen Kernel-Release String ersetzt. Das Resultat kann beispielsweise « 2.6.37.6-0.5-desktop » sein.
- **PWD := $(shell pwd)** : Die Variable PWD wird auf das aktuelle Verzeichnis gesetzt in welchem der Quelltext des zu übersetzenden Kernelmoduls liegt. In diesem Fall heißt das Verzeichnis beispielsweise «/home/bapt07/Source/modules/hello»
- In Zeile 5 wird das Ziel default festgelegt, welches im make Prozess immer dann verwendet wird, wenn kein spezielles Ziel angegeben wird.
- In Zeile 6 wird der Shellbefehl make mit entsprechenden Parametern aufgerufen um das Kernel Buildsystem anzustossen und unser Kernelmodul zu übersetzen.

Nachfolgend die Benutzung des Makefiles:

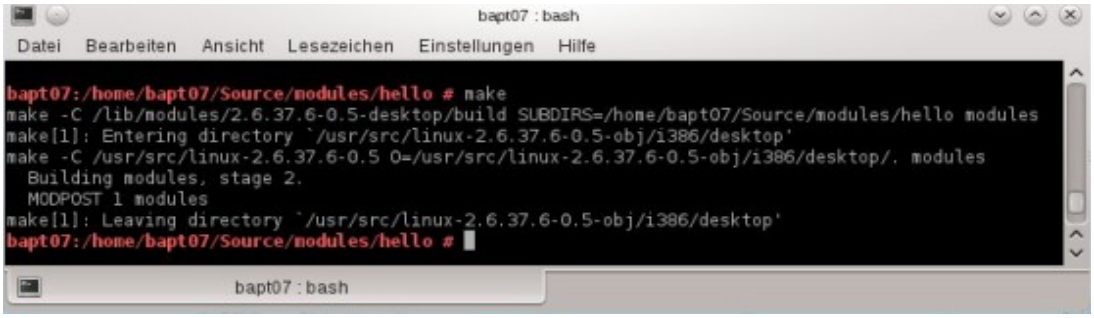


Abbildung 13: Benutzung des Makefiles

Zuerst muss man in dem Makefile das entsprechende QuelltextVerzeichnis angeben (zum Beispiel mit cd /home/bapt07/Source/modules/hello). Danach muss man nur das Kommando „make“ benutzen.

6.2 ALSA Driver Architektur

Hier ist der Aufbau des ALSA Treibers abgebildet, wenn das Modul, welches kompiliert ist, eingefügt ist (INSMOD):

Der Erste Teil ist der Anruf der Module Init() Funktion. Dann ruft ALSA den allgemeinen Konstruktor der Karte auf. Und am Ende wird der spezifische und PCM Konstruktor in der gleichen Funktion aufgerufen. Nach diesen Anrufen ist der Treiber bereit um Samples zu verarbeiten.

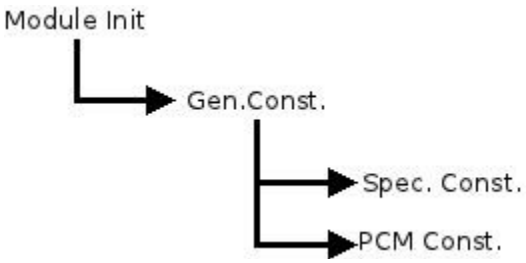


Abbildung 14: ALSA Init

Die Beschreibung der Löschfunktion des Moduls (RMMOD) gleicht der vorhergehenden Beschreibung. Aber jetzt wird der Exit Module Funktion und nicht die Konstruktoren aber die Destruktoren aufgerufen .

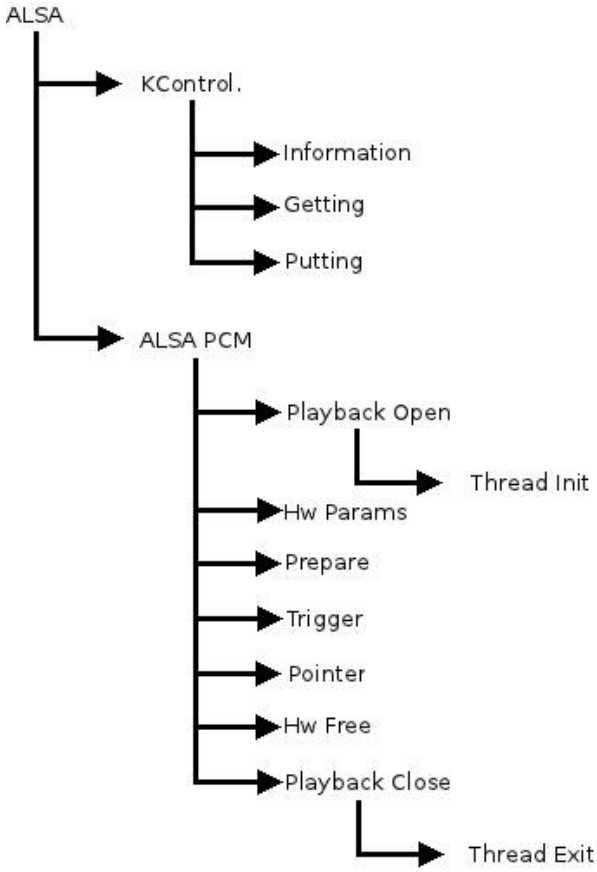


Abbildung 15: ALSA Funktion Struktur

Wenn Musik gespielt wird, kann ALSA verschiedene Funktion ausführen:

- Kcontrol Funktion: um die Lautstärke zu modifizieren
- ALSA PCM Funktion: um das Audiosignal abzuspielen. Die Vorbereitung für die Soundkarte und die Verwaltung des Samples während des Sounds wird gespielt.

6.3 IMPLEMENTIERUNG

6.3.1 Verwaltung der Karte und die Komponenten

- Driver Description: Die Treiber Beschreibung ist eine Struktur mit verschiedenen Informationen über den Treiber. Angegeben werden z.B. de Adressen der Funktionen zur Initialisierung und zur Beendigung des Moduls. Damit das Modul vom System richtig verwendet werden kann, muss ein sogenannter Strukturtyp angegeben werden. Der hier verwendete Strukturtyp heisst "platform_driver". Beispielsweise gibt es für PCI eine spezielle Struktur „pci_driver“ und für USB „usb_driver“.
- Module Init(): Das ist die Funktion, die von INSMOD aufgerufen wird. Es gibt nur ein wichtiges Ziel hierbei, die Funktion muss den Platform Treiber anmelden.
- Module CleanUp: Das ist die Funktion, die von RMMOD aufgerufen wird. Im Gegenzug zum Module Init muss diese Funktion den Platform Treiber abmelden.
- Card Constructor:
  - General: Jeder Soundkarte muss ein Datenregister zugewiesen werden. Ein Datenregister bildet die Basis für eine Soundkarte. Diese verwaltet die vielen Komponenten die es auf der Karte gibt. Beispielsweise PCM, Mixers (KControl), MIDI, Synthesizer und so weiter. Diese Datenregister enthalten die ID Nummer und Textdaten mit dem Soundkartennamen. Um eine Karte zu erschaffen muss man nur die Funktion „snd_card_create()“ anrufen. Diese benötigt fünf Parameter: die Karten-Nummer, die Textdaten, den Modul Zeiger, einen Extra-Speicher und den Zeiger auf der Karteninstanz. Diese Funktion muss auch andere Dinge ausführen. Sie muss den spezifischen Konstruktor anrufen und die PCM- und Kontroll-Komponente erschaffen.
  - Specific: Der erste Konstruktor ist für alle Soundkarten kompatibel. Jedoch muss man einen spezifischen struct für die zu entwickelnde Karten erschaffen. Alle Karten haben eine Struktur mit spezifischen Informationen (Variablen und Zeiger). Die Struktur der Soundkarte mit Ethernet:

```
struct ethSndCard
{
    struct  snd_card                *card;
    struct  snd_pcm                 *pcm;
    struct  snd_pcm_substream       *substream;
    struct  socket                  *ethSock;
    struct  msghdr                  msg;
    struct  sockaddr_in             addr;
    wait_queue_head_t               wq;
    char                            kernelStop;
    char                            connected;
    int                             port;
    char                            *ipAddr;
    char                            start;
    char                            thread_id;
    char                            *actualPointer;
    int                             Volume;
};
```

Es gibt viele wichtige Elemente. Die ersten drei Parameter sind spezifisch für ALSA Treiber. Das ist die Zeigern über die Karte, der PCM Instanz, … Es gibt viele Elemente für Ethernet, wie den Socket-Zeiger oder den Port und die IP-Adresse. Es gibt auch Informationen über der Thread um die Samples zu schicken. Am Ende gibt es eine Speicher-Zeiger, eine Lautstärke-Information, ….

Der Konstruktor gibt Speicher für diese Struktur, initialisiert alle Wert und registriert das neue Gerät.

- Card Destructor:
  - General: Es gibt zwei Teile in diesem Destruktor. Erste ist der Karte abgewiesen. Der Funktion, die benutzt ist, ist „snd_card_free()“ mit dem Kartenzeiger. Der zweite Teil ist der Link zwischen dem Treiber und der Kartenzeiger zu löschen.
  - Specific: Nur benutzt um der Gerät zu löschen

6.3.2 Verwaltung der Puffer und Speicher

ALSA hat verschiedene Funktionen um einen Puffer zu erstellen. Dies kommt auf den Bustyp und die Architektur der Karte an.

Oft probieren ALSA Treiber physikalischen Speicher für die spätere Verwendung zu reservieren. Das heißt „pre-allocation“.

Es gibt eine spezifische Funktion für diesen Fall:

snd_pcm_lib_preallocate_pages_for_all()

Diese Funktion kann der PCM Konstruktor aufrufen.

Es gibt fünf Parameter mit dieser Funktion:

- Der Zeiger auf der PCM Instance
- Der Puffertyp
- Der Gerätezeiger
- Die Größe
- Die maximale Größe

Der erste und zweite Parameter sind vom verwendeten Bustyp abhängig.

Nachfolgend die Speicherstruktur des Treibers :

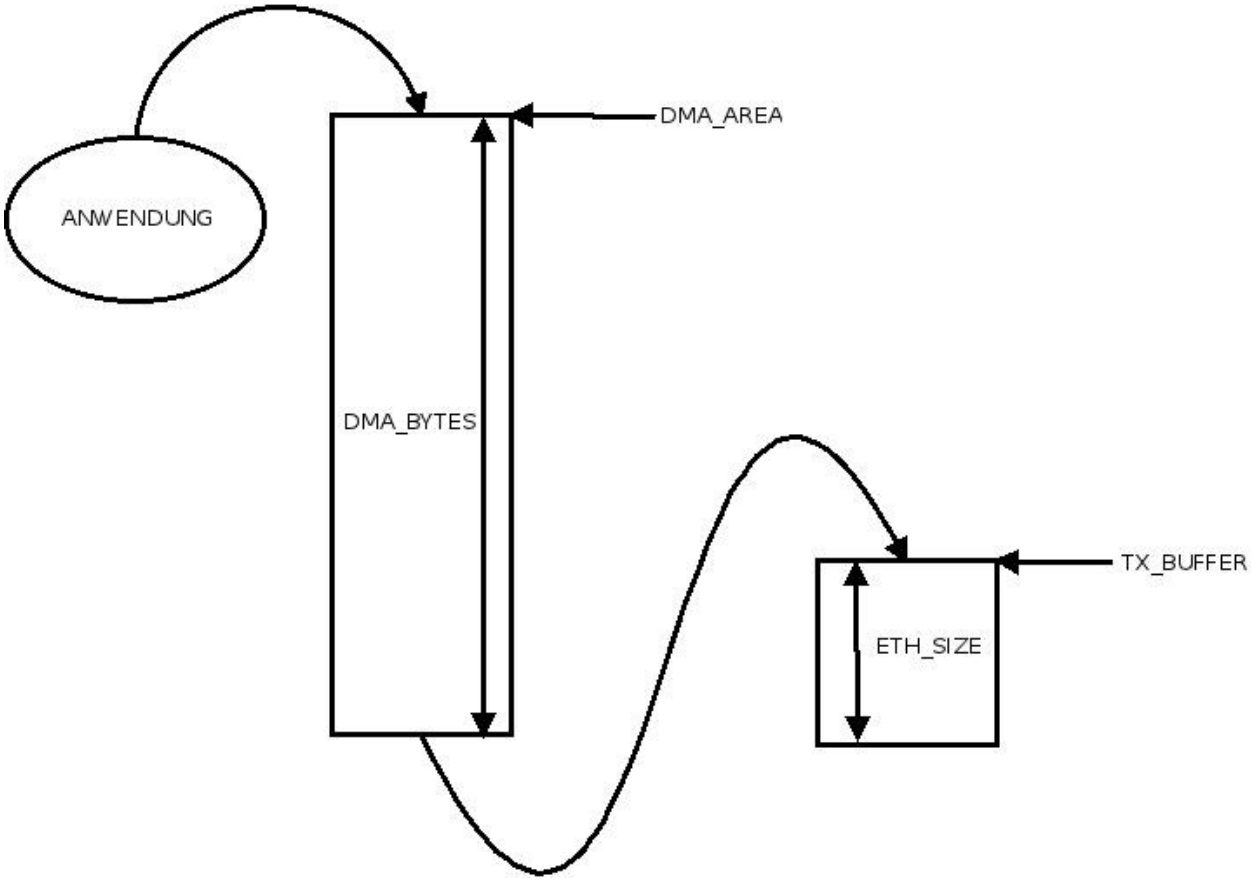


Abbildung 16: Speicherstruktur des Treibers

Wenn der Puffer zugewiesen wurde, kann die Anwendung die Samples spielen. Alle Samples werden im Puffer zwischengespeichert. DMA_AREA ist der Zeiger des Puffers und DMA_BYTES ist die Größe des Puffers.

Wenn der Treiber bereit ist, kann er die Daten in einen weiteren Puffer kopieren um diesen dann über das Ethernet weiterzuleiten. Die Größe des Puffers ist von der eingesetzten Ethernet Technologie abhängig. Beispielsweise hat dieser Treiber eine Puffergröße von 733 Bytes (ETH_SIZE) für die Nuztdaten.

Der Wert für DMA_BYTES wird im Card Constructor bei der Anmeldung der Soundkarte angegeben.

Der DMA-Puffer wird in mehrere Blöcke unterteilt. Die Größe eines Blocks wird durch die ETH_SIZE beschrieben.

Jedes Mal, wird ein Block in den TX_BUFFER zum schicken kopiert. Diese Funktion kann man „snd_pcm_period_elapsed()“ aufrufen.

Jetzt kann „ALSA“ den frei gewordenen Puffer neu verwenden und weitere Samples kopieren.

6.3.3 PCM Interface

Die PCM Schnittstelle ist eine Mittelschicht von ALSA (Ref 5.1.2 Treiber-Linux : Core-Treiber).

Alle Soundkartentreiber können vier PCM-Instanzen haben. Eine PCM-Instanz besteht aus einem PCM Playback (z.B. Lautsprecher) oder Capture (z.B. Mikrophone) Strom (Stream). Jeder PCM Strom hat Substreams um mehr als eine Playback oder Capture Funktion zu verwenden.

Es genügt für jedes PCM nur einen Substream zu verwenden.

Nachfolgend alle Funktionen und Strukturen, die benutzt werden:

- Playback Operators: Ein Operator ist eine Struktur, die eine Playback PCM Instanz nachweist. Diese Struktur hat acht Elemente. Ein Element ist ein Zeiger für eine Funktion, beispielsweise:

```
static struct snd_pcm_ops snd_ethSndCard_playback_ops =
{
    .open =         snd_ethSndCard_playback_open,
    .close =        snd_ethSndCard_playback_close,
    .ioctl =        snd_pcm_lib_ioctl,
    .hw_params =    snd_ethSndCard_pcm_hw_params,
    .hw_free =      snd_ethSndCard_pcm_hw_free,
    .prepare =      snd_ethSndCard_pcm_prepare,
    .trigger =      snd_ethSndCard_pcm_trigger,
    .pointer =      snd_ethSndCard_pcm_pointer,
};
```

  Die Beschreibungen des Operators werden unten weiter ausgeführt.

- Hardware Definition: Die Hardware Beschreibung (Struktur: snd_pcm_hardware) enthält die Definition der grundsätzlichen Hardware-Konfiguration. Die Definition dieser Struktur muss in der Funktion „snd_playback_open“ gemacht werden.

```
static struct snd_pcm_hardware snd_ethSndCard_playback_hw =
{
    .info = (       SNDRV_PCM_INFO_INTERLEAVED |
                    SNDRV_PCM_INFO_BLOCK_TRANSFER),
    .formats =      SNDRV_PCM_FMTBIT_S16_LE |
                    SNDRV_PCM_FMTBIT_S8 |
                    SNDRV_PCM_FMTBIT_S24_LE |
                    SNDRV_PCM_FMTBIT_S32_LE,
    .rates =        SNDRV_PCM_RATE_8000_96000,
    .rate_min =     8000,
    .rate_max =     96000,
    .channels_min = 1,
    .channels_max = 2,
    .buffer_bytes_max = 32768,
    .period_bytes_min = 728,
    .period_bytes_max = 32768,
    .periods_min = 1,
    .periods_max = 45,
};
```

  - Info: Dieses Feld enthält den Typ und die Eigenschaften des PCM-Geräts. Alle SchalterEinstellungen sind in der Datei <sound/asound.h> definiert (wie auch alle SchalterEinstellungen für Hardware Beschreibung). Das Bit INTERLEAVED sagt aus, daß die Hardware das „Interleaved“ Format unterstützt. Das Bit

BLOCK_TRANSFER besagt, daß die Hardware die Samples en Block bekommt. In unserem Fall lesen wir einen Block und schicken diesen mit TCP/IP dem Hardware-Gerät.

- Formats: Dieses Feld enthält die Bit-SchalterEinstellungen aller unterstützten Formate. In unserem Beispiel bieten wir verschiedene Formate an (S16_LE = Signed 16 bit Little Endian, S8 = Signed 8 bit, …).
- Rates: Dieses Feld enthält die Bit-SchalterEinstellungen der Abspielrate. Es kann mehrere Rate haben, wie in unserem Beispiel, indem die Hardware Abspielraten zwischen 8kHz und 96kHz unterstützt.
- Rate_min, Rate_max: Definiert die maximale Rate und die minimale Rate.
- Channels_min, Channels_max: Definiert mit wie vielen Kanäle die Karte arbeiten kann.
- Buffer_bytes_max: Definiert die maximale Puffer Größe in Bytes. Der Buffer_bytes_min ist automatisch mit die minimale Periode-Größe und der Anzahl der Periode gerechnet.
- Period_bytes_min: Dieses Feld ist die Größe einer Periode. Der Wert ist der ETH_SIZE (wie viele Bytes man mit TCP/IP schicken kann).
- Period_bytes_max: Dieses Feld ist die maximale Größe einer Periode. Das ist die Größe des Puffers.
- periods_min: aufgerufen Dieser Wert gibt den minimal erlaubten, zeitlichen Abstand zwischen zwei Datenblöcken an. Schneller kann die Hardware die Daten nicht verarbeiten.
- periods_max: Der maximale zeitliche Abstand zwischen zwei Datenblöcken. Falls länger Pakete ausbleiben, ist ein Stocken der Audioausgabe der Fall.

- PCM Constructor: Der PCM Konstruktor hat in diesem Projekt drei Aufgaben. Zunächst einmal die PCM Instanz anzumelden. Man benutzt die Funktion „snd_pcm_new() mit fünf Argumenten, den Kartenzeiger, ID Textdaten, eine Indexnummer (erste Nummer ist 0) und das vierte und fünfte Argument sind die Anzahl der „Playback Substream“ bzw. der „Capture Substream“. Das zweite Ziel ist die PCM Flüsse zu geben mit dem Funktion „snd_pcm_set_ops()“ mit verschiedene Parameters wie der PCM Instanz, der PCM Stromtyp (PLAYBACK oder CAPTURE) und der Operators-Zeiger. Das dritte Ziel ist die Puffervorzuweisung mit „snd_pcm_lib_preallocate_pages_for_all()“.

- Runtime Pointer: Wenn ein PCM Substream geöffnet wird, wird eine PCM-Runtime Instanz zugewiesen und mit dem Substream nachgewiesen. Der Zeiger ist erreichbar via „substream->runtime“. Dieser Runtime-Zeiger enthält viele Informationen über die PCM Kontrolle. Es gibt eine Kopie von Hw_params und sw_params, der Pufferzeiger, …

- Playback open: Diese Funktion wird aufgerufen, wenn eine Anwendung einen PCM Strom (Playback oder Capture) öffnet.
  - Socket Create: Um das Socket zu erschaffen muss man die Funktion „sock_create“ benutzen. Es gibt vier Argumente: die Socket-Familie (AF_INET für Adresse Familie mit Internet IP Protokoll), der Socket-Typ (SOCK_STREAM um einen Datenstrom zu haben), das Protokoll wie IPPROTO_TCP (Protokoll TCP/IP) und der Socket-Zeiger.
  - Connect Socket: Um ein Socket zu verbinden muss man die Funktion „kernel_connect()“ verwenden, da man in der Kernel-Space bleiben will (alle Zuweisungen werden zum Beispiel in der Kernel-Space gemacht). Es gibt vier

Parameter: den Socket-Zeiger, die Adressstruktur (enthält die Adress-Familie, AF_INET, der IP-Adresse und die Port-Nummer), die Adressgröße und einen Schalter (immer 0 in unserem Fall).

- Create Thread: Um ein Thread zu erschaffen muss man die Funktion „kernel_thread()“ benutzen. Es gibt drei Parameter:
  - Function_ptr: Die Adresse der Funktion, die bei erstmaliger Aktivierung durch den Scheduler aufgerufen werden soll
  - Arg: Ein Argument für die aufzurufende Funktion. Beispielsweise der Substream-Zeiger
  - flags: Ein Bitfeld, welches die Erzeugung des neuen Rechenprozesses steuert

- Playback close: Die Funktion wird aufgerufen, wenn ein PCM-Substream geschlossen wird. Zunächst muss man den Kernel beenden und löschen. Die Funktion ist „kill_pid()“mit Informationen über den Thread. Mit der Funktion „wait_for_completion()“ kann man auf den Abschluss warten, bevor der Thread gelöscht wird.

- Hardware Parameters: Diese Funktion wird aufgerufen, wenn der Hardware-Parameter von der Anwendung aktualisiert wird. In dieser Funktion wird auch der Pufferspeicher belegt. Es wird „snd_pcm_lib_malloc_pages()“ mit den Parametern Substream und der Puffergröße aufgerufen.

- Hardware Free: Diese Funktion wird aufgerufen, um die Ressourcen zu löschen. Um den Pufferspeicher zu löschen, wird „snd_pcm_lib_free_pages“ mit dem Substream für Parameter aufgerufen.

- PCM Prepare: Diese Funktion wird aufgerufen, wenn der PCM vorbereitet wird. Das Ziel ist die Hardware vorzubereiten. In unserem Fall muss man alle wichtigen Informationen an die Hardware-Karte schicken. Die Anwendung hat die Rate, das Format und die Kanalanzahl gegeben. Also muss man nun eine Nachricht mit diesen Informationen erstellen und schicken. Zum Schicken benutzt man die Funktion „kernel_sendmsg()“, um im Kernel-Space zu bleiben. Es gibt zwei spezifische Strukturen um eine Nachricht zu verschicken. Die erste ist ein Message-Header „msghdr“, der die Adresse, die Adressgröße und einen Schalter (immer 0) enthält. Die zweite ist eine Vektor-Struktur „kvec“ mit zwei Feldern (der Nachrichten-Zeiger und der Nachrichten-Größe.

- PCM Trigger: Diese Funktion wird aufgerufen, wenn der PCM gestartet oder gestoppt wird. Ein Kommando wird mit dem Funktionsaufruf angegeben, ob dies ein START oder ein STOP ist. In unserem Fall ist es ein START, um das Abschicken zu beginnen (man sagt dem Thread, daß er anfangen kann). Für einen STOP sagt man, daß der Thread das Abschicken stoppen kann und schickt an die Karte eine Nachricht, um das Ende anzumelden.

- PCM Pointer: Diese Funktion wird aufgerufen, wenn die PCM Mittelschicht die Hardware-Puffer Position wissen will. „PCM Pointer“ wird auch aufgerufen, wenn snd_pcm_period_elapsed() von der Thread aufgerufen wird. Die Mittelschicht updatet und kalkuliert den freien Speicher.

6.3.4 Kontroll Interface

Die Kontrollschnittstelle wird für alle Cursor und Switch (Mixer-Anwendungen und Lautstärke-Controller, cf. 5.2 Linux ALSA : ALSAMixer) benutzt, die seit der User-Space erreicht worden sind. Die wichtigste Benutzung ist die Mixer-Schnittstelle.

- Control Structure: Die Kontrolldefinition ist eine Struktur „snd_kcontrol_new“ mit verschiedenen Informationen :

```
static struct snd_kcontrol_new ethSndCard_ctrl __devinitdata =
{
        .iface =        SNDRV_CTL_ELEM_IFACE_PCM,
        .name =         "PCM Playback Volume",
        .index =        0,
        .access =       SNDRV_CTL_ELEM_ACCESS_READWRITE,
        .private_value = 0xffff,
        .info =         ethSndCard_ctrl_info,
        .get =          ethSndCard_ctrl_get,
        .put =          ethSndCard_ctrl_put
};
```

  - Das Feld „iface“ definiert den Kontrolltyp. In unserem Fall wird die Kontrolle mit PCM verbunden. Das Feld ".name" wird dem Benutzer als Text im Mischerprogramm (z.B. alsamixer), unter dem jeweiligen Lautstärkeregler, angezeigt. EIne genauere Beschreibung folgt unter „Control Names“.
  - Das Feld „index“ ist die Kontrollnummer
  - Das Feld „access“ sagt aus, ob die Anwendung lesen, schreiben oder beides machen kann
  - Das Feld „private_value“ enthält eine arbiträre Nummer für diese Kontrollstruktur
  - Die drei letzten Felder beinhalten die „Callback“ Funktion, welche die Anwendung via ALSA anrufen kann

- Control Names: Das Feld ".name" kann bestimmte Schlüsselwörter enthalten. Die ersten Namen geben die Quelle der Kontrolle an und sind Name wie „Master“, „PCM“, „CD“ und „Line“. Der zweite Name ist die Richtung der Kontrolle wie „Playback“, „Capture“, „Bypass Playback“, „Bypass Capture“. Der dritte Name ist die Funktion wie „Switch“, „Volume“ oder „Route“.

- Control Information: Die Informationsfunktion wird benutzt, um mehr Information über die Kontrolle zu erhalten. Eine Struktur „snd_ctl_elem_info wird dazu verwendet. Sie hat verschiedene Felder: „type“ ist der Kontrolltyp, wie BOOLEAN, INTEGER, ENUMERATED, BYTES, … „count“ gibt die Elementanzahl an. Für eine Stereo Ausgabe muss „count“ den Wert '2' haben. „value“ gibt den minimalen und maximalen erlaubten Wert an.

- Control Getting: Wenn diese Funktion aufgerufen wird, will die Anwendung den Lautstärke-Wert wissen. Diese Funktion muss diesen Wert wiedergeben.

- Control Putting: Wenn diese Funktion aufgerufen wird, will die Anwendung den Lautstärke-Wert modifizieren. Also muss man diesen Wert an die Hardware schicken.

6.3.5 Thread

Der Treiber arbeitet nicht direkt mit der Hardware. So braucht man eine andere Schicht, welche die Hardware abbildet. Wenn ein PCM Substream geöffnet wird, wird der Thread erschaffen.

Es ergeben sich verschiedene Schritte:

- Wenn der Thread erschaffen wird, macht man die Initialisierung
- Dann wartet der Thread darauf, dass ALSA die Funktion snd_pcm_trigger() aufruft.
- Jetzt muss man dem Speicher den txBuffer zum Schicken zuweisen
- Danach muss man die Nachricht vorbereiten. Es gibt einen Header Teil und einen Daten Teil. Detailliertere Erklärung über den Header in Punkt « 6 Verbindung TCP/IP »
- Die Nachricht kann jetzt verschickt werden. Man wartet auf eine Antwort, da die Soundkarte sagen muss, daß alles angekommen ist
- Die Antwort enthält auch eine andere Information um eine Regelung des Abschickens zu machen. So muss der Thread diese Antwort verarbeiten
- Ist die Nachricht verschickt, kann der Thread den Speicher des TxBuffer wieder freigeben.
- Wie in Punkt « 5.3.2 Verwaltung der Puffer und Speicher » beschrieben ist, muss der Thread sagen, daß der Speicher jetzt frei ist. Das geschieht durch Aufruf der Funktion « snd_pcm_period_elapsed ».
- Jetzt muss man bis zum nächsten Abschicken warten. Deshalb macht man ein Delay in Mikrosekunden. Bei diesem Delay kommt es auf drei Parameter an: die Nachrichten-Größe, die Rate und die Anzahl der Kanäle
- Der letzte Test zeigt, ob der Thread fertig ist. In diesem Fall kann man den Thread beenden.

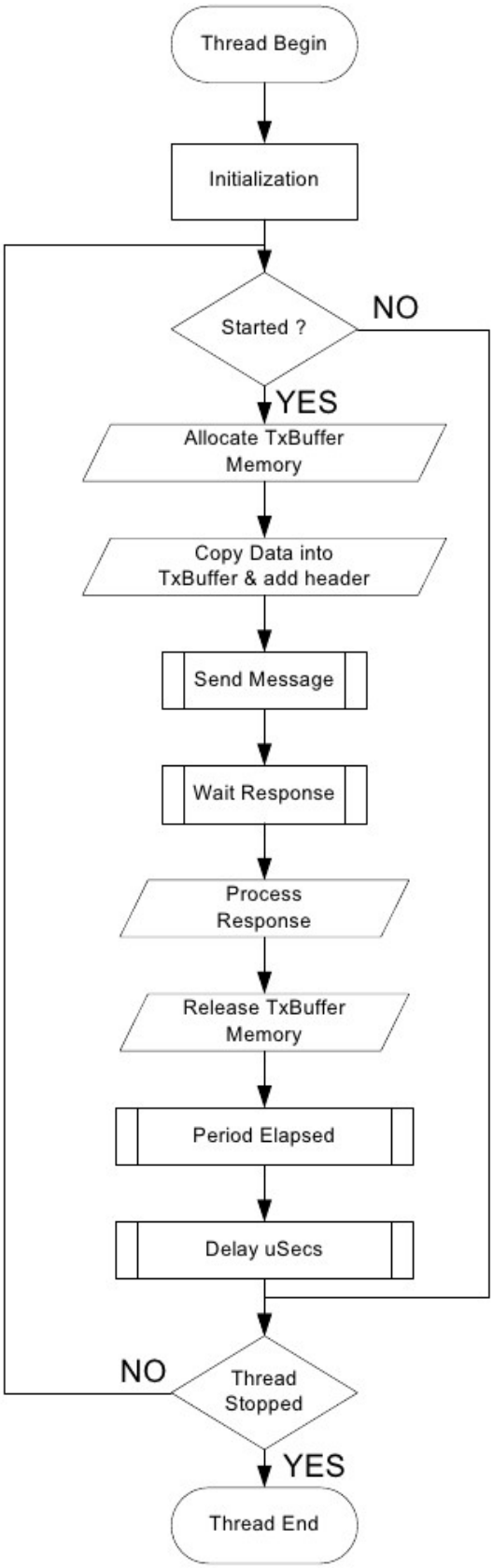


Abbildung 17: Thread Architektur

6.4 TEST MIT SOCAT

Um den Treiber ohne Hardware zu testen, kann man Socat benutzen.

Ziel ist es, die Soundkarte zu simulieren. Das Werkzeug von Socat, das wir dazu benutzen können ist „stdio“.

Socat braucht als Parameter die Adressen des Client und ders Servers.

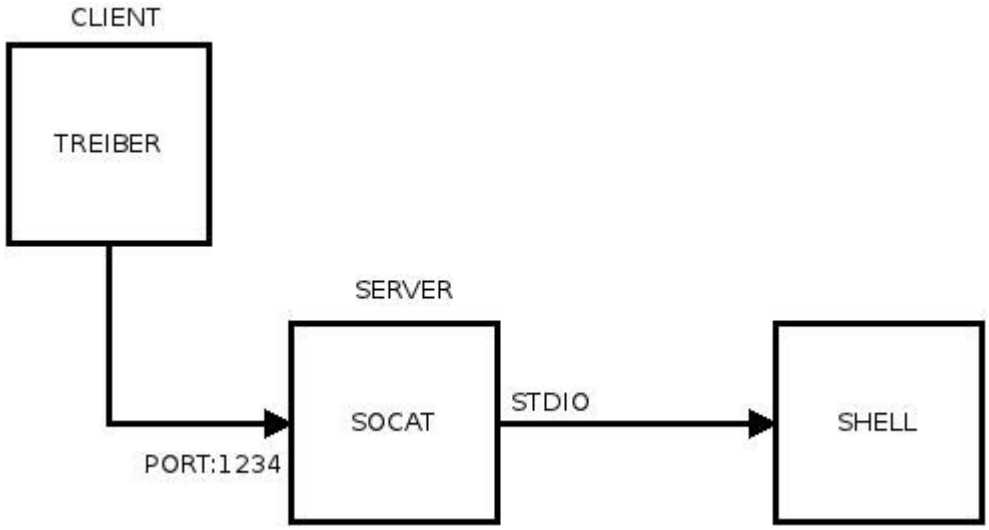


Abbildung 18: Socat Treiber Test

Das Kommando ist:

 # socat -x STDIO TCP-LISTEN:1234

Die Option -x beschreibt, daß die Datenausgabe im Hexadezimal-Format sein muss. Weiterhin gibt es die zwei Parameter mit STDIO und TCP-LISTEN:1234.

- STDIO: Benutzt die Datei Descriptor, welche die Daten direkt auf der Konsole darstellt
- TCP-LISTEN: Überwacht einen Port (hier der Port 1234) und akzeptiert ein TCP/IP Verbindung.

Zum Testen braucht man auch einen Soundtester. ALSA unterstützt zwei Programmen für diese Anwendung.

- Aplay: Ein Soundplayer mit einem Kommando für ALSA. Nachfolgend gibt es ein Beispiel um die Datei „test.wav“ zu spielen. Mit der Option –D wird Ausgangsgerät (Soundkarte Nummer 1 und PCM 4) gewählt

  aplay -D hw:1,4 test.wav

- Speaker-test: Lautstärke-Tester mit einem Kommando für ALSA. Nachfolgend gibt es ein Beispiel um einen Sinus zu spielen. Die Option -D legt das zu verwendende PCM-Gerät fest.

  Speaker-test -D hw:1,4 -l 4 -t sine

7 VERBINDUNG TCP/IP

7.1 ALLGEMEINHALT

TCP/IP oder „Transmission Control Protocol / Internet Protocol“ ist eine Familie von Netzwerkprotokollen und wird wegen ihrer großen Bedeutung für das Internet auch als Internetprotokollfamilie bezeichnet.

Die Identifizierung der am Netzwerk teilnehmenden Rechner geschieht über IP-Addressen. Ein Rechner oder allgemein ein Gerät mit IP-Adresse wird im TCP/IP-Jargon als *Host* bezeichnet. Zuerst wurde TCP als monolitisches Netzwerkprotokoll entwickelt, jedoch später in die Protokolle IP und TCP aufgeteilt. Die Kerngruppe der Protokollfamilie wird durch das User Datagram Protocol (UDP) als weiteres Transportprotokoll ergänzt. Außerdem gibt es zahlreiche Hilfs- und Anwendungsprotokolle, wie zum Beispiel DHCP und ARP.

7.1.1 Warum TCP/IP mit Ethernet

Zuerst wollen wir nur die Ethernet-Schicht benutzen (Physikalische Schicht). Aber dann haben wir mit TCP/IP Stack entwickelt. Die Vorteile und Nachteile, um TCP/IP zu benutzen, sind nachfolgend beschrieben:

Vorteile :

- Verbindung basiert: Der erste Teil von TCP/IP ist es, eine Verbindung herzustellen. Die Daten werden sicher geschickt
- robust und zuverlässig: Mit TCP/IP gibt es verschiedene Kontrollen, die Fehlerkontrolle mit einem „Checksum“ und eine Kontrolle des Verloren-Pakets mit der Sequenz-Anzahl
- Wenn wir den TCP-IP Stack benutzen, können wir die physikalische Schicht leicht ändern. Beispielsweise können wir mit einer Wifi-Verbindung arbeiten.

Nachteile :

- Langsamer: Mit der Verbindungskontrolle dauert es länger, wenn ein Paket verloren geht, jedoch kommen alle Pakete am Ziel an

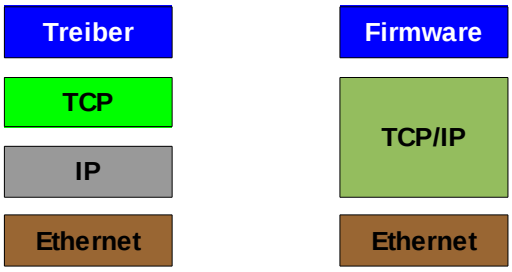


Abbildung 19: Beschreibung der Schichten

Die Abbildung zeigt eine Beschreibung der Schichten. Auf dem Rechner sind die Schichten gespalten und es gibt eine Kommunikation zwischen allen Schichten.

Auf der Soundkarte gibt es nur einen Unterschied, die TCP und IP Schichten sind zusammen implementiert.

7.2 PROTOKOLL

Um den Sound und die Informationen auszurichten, muss man ein Protokoll definieren. Das Protokoll hat zwei Teile, den „SND_HEADER“ und die Daten. Der Header hat nur einen Byte und die Daten zwischen 1 und 732 Bytes (Diese Größe kann ungleich sein, entsprechend der Kapazität der physikalischen Schicht).

Nachfolgend alle Komponenten einer Ethernet-Nachricht mit allen Headern, Trailern und Daten:

Abbildung 20: Ethernet-Nachricht

Nachfolgend die Beschreibung des Soundprotokolls:

	SND HEADER BYTE 0	BYTE 1	BYTE 2	BYTE 3	BYTE 4	BYTE ...	BYTE N-1	BYTE N
Information	1	RATE	FORMAT	CHANNEL	---	---	---	---
Control	2	MUTE	VOLUME		---	---	---	---
Data driver->Card	3	PACKET NUMBER		DATA				
				SAMPLE 0		...	SAMPLE N	
Data Card->Driver	3	REQUEST MORE-LESS	---	---	---	---	---	---
Start/Stop	4	REQUEST START-STOP	---	---	---	---	---	---

Abbildung 21: Protokoll

- Information: Die Header Nummer ist '1' und sie umfasst drei Bytes mit Informationen über die Rate, das Format und die Kanalanzahl. Um nur ein Byte pro Daten?? zu benutzen, haben wir alle Werte in der nächsten Tabelle definiert:

RATE	BYTE	1	2	3	4	5	6	7	8	9	10	11	12
	VALUE [Hz]	8000	11025	16000	22050	32000	44100	48000	64000	88200	96000	176400	192000

FORMAT	BYTE	1	2	3	4
	VALUE	S8	S16	S24	S32

Abbildung 22: Soundprotokoll -> Rate & Format

Alle Rate-Werte sind normalisiert. Die Soundkarte benutzt nicht alle Werte, sondern nur von 8KHz bis 96KHz, jedoch alle werden implementiert.. Das Format hat viele verschieden Werte, aber nur die „Signed“ Werte werden von der Karte benutzt. „S“ bedeutet „Signed“ und die Nummer gibt die Bit-Anzahl wieder. Alle Werte sind möglich.

- Control: Mit der Kontrolle kann man die Lautstärke modifizieren und den Ausgang „On“ oder „Off“ schalten (« Mute »). Der Treiber kann die Daten empfangen oder senden. So ist dieser Datentyp in beide Richtungen gültig.
- Data: Der Datentyp für die Daten ist ungültig in beide Richtungen.
  - Treiber->Karte: Der Treiber gibt jedem Paket einen fortlaufende 16 Bit Zähler mit. So kann die Soundkarte die Reihenfolge aller eingehenden Pakete überprüfen und ggfs. umordnen.
  - Karte->Treiber: Wenn die Daten angekommen sind, kann die Karte eine Antwort schicken. Diese Antwort enthält eine Flusskontrolle. Die Soundkarte weiß, wie viele Samples es schon in ihren Puffern gibt. Sie kann also sagen, wenn es zu viele oder zu wenig Samples gibt. Wenn es zu viele Samples gibt, fordert die Soundkarte den Kerneltreiber auf, die Datenrate zu senken. Droht der Puffer in der Soundkarte leer zu laufen, so fordert die Soundkarte den Kerneltreiber auf, seine Datenrate zu erhöhen.

    Nachfolgend ein Datenaustausch zwischen dem Treiber und der Soundkarte.

    In diesem Beispiel gibt es eine Basiszeit von 7ms. Wenn die Soundkarte „gleich“ sagt, muss der Treiber die nächste Nachricht in der gleichen Zeit schicken. Aber bei einem „langsamer“ muss der Treiber die Zeit absenken und mit einem „schneller“, die Zeit dann verringern.

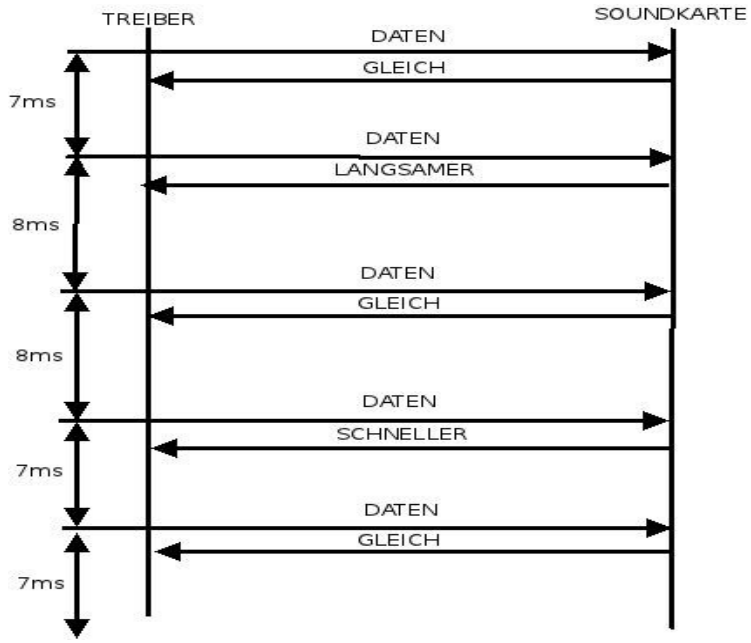


Abbildung 23: Datenaustausch

Die Antwort ist:

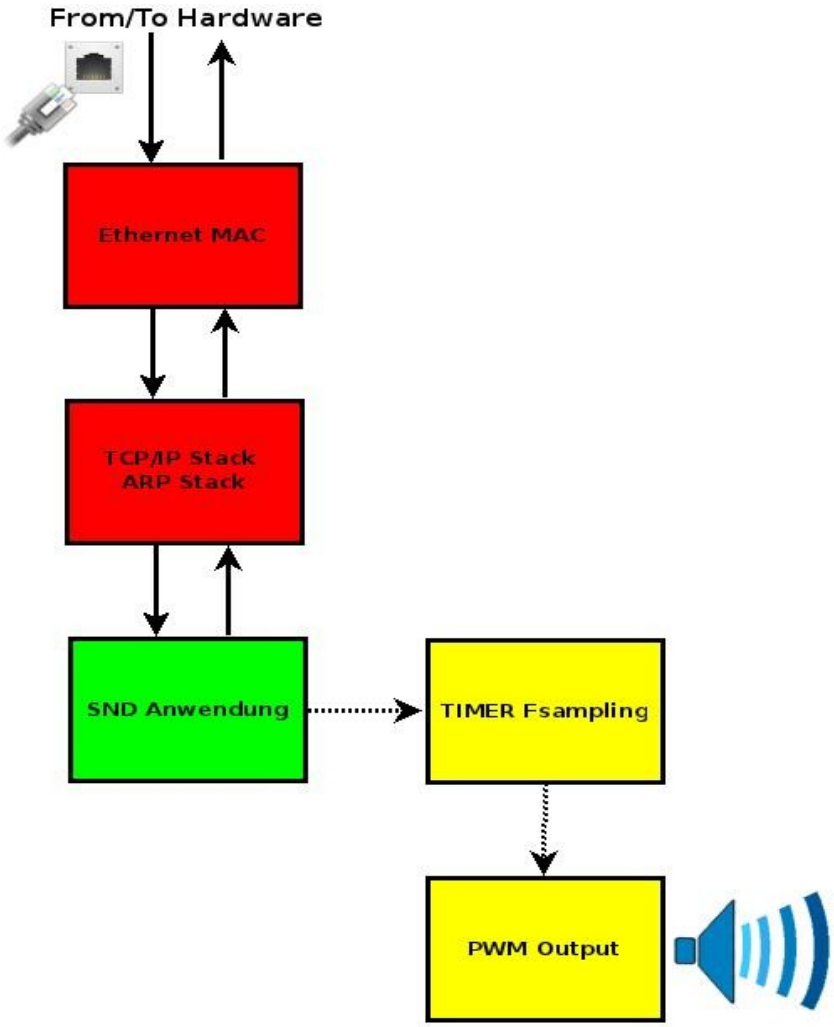
- Gleich = '0'
- Langsamer = '-1'
- Schneller = '1'

- Start/Stopp: Nachricht um die Soundkarte zu informieren, daß sie den Ausgangsstrom starten oder stoppen soll

8 FIRMWARE SOUNDKARTE

8.1 ALLGEMEINHEIT

Nachfolgend die Struktur der Soundkarte Firmware:

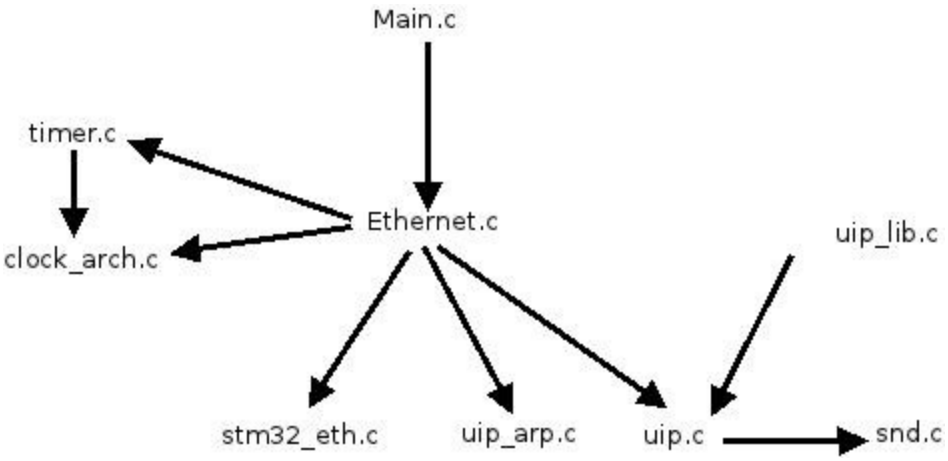


- Zunächst gibt es die Zugangsschicht Ethernet. Der Treiber hierfür kommt von der Firma ST-Microelectronics. Die Funktionen der Softwarebibliothek interagieren direkt mit der Hardware.
- Dann gibt es die TCP/IP Schicht. Diese Schicht wird in Punkt „7.2 TCP/IP Stack „ beschrieben. Der hier verwendete TCP/IP Stack ist eine sehr rudimentäre Implementierung, die von Adam Dunkels geschrieben wurde.
- Die ARP-Schicht gehört ebenfalls zu dem TCP/IP Stack. ARP für „Address Resolution Protocol“ und wird benutzt, um eineIP Adresse in eine Zugangsschichtadresse (Hardwareadresse) zu übersetzen. ARP benutzt die „broadcast“ Anfrage um die MAC Adresse von einer bekannten IP-Adresse zu erfragen. Daraufhin muss der Server, der diese IP Adresse hat mit seiner MAC Adresse antworten.

- Danach folgt die Soundkartenanwendung, welche alle Datenpakete, die ankommen verwaltet.
- Am Ende gibt es einen Timer mit zwei PWM-Ausgängen (mono und stereo), um den Sound abzuspielen.

Datei Architektur

Nachfolgend die Beschreibung der Dateiarchitektur:

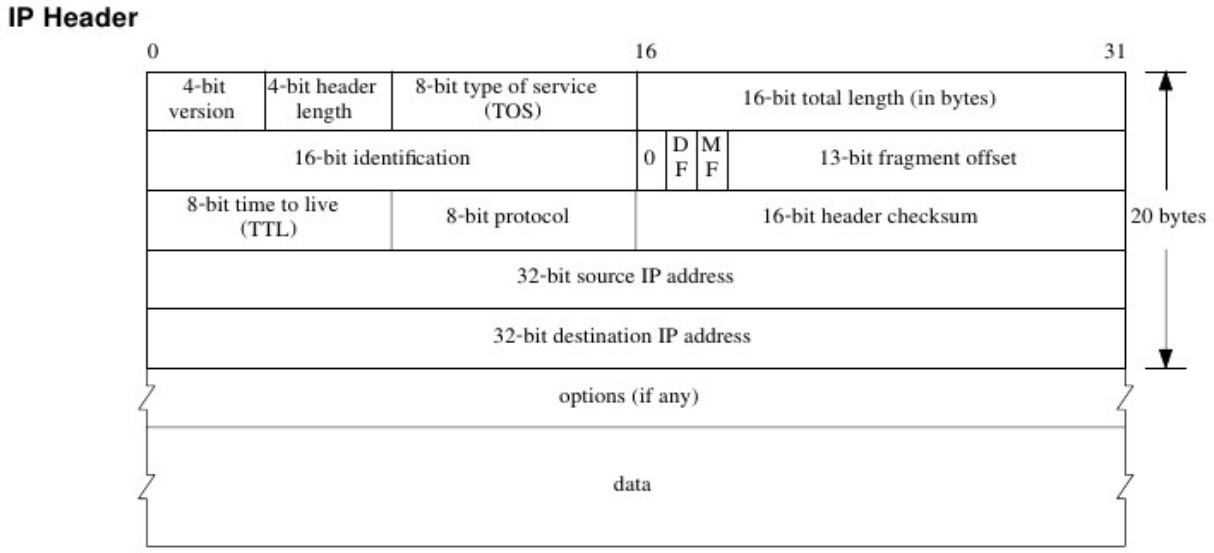


- Main.c: Enthält die Main Funktion mit allen Konfigurationen für die Ausgänge und Eingänge, für Ethernet-Verbindungen und für alle Timer und PWM.
- Ethernet.c: Enthält die Basis-Schleife mit dem Kern des Programms. Diese Schleife macht mehrere Schritte :
    1. Liest den Ethernet Puffer
    2. Testet den Nachrichten-Typ zwischen ETH_IP oder ETH_ARP.
    3. Bearbeitet die Nachricht. Für ARP, sendet wieder einen Antwort. Für IP, ruft den IP-Schicht mit dem Nachricht als Parameter.
    4. Testet auch die Verbindungszeit mit einem Timer. Wenn die Zeit abgelaufen ist, macht man in der Schleife einen ARP-Antrag.
- stm32_eth.c: Das ist der Treiber für die Ethernet-Schnittstelle auf STM32. Diese Schicht arbeitet direkt mit der Hardware.
- uip_arp.c: Wenn eine Nachricht mit einem ARP-Typ kommt, ruft man die Funktion in diese Datei. Sie enthält alle Funktionen für ARP.
- uip.c: Enthält den TCP-IP Stack. Die Beschreibung dieser Schicht befindet sich in Punkt 7.2.
- snd.c: Das ist die Anwendungsdatei mit allen spezifischen Funktionen für die Soundkarte.
- uip_lib.c: Enthält eine übliche Funktion, um eine IP-Adresse von einem Textformat in ein nummerisches Format zu ändern.
- clock_arch.c: Enthält die Initialisierung einer Timer-Funktion für die Basiszeit des Systems.
- timer.c: Enthält die Verwaltung mehrerer Software-Timer, gestützt auf den „clock_arch“ Timer.

8.2 TCP-IP STACK

8.2.1 IP Process

Nachfolgend die Beschreibung des IP-Headers:



- Version: Dank dieses Feldes können verschiedene Versionen des IP-Protokolls (verschiedene Header-Formate) auf demselben Netzwerk vorhanden sein. Gegenwärtig wird die Version 4 verwendet. Die nächste Version (Version 6, IPv6) wurde zwar schon definiert, deren breite Nutzung ist mittelfristig jedoch nicht vorgesehen.
- IHL: Internet Header Length: Dieses Feld gibt die Größe des Headers des IP-Pakets an. Die Einheit ist das 32 Bit lange Wort. Dieses Feld ist für den Header notwendig, da es optionale Teile enthalten kann (Feld Options).
- TOS: Type of Service
- Total Length: Dieses Feld gibt die Gesamtlänge des IP-Pakets an (Anzahl Bytes).
- Identification: Wenn ein Paket aufgeteilt ist, ist dieses Feld die Fragment-Nummer.
- DF: Don't Fragment: Gibt an, ob das Paket aufgeteilt werden kann.
- MF: More Fragment: Gibt an, ob das Paket ein Fragment ist.
- Fragment Offset: Zeigt die Position der Fragmente in dem vollen Datagramm.
- TTL: Time To Live: Wenn die Routing-Tabelle eines oder mehrerer Gateways Fehler enthält, kann es vorkommen, daß ein Paket unendlich lange in der Schleife bleibt. Um dadurch ausgelöste Überlastungen zu vermeiden, ist die Lebensdauer der Pakete begrenzt. Ursprünglich sollte jeder Gateway den Wert des Felds TTL um die Zahl verkleinern, die der seit dem Durchgang im vorhergehenden Gateway vergangenen Zeit (in s) entsprach. In der Praxis wird TTL von jedem Gateway um eine Einheit verkleinert. Das Paket wird von den Gateways zerstört, sobald TTL 0 erreicht (in diesem Fall wird der Quellstation eine Fehlermeldung geschickt). Der Ausgangswert von TTL wurde auf 255 festgelegt (diese Norm wird jedoch nicht immer eingehalten).

- Protocol: Dieses Feld gibt den Inhalt der Benutzerdaten an. Für TCP beispielsweise ist der Wert '6' und für ICMP '1'.
- Header Checksum: Dank dieses Feldes kann die Vollständigkeit des Headers des IP-Pakets überprüft werden. Wenn im Header ein Fehler entdeckt wird, wird das Paket zerstört.
- Source IP-Adress: Die IP-Adresse der Quellstation.
- Destination IP-Adress: Die IP-Adresse der Zielstation.

Alle Schritte des IP-Stacks, zur Überprüfung eines IP-Paketes:

1. Test IP Version and Header Length if false -> drop and ending
2. Überprüft die Paketgröße. Wenn die Größe, die wir bekommen haben in „uip_len“ (Wert von Ethernet-Schicht) kleiner als der Wert in dem IP-Header ist, bedeutet dies, daß das Paket verdorben ist. Wenn der Wert in „uip_len größer ist als in dem IP-Header, enthält das Paket ungenutzte Felder und verschwendet Übertragugsbandbreite.
3. Überprüft die Fragment-Schalter. Wenn das Paket aufgeteilt wurde, wird es verworfen. Fragmentierung wird vom hier verwendeten TCP/IP-Stack nicht unterstützt.
4. Kontrolliert, daß die Zieladresse gleich unserer IP-Adresse ist. Wenn die Adresse nicht unsere ist, muss das Paket gelöscht werden.
5. Kalkuliert den Checksum Wert mit der Funktion „uip_checksum()“ und vergleicht es mit dem Header-Feld „Checksum“. Wenn diese nicht gleich sind, muss das Paket ebenfalls gelöscht werden.

Nach diesen Schritten kann das Paket in die TCP-Schicht geschickt werden.

8.2.2 TCP Process

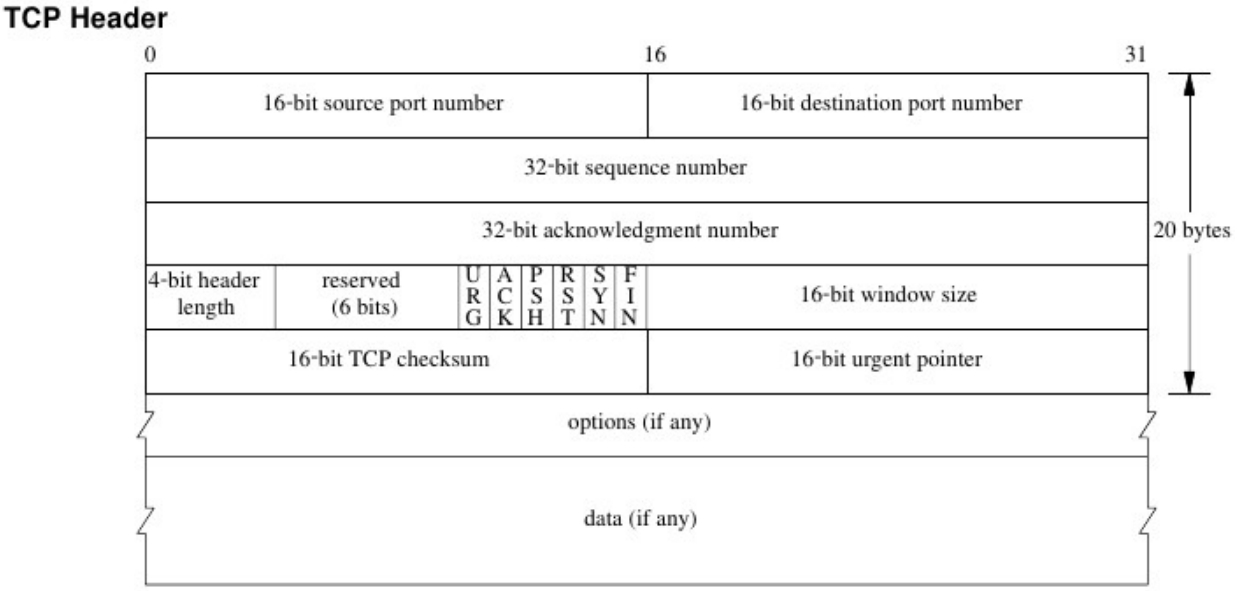


Abbildung 27: TCP Header (cf. Ref. 11)

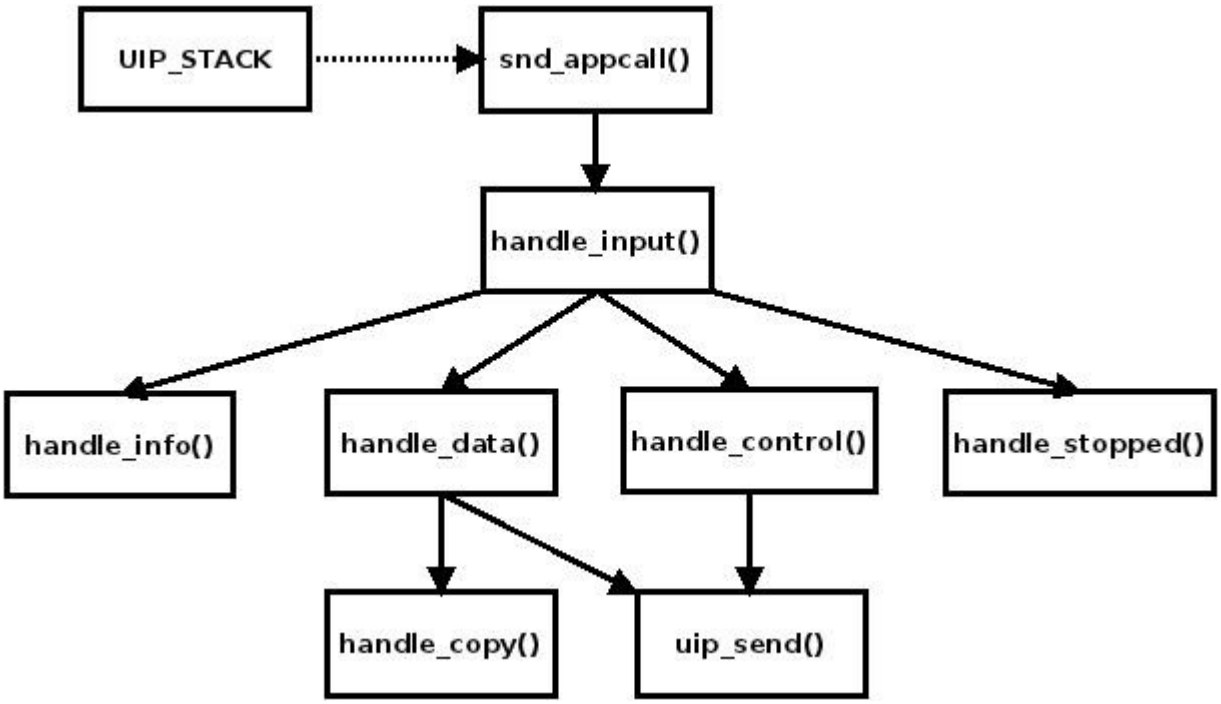
- Source Port Number, Destination Port Number: Source und Destination Port Nummer . Die TCP-Port-Nummern sind analog zu den UDP-Port-Nummern (Serverportnummer ist dem Kunden bekannt, Clientportnummer wird vom Betriebssystem verwaltet).
- Sequence Number: Sequenznummer des nächsten Bytes von A in Richtung B.
- Acknowledge Number: Bestätigungsnummer für die von B nach A übertragenen Bytes (alle Bytes bis ausschliesslich der Acknowledge Number sind bestätigt).
- Code Bits
  - ACK gibt an, daß die Quelle die in der Bestätigungsnummer angegebenen Rahmen quittiert.
  - SYN ermöglicht es, die Verbindung zu initialisieren und so die Sequenznummern zu synchronisieren.
  - PSH weist den Empfänger darauf hin, daß die nachfolgenden Informationen unmittelbar an die Anwendung übermittelt werden müssen.
  - RST ermöglicht das Rücksetzen der Verbindung.
  - URG gibt an, daß sich im Feld "Data" dringende Informationen befinden.
  - FIN zeigt dem Empfänger an, daß der Datenstrom nach diesem Rahmen abbrechen wird (Unterbrechung der Verbindung).
- Window: A gibt dem Sender von B die Größe des Schiebefensters NFC (von A!) an.
- Checksum: Es handelt sich dabei um ein Feld des Typs CRC für die Kontrolle von Fehlern.
- Data: Dies sind Benutzerdaten. Ein TCP-Segment muss nicht unbedingt Benutzerdaten enthalten.

Beschreibung der TCP-Schicht :

1. Kalkuliert den Checksum Wert mit der Funktion „uip_tcpchksum()“ und vergleicht dies mit dem Header-Feld „Checksum“. Wenn die Checksummen ungleich sind, muss das Paket gelöscht werden.
2. Sucht eine aktive Verbindung. Wenn es keine aktive Verbindung gibt, die dieses Paket braucht, bedeutet dies, daß es ein altes Duplikat ist oder das es ein SYN-Paket für eine Verbindung in LISTEN ist. Wenn der SYN-Schalter in dem Header nicht definiert wird, ist es ein altes Paket und man muss eine RST-Nachricht verschicken.
3. Wenn das Paket ein SYN-Paket für eine Verbindung in LISTEN ist, überprüft man, ob eine Verbindung verfügbar ist. Unbenutzte Verbindungen findet man in der gleichen Tabelle wie auch die benutzten Verbindungen. Aber die unbenutzten Verbindungen haben einen speziellen Schalter „CLOSED“. Wenn keine unbenutzte Verbindung gefunden wird, kann man eine „TIME_WAIT“ Verbindung benutzen. Das ist eine aktive, jedoch nicht benutzte Verbindung.
4. Wenn keine unbenutzte Verbindung gefunden wird, löscht man die Nachricht. Sonst füllt man alle richtigen Felder für eine neue Verbindung aus.
5. Dann schickt man eine ACK-Antwort.
6. Wenn eine aktive Verbindung für die Nachricht gefunden wird, kann man die Nachricht bearbeiten.
7. Wenn der Nachrichten-Code RST (Reset) ist, löscht man die Verbindung (Status = CLOSED) und informiert die Anwendung, daß die Verbindung gelöscht wurde.
8. Überprüft, ob die Sequenz-Nummer des arrivierten Pakets richtig ist. Sonst schickt man eine ACK-Nachricht mit der korrekten Anzahl.
9. Überprüft, ob die Nachricht ein ACK ist, ggf. modifiziert man die Sequenz-Anzahl Feld.
10. Jetzt kann man die Nachricht bearbeiten. Man überprüft die Verbindung Status (LISTEN und CLOSED sind schon überprüft) :
    - UIP_SYN_RCVD: Ein SYNACK wurde als Antwort auf ein SYN geschickt. Jetzt wird auf ein ACK der versendeten Daten mit aktiviertem UIP_ACKDATA. In diesem Fall ist der nächste Zustand ESTABLISHED.
    - UIP_ESTABLISHED: In diesem Zustand kann man die Anwendung anrufen um den Puffer auszufüllen. Wenn der UIP_ACKDATA Schalter aktiv ist, kann die Anwendung neue Daten in den Puffer schreiben, ansonsten schickt man ein altes Segment.
    - UIP_LAST_ACK: Man kann diese Verbindung schließen, wenn der Client unsere FIN bestätigt. Das wird vom UIP_ACKDATA Schalter angezeigt.
    - UIP_FIN_WAIT_1: Die Anwendung hat die Verbindung geschlossen, aber der Client noch nicht. In diesem Zustand wartet man auf eine FIN-Antwort des Client.
    - UIP_TIME_WAIT: In diesem Zustand schickt man ein ACK, da die Verbindung keine Daten benutzt.
    - UIP_CLOSING : Die Verbindung wird jetzt geschlossen.

8.3 SOUNDKARTE ANWENDUNG

Nachfolgend die Architektur der Soundkarten-Anwendung:



- snd_appcall(): Das ist die Basisfunktion der Anwendung. Diese wird vom TCP/IP Stack aufgerufen, wenn eine Nachricht ankommt. Hier muss man den Status für den Verbindungszustand überprüfen. Wenn der Zustand „CLOSED“, „ABORTED“ oder „TIMEDOUT“ ist, muss man die Funktion „handle_stopped() anrufen, um alle Pufferparameter noch einmal zu initialisieren. Wenn der Zustand „CONNECTED“ ist, bedeutet dies, daß eine Verbindung erstellt wurde. Ist der Zustand „NEWDATA“ erreicht, wird „handle_input()“ aufgerufen, um die Daten zu verarbeiten.
- handle_input(): Entscheided über die weitere Verarbeitung anhand des ersten Datenbytes:
  - Byte 0 == '1' ist, ruft man „handle_info()“ an.
  - Byte 0 == '2' ist, ruft man „handle_control()“ an.
  - Byte 0 == '3' ist, ruft man „handle_data()“ an.
  - Byte 0 == '4' ist, ruft man „handle_stopped()“ an.
- handle_info(): Setzt neue Einstellungen in der Soundkarte wie z.B. die Abspielrate. Die Abspielrate bestimmt die Frequenz mit welcher die Timerbausteine die verschiedenen PWM-Signale generieren.
- handle_data(): In dieser Funktion macht man drei Etappen. Am Anfang prüft man die Paketnummer. Dann kann man die Daten in einen lokalen Puffer kopieren (Anruf der Funktion „handle_copy“) und danach die Antwort vorbereiten und verschicken (mit „uip_send“). Die Antwort enthält, ob man die Daten schneller oder langsamer bekommen will. Um zu wissen, welche Antwort man schicken muss, guckt man wie viele Samples es noch in dem Puffer gibt. Wenn es weniger als 800 Bytes gibt, braucht man

die Daten schneller, wenn es mehr als „BUFFER_SIZE – 800“ gibt, braucht man die Daten langsamer.

- handle_control(): Man nimmt die Werte (Mute und Volume) um den Ausgangszustand zu modifizieren und schickt eine Nachricht mit dem gleichen Wert zurück.
- handle_stopped(): Mit diesem Funktionsanruf werden alle Ausgänge gestoppt. Man stoppt den Timer und initialisiert die Pufferwerte.

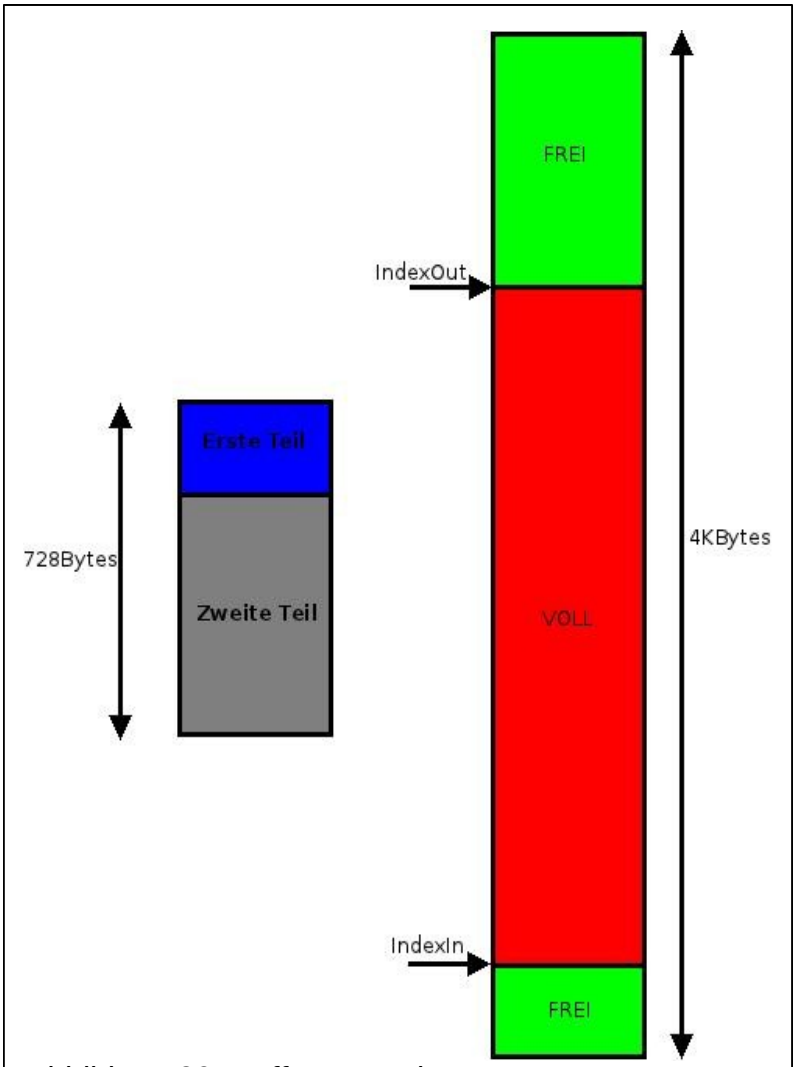


Abbildung 29: Puffer Verwaltung

- handle_copy(): Es gibt eine Problematik mit einem Ringpuffer. Wenn es nicht mehr genügend Platz für alle Daten gibt, die kommen muss man in zwei Teile kopieren. Den ersten Teil am Ende des Puffers und den zweiten Teil am Anfang des Puffers. Wenn es genügend Platz gibt, kann man alles zusammen kopieren.

In dem Puffer benutzt man zwei Indices. IndexOut gibt den nächsten Wert an, den man im Ausgang spielen kann. IndexIn gibt den letzten Wert an, der kopiert wurde.

- uip_send(): Diese Funktion gehört zu dem TCP/IP Stack. Dieser benötigt zwei Parameter, den Pufferzeiger, der geschickt werden muss und die Puffergröße.

8.4 TEST MIT SOCAT

Dieses Mal wird SOCAT wie ein Client benutzt. Das Ziel dieses Tests ist es, das Protokoll der Soundkarte zu überprüfen.

Im ersten Teil schickt man einen Antrag und überprüft die Antwort auf der Konsole mit SOCAT.

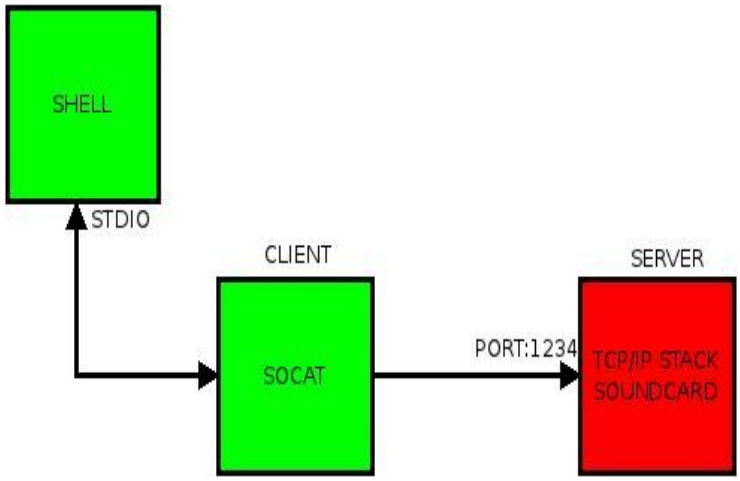


Abbildung 30: Socat -> Firmware Test

Das Kommando ist fast wie der vorherige Test mit SOCAT (Punkt 5.4):

 # socat -x STDIO TCP:192.168.0.111:1234

Der Unterschied ist, daß man nun nicht LISTEN (für einen Server) gibt, jedoch die IP-ADRESSE des Servers.

Somit ist der erste Parameter STDIO der Client und der zweite ist die Server-Adresse.

9 LAUTSPRECHER-AUSGANG PWM

9.1 ALLGEMEIN

Die Soundkarte bekommt jetzt die Samples von dem Rechner und speichert alle Daten in einem Puffer. Das letzte Ziel ist den Sound auf einem Lautsprecher zu spielen.

In unserem Fall gibt es zwei Techniken, die hierbei möglich sind. Es gibt die einfache Lösung, die Benutzung eines DAC-Wandler auf dem Mikrocontroller. Die zweite Lösung ist einen nummerischen Ausgang zu benutzen. Das nummerische Signal ist eine PWM (Pulse Width Modulation).

9.2 PULSE WIDTH MODULATION

Dieses Kapitel erklärt wie man ein Analogsignal mit einem PWM-Signal darstellen kann.

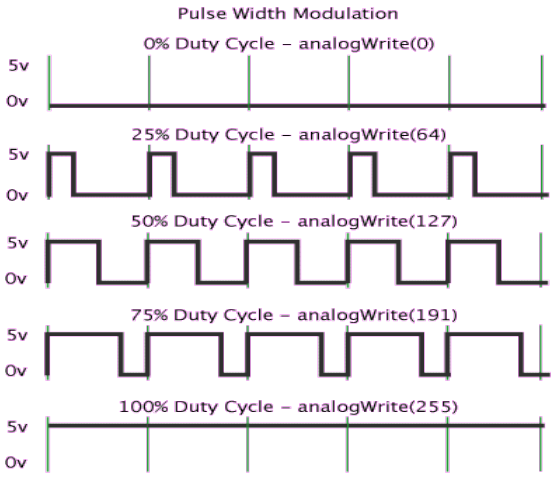


Abbildung 31: PWM Signal (cf. Ref. 13)

Nach rechts gibt es ein PWM-Beispiel mit verschiedenen Werte.

Eine PWM oder Pulsweitenmodulation ist eine Modulationsart, bei der eine technische Größe zwischen zwei Werten wechselt (hier ist 0V und 5V). Dabei wird bei konstanter Frequenz der Tastgrad (duty-cyle) des Signales moduliert, also die Breite eines Impulses.

Der Tastgrad stellt einen nummerischen Wert dar. Mit einem Wert auf 8 bit (1 Byte) hat man beispielsweise diese Darstellung:

Für 0% Duty-Cycle hat man den Wert '0'; für 25% den Wert '64', und so weiter …

Alle PWM-Signale stellen einen Analogwert dar. Ein Duty-Cycle hat einen Mittelwert. Zum Beispiel hat ein D-C von 25% den Mittelwert 1,25V mit einer maximalen Spannung von 5V.

In unserem Fall ist es dieser Mittelwert, der benutzt wird.

Nachfolgend illustriert dieses Bild ein PWM-Signal und einen Ausgangssinus. Das ist eine Darstellung von einem Sinus mit einem nummerischen Signal:

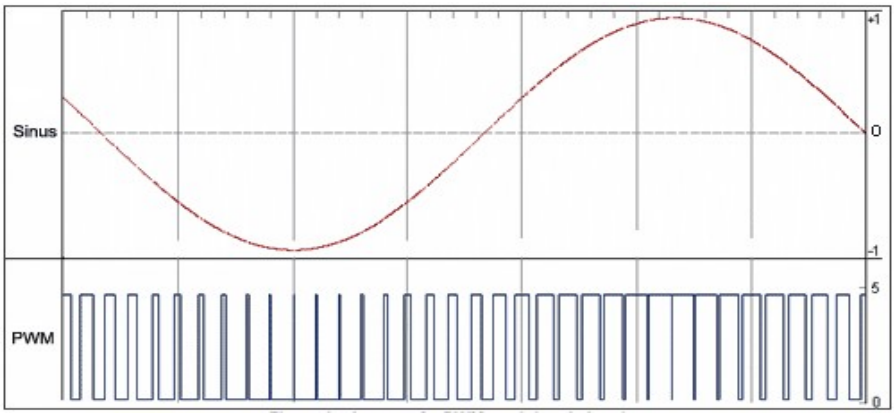


Abbildung 32: Darstellung PWM Sinus (cf. Ref. 12)

9.3 SCHEMA

Die Ausgangsplatine ist sehr einfach:

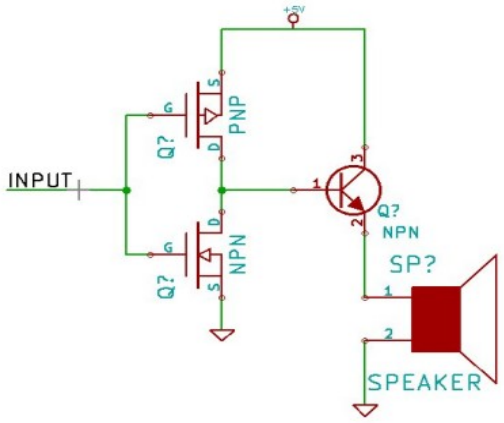


Abbildung 33: Ausgangplatine

- Der „Input“ ist dirkt an den PWM-Ausgang der Mikrocontroller angeschlossen.
- Die Spannung ist 5V. Es gibt eine Wechsle-Montage mit zwei MOSFET (PNP und NPN). Wenn der Eingang 5V ist, ist der Ausgang 0V, da der NPN Mosfet leiten kann (Vgs positiv). Wenn der Eingang 0V ist, ist der Ausgang 0V, weil der PNP Mosfet jetzt leiten kann (Vgs negativ). Es gibt nun einen Bipolar-Transistor, um die Lautstärke mit Strom anzutreiben.

Diese Ausgangsplatine ist nur zum Testen und nicht optimiert für unsere Anwendung (Leistung, Arbeitsfrequenz, …).

Nachfolgend der PWM Eingang (unten) mit dem Mosfet Ausgang (oben):

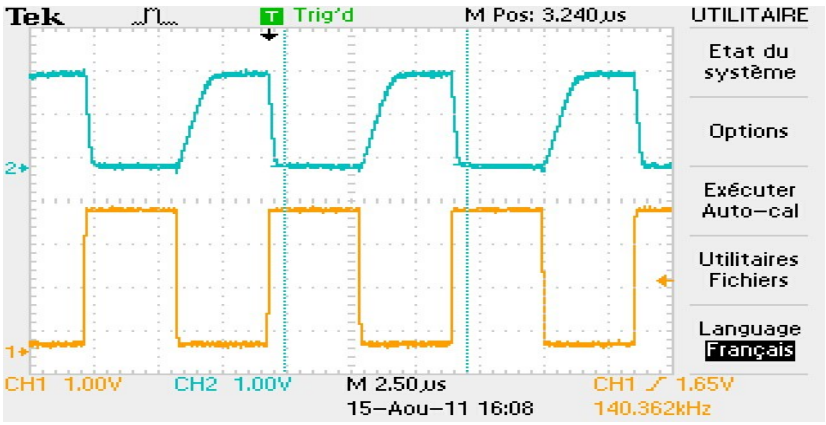
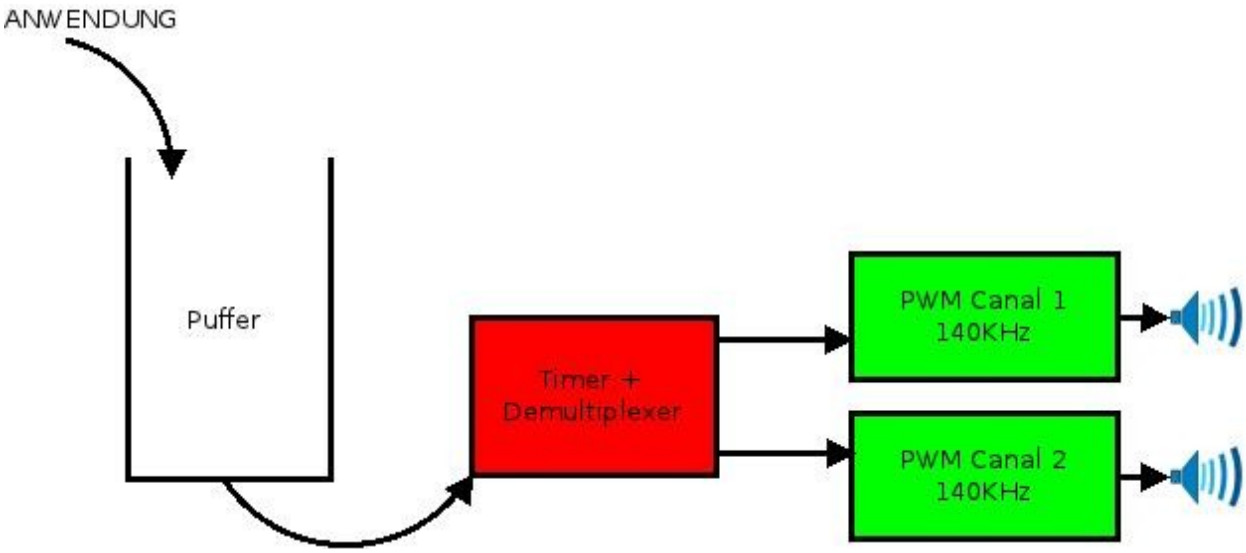


Abbildung 34: PWM-Ausgang Messung

Die Anstiegszeit (RiseTime) des Ausgangs ist sehr lang, weil der Eingang nicht 5V sondern 3,3V hat.

9.4 IMPLEMENTIERUNG



1. Am Anfang der Firmware ist der Timer mit 32 Khz vorkonfiguriert. Wenn die Informationen aus dem Rechner kommen, kann die Anwendung den Timer konfigurieren. Beispielsweise bekommt man die Rate-Information 48KHz. Dann gibt man den richtigen Wert für den Timer ein:

   TIM5->ARR = configCPU_CLOCK_HZ/(frequency);

   „TIM5->ARR“ ist der Zählerregister. Am Anfang hat man keine Prescaler und ClockDivision gegeben. Somit ist die Formel sehr einfach. Die CPU-Clock dividiert durch die Frequenz. In unserem Beispiel ist das 72MHz / 48KHz = 1500. Also der Timer zählt von 0 bis 1500 und wird dann reinitialisiert und der Timer wird unterbrochen.

2. In dieser Unterbrechung testet man, ob das Audiosample gestartet ist. Welches man feststellen kann wenn das erste Paket ankommt.

3. Dann wartet man, bis der Puffer halb gefüllt ist (bei einem Puffer mit 4KBytes braucht man mindestens 2048 Bytes). Es ist sehr wichtig dieses ab zu warten, da wenn es nur ein Paket in dem Puffer gibt und man ein Problem mit dem nächsten Paket bekommt, man eine Sicherung braucht.

4. Dann kann man die Daten behandeln. Man bekommt auch den Lautstärke-Wert aus dem Treiber. Dieser Wert liegt zwischen 0 und 255. Für alle Samples multipliziert man den Wert mit dem Wert der Lautstärke. Dann braucht man einen Wert 16 Bit Wert, weshalb man um acht nach rechts shiftet. Eine letzte Behandlung muss gemacht werden. Jetzt ist der Wert „Signed“ von -32768 bis 32767 für 16bit. Der PWM-Wert muss „Unsigned“ sein. So kann man ein Offset von 32768 machen.

5. Dann muss man das Format testen.
6. Danach muss man die Anzahl der Kanäle überprüfen. Es gibt nur zwei Möglichkeiten: ein Kanal oder zwei Kanäle. Wenn es zwei Kanäle gibt, sind die Samples im Gefolge (mit zwei Samples in dem Puffer, eins ist für einen Kanal und das Zweite für die anderen). Für eine Stereoausgabe werden die Schritte 4 und 5 für das zweite Sample und den zweten Kanal wiederholt.
7. Am Ende sind es dann nur noch zwei Schritte → Inkrementieren der Pufferposition (IndexOut) und Modifizieren des Wertes, wenn der Zahler mehr als der Puffergröße angebt.

10 HARDWARE-ENTWICKLUNG

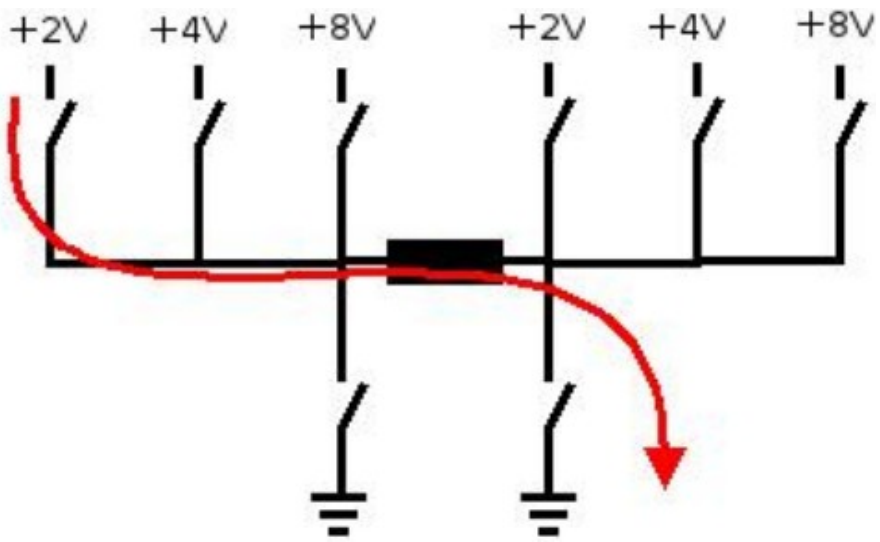
10.1 ZIEL

Das Ziel dieser Hardware-Entwicklung ist die Entwicklung einer eigenständigen Soundkarte mit Ethernetanschluss.

Mit einem PWM Ausgang gibt es kritischen Werten. Ein PWM Wert nah 0% und nah 100% ist sehr schwer zu erzeugen, weil die Übergänge zu schnell sind.

Die Lösung ist drei verschiedenen Spannungen zu benutzen. Wenn der PWM zu klein ist, kann man die Spannung ändern. Zum Beispiel hat ein PWM der Wert 10% mit 6V (Mittelwert gleich 0.6V), mit ein PWM von 2V kann man einen Wert größer haben. So 10% mit 6V ist gleich 30% mit 6V.

Der Auswahl der Spannung wird mit MOSFET Transistoren gesteuert von einem PWM Eingang gemacht. Nachfolgend kann man ein Beispiel sehen, wenn man +2V auf dem Lautsprecher haben will. Man muss zwei MOSFET auswählen, die erste für die 2V Spannung und die zweite für die Masse, damit kann ein Strom fließen.



10.2 BESCHREIBUNG

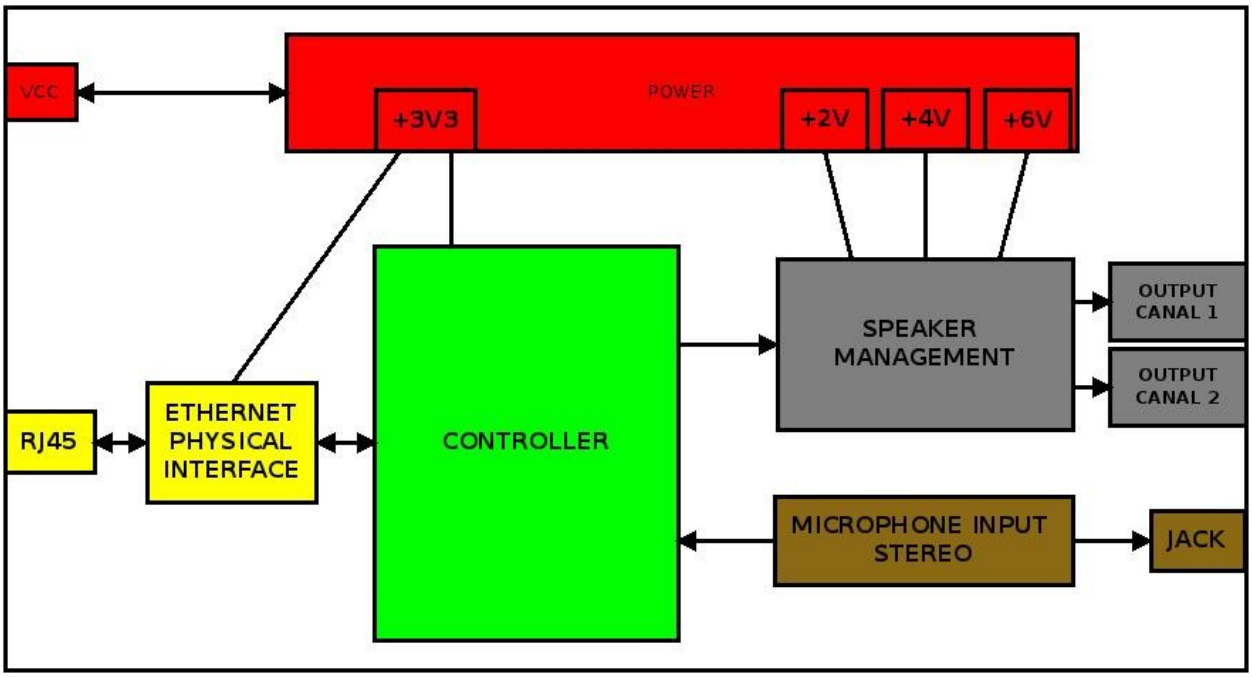


Abbildung 37: Hardware Platine

Die Platine enthält vier Hauptkomponenten.

Zuerst gibt es den Mikrocontroller. Es handelt sich um ein STM32F107 Mikrocontrollermodel mit einem ARM 32-bit Cortex M-3 Kern mit einer maximalem Frequenz von 72 Mhz. Der Mikrocontroller bietet spezielle Funktionseinheite zum Anschluss eines Ethernet-MAC und für die Erzeugung von PWM-Signalen. Er bietet darüber hinaus 128 - 256 kB Flash für Programmcode sowie 32kB RAM.

Die Ethernet Schnittstelle erlaubt die Kommunikation mit dem Desktoprechner und dem darauf laufenden Kernelmodul. Als Ethernet-MAC kommt ein STE101P 10/100 Mhz Fast Ethernet Transceiver zu Einsatz. Mikrocontroller und Ethernet-MAC kommunizieren mittels einer MII-Verbindung (Media Independent Interface).

Auf der Platine wird auch eine voll digitale Endstufe für den direkten Anschluss zweier Lautsprecher enthalten. Die digitale Endstufe hat dabei gegenüber klassischen Typ A und Typ AB Endstufen den Vorteil, dass sie klein und besonders leistungseffizient ist.

Um den Dynamikbereich von nur 256 verwertbaren Werten zu erweitern, werden per DC-DC Wandler drei verschiedene Spannungsebenen zur Verfügung gestellt. Zu jedem Sample kann der Mikrocontroller den Lautsprecher eine der drei Spannungen in Vorwärts- oder Rückwärtsrichtung an den Lautsprecher anlegen. Die Idee dabei ist, den gesamten Dynamikbereich auf drei, aneinander anschliessende, Bereiche aufzuteilen.

Die gesamte Schaltung ist für eine Betriebspannung von 9V Gleichspannung ausgelegt.

Desweiteren existiert ein Mikrophoneingang in Form einer Klinkenbuchse auf der Platine.

10.3 SCHEMA

10.3.1 Power

Für die 3.3V Verwaltung braucht man der LM1117-3.3. Das ist einen Spannungsregler von 3.3V. Die maximale Eingangspannung kann 20V sein, also kann man für unsere Anwendung benutzen. Der maximale Ausgangsstrom ist 800mA, aber der Spannungsregler speist nur der Mikrocontroller und der Transceiver ein. So braucht man nicht mehr als 170mA (68mA für uC mit alle Peripherie eingeschaltet auf 72Mhz, 100mA für Transceiver mit einer Benutzung von 100%).

Nachfolgend gibt es der Schema mit zwei Kondensatoren.

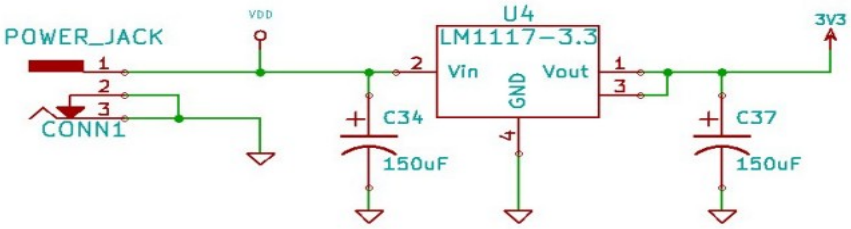


Abbildung 38: LM1117 Schaltplan

Jetzt muss man bestimmen, ob der Layout einen Kühler braucht. Der Verlustleistung von dem Spannungsregler ist P_D = (Vin-Vout)*Iout = (9V-3.3V)*170mA = 0.969W. Im Sinne von dem Datenblatt ist der maximal Temperatur $T_r = Tj_{(max)}-T_a$ = 125°C – 25°C = 100°C wo Tj ist der maximal Junctiontemperatur und Ta ist die Umgebungstemperatur. Jetzt muss man der Thermalwiderstand kalkulieren : $T_{ja} = T_r / P_D$ = 100/0.969 = 103 [°C/W]. Der Datenblatt sagt, daß der Spannungsregler einen Kühler braucht, wenn T_{ja} kleiner als 136 [°C/W] ist. So in unserem Fall braucht man ein Kühlergebiet von ungefähr 2cm^2 (Wert aus dem Datenblatt).

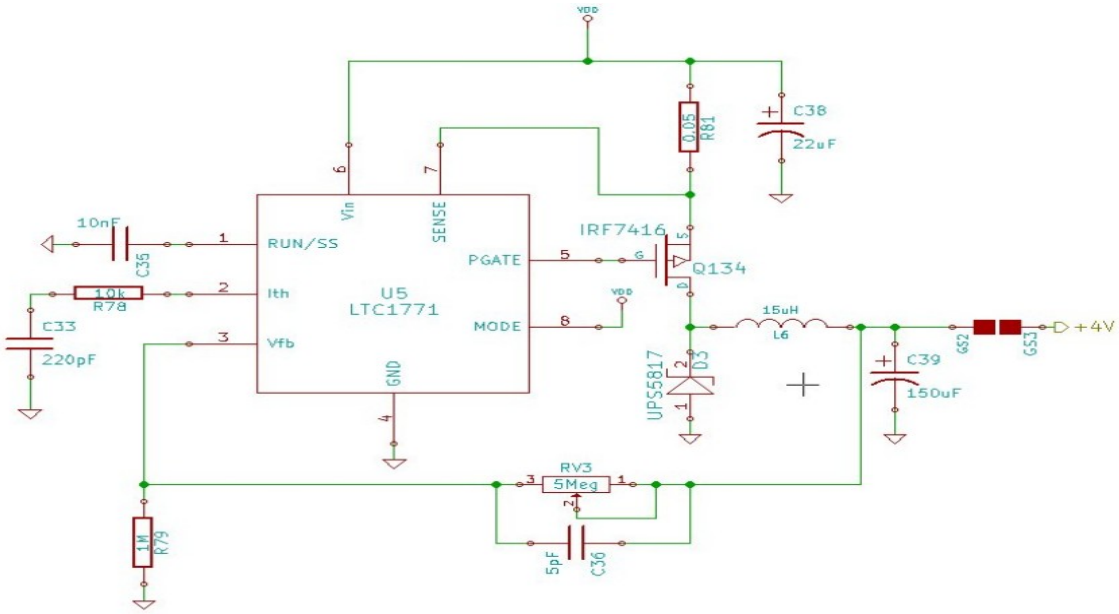


Abbildung 39: LTC1771 Schaltplan

Nachfolgend die zweite teil des Leistungsschemas :

Die Komponente LTC 1771 ist ein „High Efficient Step-Down“ DC/DC Wandler.

1. RUN/SS : Diese Pin kontrolliert die « Shutdown/Run » Mode. Wenn das mit einem extern Kondensator verbindet wird, bedeutet das « Soft Start » Mode.
2. I_{TH} : Diese Pin wird benutzt um ein Verstärkungsfehler zu korrigieren. Nicht benutzt in unserem Fall, also verbindet mit einem Kondensator und einem Widerstand.
3. V_{FB} : „Feedback“ von der Ausgangsspannung. Es gibt eine Vergleichung zwischen diese Pin und einen intern 1,23V Referenz. Die Formel um dem Widerstandsdivisor zu entscheiden wird nachfolgend beschreiben.

$$V_{OUT} = 1.23\left(1+\frac{R2}{R1}\right)$$

   So der „Feedback“ Eingang muss 1,23V sein. In unserem Fall haben R2 (R79 gleich 1MΩ und R2 wird von dem Trimmer (RV3) dargestellt. Mit diesem Montage kann man ein Ausgangsspannung zwischen 0V und 7.38V. Es gibt mit diesem Technik mehr Spannungspräzision.
4. GND : Ground Pin
5. PGATE : Das ist der „Gate“ Pin für dem P-MOSFET Switch. Der Ausgangsspannung ist zwischen 0V und Vin.
6. Vin : Eingangspannung. In unserem Fall ist diese Spannung 9V.
7. SENSE : Sinn Eingang um der Strom für der Switch zu kontrollieren. Es gibt einen Widerstand zwischen Vin und diese Pin um der Strom zu messen.
8. MODE : Burst Mode Operation. Wenn der Pin mit Vdd verbindet wird, wird diese Mode aktiviert.

Zwei Parametern müssen entscheidet werden. Der Widerstand R_{SENSE} (R81) und der Induktivität L6.

Der Formel für R_{SENSE} ist $R_{SENSE} = 100mV / I_{max}$.

Der Formel für L6 ist $Lmin = (75uH)(V_{OUT} + V_D)(R_{SENSE})$

Der Resultat für die drei Spannungen wird nachfolgend gegeben:

Spannung	RV3	Imax	RSENSE	L6
6V	3.9 MOhm	3 A	33 mOhm	16 uH
4V	2.25 MOhm	2 A	50 mOhm	17 uH
2V	626 KOhm	1 A	100 mOhm	19 uH

10.3.2 Ethernet Schnittstelle

Der Chip STE100p ist der physikalische Schicht von der Soundkarte. Der Partikularität dieses Chips ist seine Geschwindigkeit und der „High-Performance“. STE100P kann für Anwendungen mit 10BASE-T (Standard 10Mbit/s) und 100BASE-TX(Standard 100Mbit/s) benutzen.

Der Chip beistellt die MII Schnittstelle (Media Access Controllers) um zwischen der 10/100 MAC Schicht von dem Mikrocontroller und der physikalische Schicht zu kommunizieren.

Der STE100P unterstützt die Beide „half-duplex“ (nur eine Richtung zugleich) und „full-duplex“ (Die Beide Richtung zusammen) Operation. Der „Auto-Negotiation“ (Die Beide Geräte entscheidet die Parametern wie Geschwindigkeit, Duplex Mode, Flow Control), „Parallel Detection“ und „Manual Control“ Mode kann auch definiert werden.

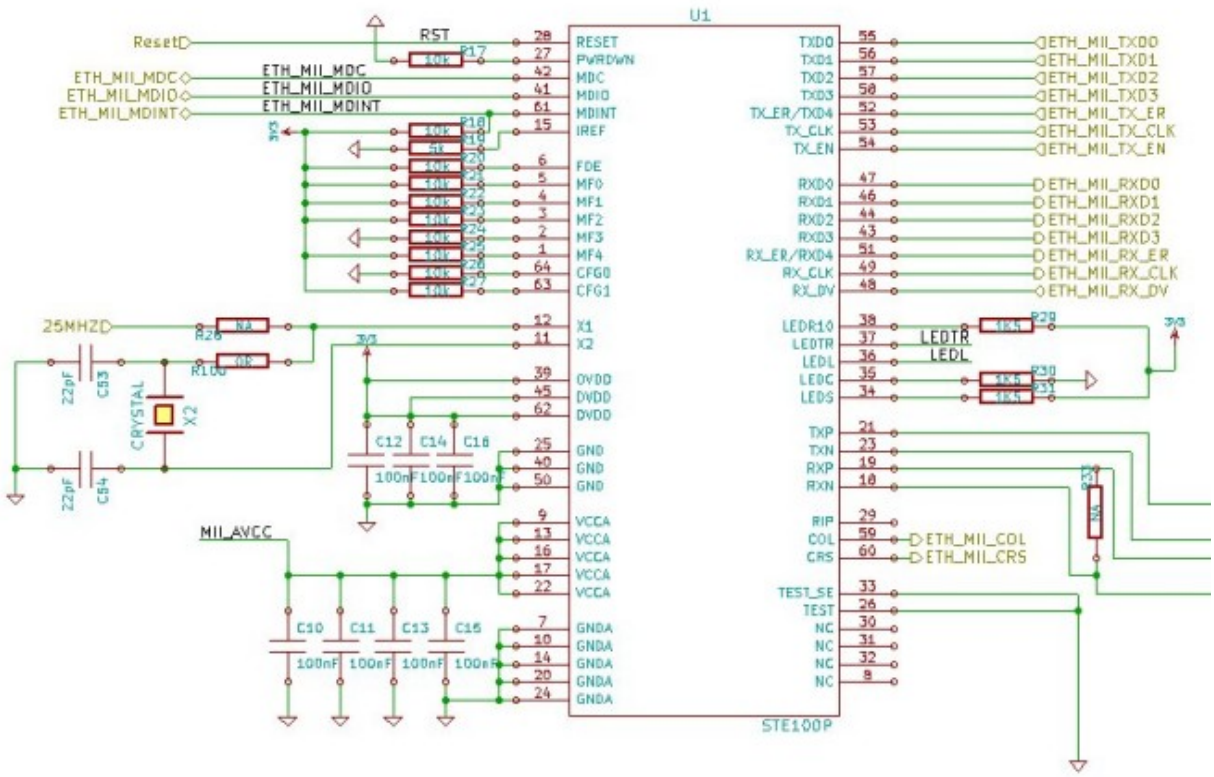


Abbildung 40: Ethernet Transceiver

- Management Data
  - MDC : Management Data Clock : Der Clock für der Control Schnittstelle (Serie Daten Kanale). Der maximal Frequenz ist 2,5 MHz.
  - MDIO : Management Data Input/Output : Bidirektional Serien Daten Kanale für PHY Kommunikation.

- MDINT : Management Data Interrupt : Unterbrechung für den Mikrocontroller. Nicht benutzt in unserem Fall.

- MII Tx
  - TxD0-TxD3 : Transmit Data : Der „Media Acces Controller“ (MAC) schaltet die Daten bis STE100P mit diesen Ausgängen durch. Diese Signalen wird von dem Mikrocontroller synchronisiert.
  - TxER : Transmit Coding Error : Der MAC Schicht vom Mikrocontroller benutzt diesen Eingang, wenn ein Fehler mit dem Transmit Daten eintretet. Wenn STE100P mit 100base-tx (100Mbps) funktioniert, antwortet der Chip mit eine invalide Code auf dieser Leitung.
  - TxCLK : Transmit Clock : Der Chip erschafft ein Clock für den Mikrocontroller. Der Frequenz ist 25MHz für 100Mbps Operation und 2.5MHz für 10Mbps Operation.
  - TxEN : Transmit Enable : Wenn der MAC Schicht Daten schickt, wird diese Signal aktiviert.
- MII Rx
  - RxD0-RxD3 : Receive Data : STE100P treibt die Daten für den Mikrocontroller auf diesem Pin. Synchronisiert mit RxCLK.
  - RxER : Receive Error : Information für den Mikrocontroller, wenn es ein Fehler auf dem Netzwerk gibt.
  - RxCLK : Receive Clock : Referenz Clock für alle Rx-Signal. 25MHz für 100Mbps Operation und 2.5MHz für 10Mbps Operation.
  - RxDV : Receive Data Valid : Diese Pin wird aktiviert, wenn STE100p Daten für den Mikrocontroller schickt.
- MII Control
  - COL : Collision Detected : Diese Pin wird aktiviert, wenn ein Kollision festgestellt wird. Diese Ausgang bleibt mit einer Hochstufe während der Kollision. Diese Signal ist asynchrone und nicht aktiv während eines Full-Duplex Betriebs.
  - CRS : Carrier Sense : Während eines Half-Duplex Betriebs wird diese Pin getrieben, wenn Tx oder Rx Medien „Non-Idle“ sind. Während eines Full-Duplex Betriebs wird diese Pin nur mit Rx funktioniert.
- LED :
  - LEDR10 : LED Ausgang für 10Mb/s Link Status.
  - LEDTR : LED Ausgang für Tx/Rx Aktivität Status.
  - LEDL : LED Ausgang für Link Status.
  - LEDC : LED Ausgang für Full-Duplex oder Kollision Status.
  - LEDS : LED Ausgang für 100Mb/s Link Status.
- Physical
  - TXN, TXP : Differenzial Tx Ausgang für 100Base-Tx und 10Base-T.
  - RXN, RXP : Differenzial Rx Eingang für 100Base-Tx und 10Base-T.
  - X1, X2 : Extern Clock von 25MHz.

10.3.3 Controller

Der Mikrocontroller, der benutzt wird, ist STM32F107 von STMicroelectronics. Diese Chip enthält ein Prozessor ARM 32-Bit Cortex-M3 mit einem maximalem Frequenz von 72MHz. Die wichtigsten Parametern von diesem Mikrocontroller sind die PWM-Ausgänge, nützlich für unsere Anwendung, und die Ethernet Kommunikation Schnittstelle (Beide Möglichkeiten 10 und 100 Mhz, SRAM und DMA) , die kompatibel mit dem MII Protokoll ist. Für unserem Anwendung braucht man nicht mehr als 64 Pins.

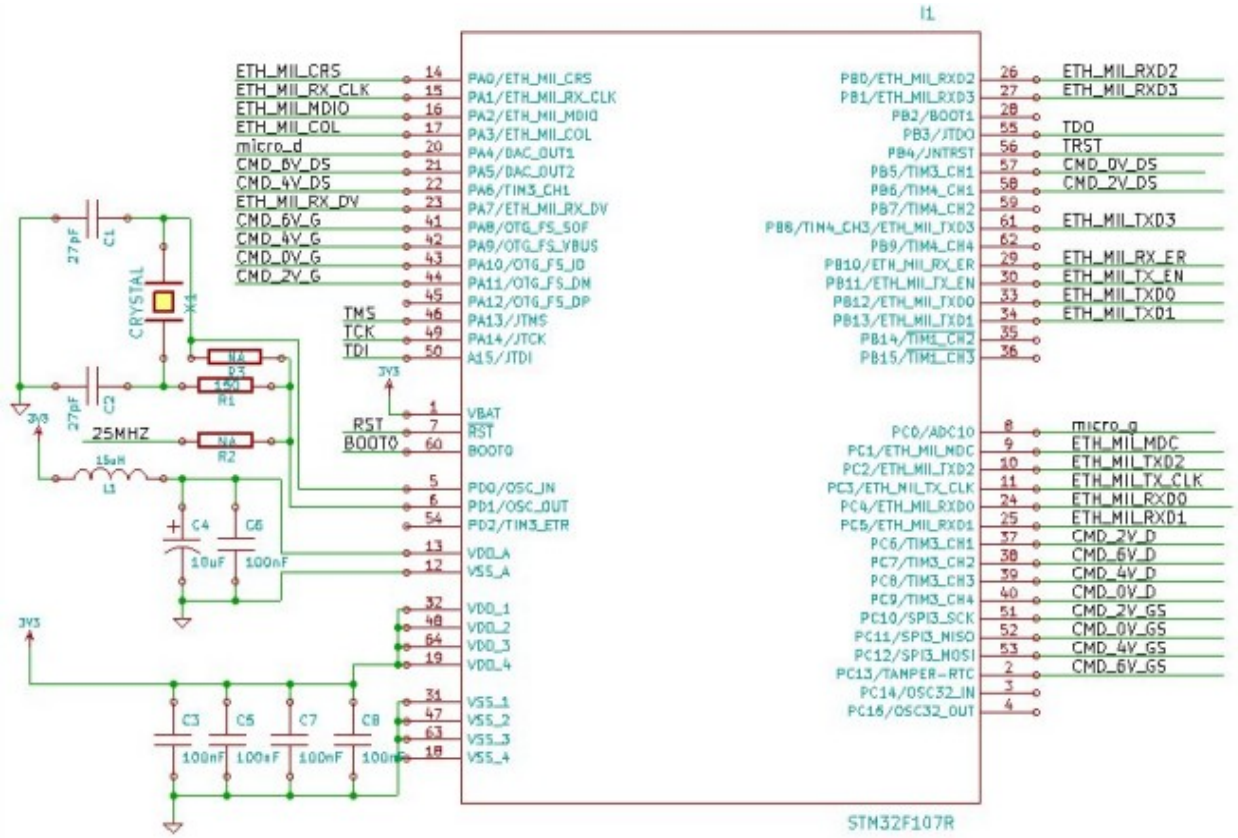


Abbildung 41: Controller Schaltplan

- JTAG : (Join Test Action Group) Das ist ein Kommunikationsstandart. Das ist ein Verfahren für den Test und Debug von elektronischer Hardware direkt in der Schaltung. Eine JTAG-Komponente besteht aus verschiedenen Teilen. Dem Test Access Port (TAP) mit den Steurleitungen, im allgemeinen auch JTAG-Port oder JTAG-Schnittstelle genannt, dem TAP-Controller, eine State-Machine, welche die Testlogik steuert und zwei Schieberegistern, dem « Instruction Register » (IR) und dem « Data Register » (DR).

- TMS : Test Mode Select Input : Diese Leitung bestimmt, in welchem folgenden State die State Machine des Test Access Port bei der nächsten positiven Signalflanke des TCK-Signals springt.
- TCK : Test Clock :Das Taktsignal für die gesamte Testlogik.
- TDI : Test Data Input :Serieller Eingang der Schieberegister in Mikrocontroller
- TDO : Test Data Output : Serieller Ausgang der Schieberegister
- TRST : Test Reset : Reset der Testlogik. Dieses Signal ist optional.

- Ethernet : Alle Ethernet Leitungen sind die gleichen als die Ein- Ausgänge vom STE100P. Sie haben auch die gleiche Funktionalität.
- Micro : Zwei Mikrophone Eingänge für Stereosignal sind bereit. Diese Eingänge muss mit dem ADC Wandler des Mikrocontrollers verbinden werden. Die zwei Pins sind ADC12_IN4 und ADC12_IN10.
- CMD : Das ist die Steuerung für den Ausgang mit MOSFET. Der Signaltyp ist PWM.
  - Mono : Für den erste Kanal kann man spezifische PWM-Ausgänge benutzen. Der Timer 1 und der Timer 3 vom Mikrocontroller haben vier Kanäle, die mit vier Pins verbindet sind.
  - Stereo : Für den zweite Kanal kann man nicht die PWM-Ausgänge benutzen, weil die Pins schon besetzt von der Ethernet Schnittstelle. So muss man GPIO (General Purpose Input/Output) benutzen und die Ausgänge mit einem spezifischem Algorithmus kontrollieren.
- Oscillator : Um das Clock zu generieren benutzt man ein extern Quartz Crystals mit einem Frequenz von 25 MHz.

10.3.4 Lautsprecher Verwaltung

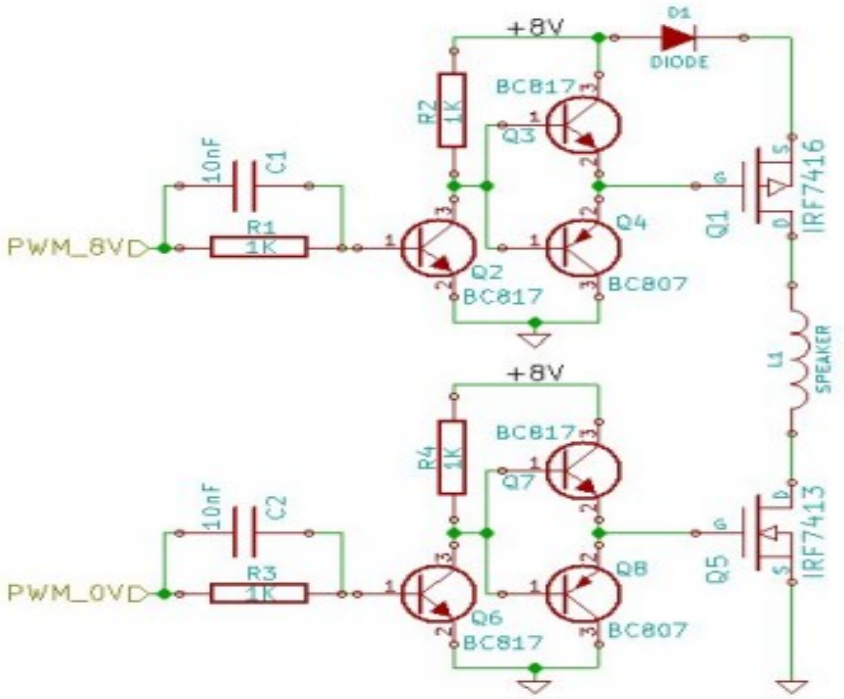


Abbildung 42: Lautsprecher Verwaltung

- Wenn der PWM Signal vom Mikrocontroller ist gleich '1' (3.3V), gibt es ein Strom durch die Base vom Transistor Q2 und kann der Transistor leiten. Der Spannung in dem Kollektor ist also 0V. Auf dem Transistor PNP Q4 gibt es jetzt einen Spannung Vbe = -0.7V und der Strom kann durch Q4 fließen. Der MOSFET-P Q1 hat eine Spannung Vgs = -6V und der Strom kann durch den Lautsprecher fließen.
- Wenn der PWM Signal ist gleich '0' (0V), gibt es kein Strom in der Base vom Q2. Also kann der Strom nicht fließen und die Kollektor Spannung ist 6V. In der Base von Q3 fließt ein Strom und der Transistor kann leiten. Die „Gate“ Spannung von Q1 ist 6V und Vgs = 0V (mit dem Diode ist die Spannung Vgs = 0.7V) und der Strom kann nicht durch dem Lautsprecher fließen.
- Der Transistor Q2 muss gesättigt sein. Für eine gesättigte Mode ist der Kollektor Strom Ic = Vcc/R2 = 6V/1KOhm = 6mA. Mit dem Parameter h_{fe} (Datenblatt von BC817-40) gleich 250 (minimal Wert) kann man der Strom I_b kalkulieren : $I_b = I_c / h_{fe}$ = 6mA / 250 = 24uA. Um sicher zu sein, daß der Transistor gesättigt ist, muss man mindestens ein Base-Strom von 10 mal höher haben. So Ib muss größer als 240uA sein.

Rb = (3.3V – Vbe) / Ib = 2.6V / 240µA = 10.8 KOhm

VCC	Rc	Ib	Rb
2V	1KOhm	80uA	32.5 KOhm
4V	1KOhm	160uA	16.25 KOhm
6V	1KOhm	240uA	10.8 KOhm

- Es gibt ein Kondensator in Parallel von Rb, weil die Übergänge zu schnell sind.

Hier ist der Lautsprecher-Ausgang Schaltplan.

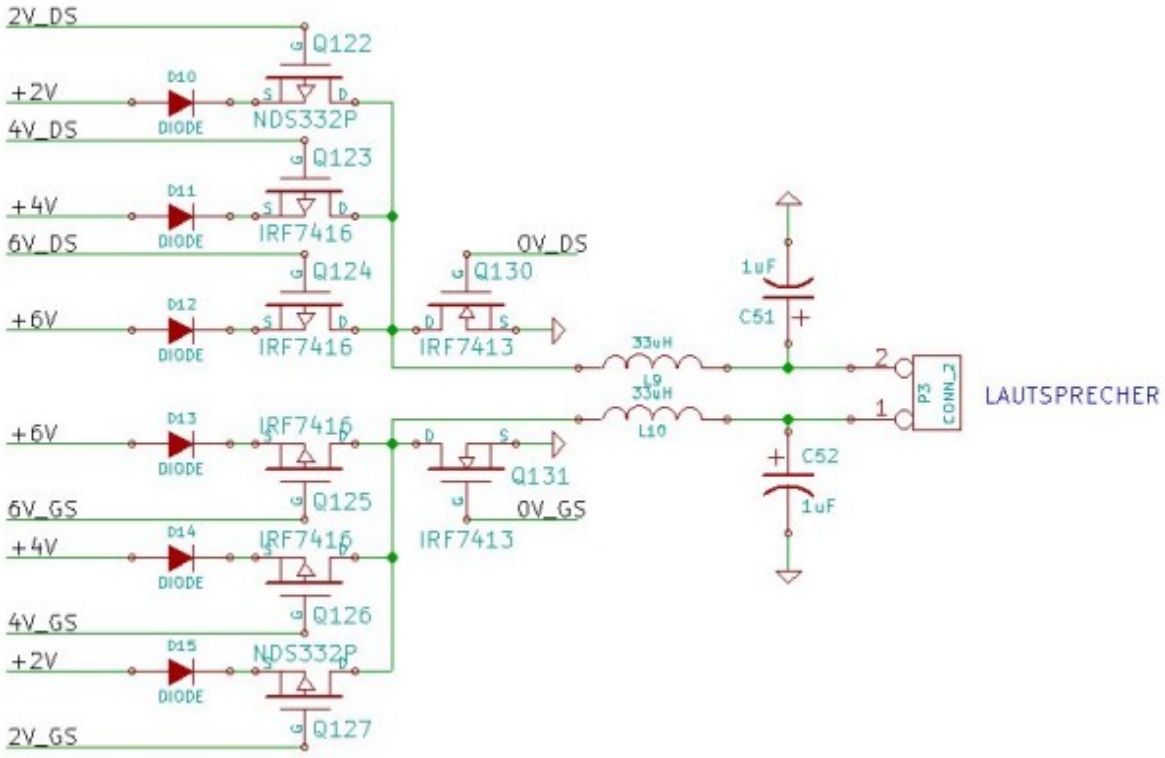


Abbildung 43: Lautsprechersausgang Schaltplan

- Mit den 8 Kommandos kann man entscheiden in welchem Richtung fließt der Strom durch den Lautsprecher und mit welchem Intensität. Zum Beispiel für +2V kann man der Signal 2V_GS und 0V_DS aktivieren. Für -4V kann man der Signal 4V_DS und 0V_GS aktivieren.
- Es gibt eine Diode auf alle Source Pin vom MOSFET-P um alle Störungen auf der Spannungen zu vermeiden.
- Es gibt im Ausgang ein Filter mit einem Induktivität und einem Kondensator um eine sauberer Signal zu haben.

10.3.5 Mikrophone

Der Eingang Signal, der von einem Mikrophone kommt, kann positiv oder negativ sein. Der Mikrocontroller kann nicht diese Signal handeln. So braucht man einen kleinen Adaptierung-Kreis.

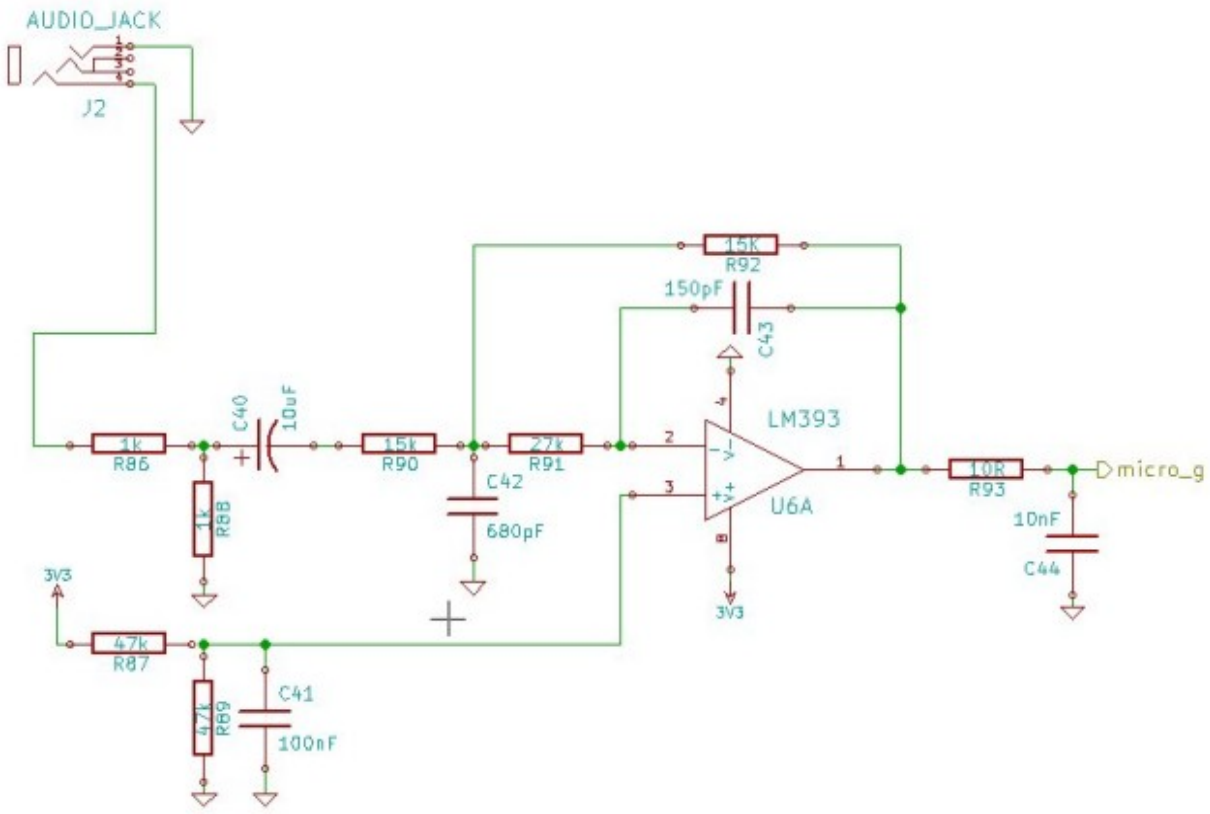


Abbildung 44: Mikrophone Schaltplan

Es gibt auf diesem Schaltplan ein Verstärker mit ein Filter (Fc = 22KHz). Es gibt auch ein Offset von Vcc/2 um der Signal für den Mikrocontroller zu adaptieren.

Diese Schaltplan kommt von einem anderem Projekt, hat schon getestet geworden und funktioniert.

Aber für diese Projekt braucht man nicht diese Eingänge.

10.3.6 Layout

Nachfolgend gibt es ein Bild mit dem Layout :

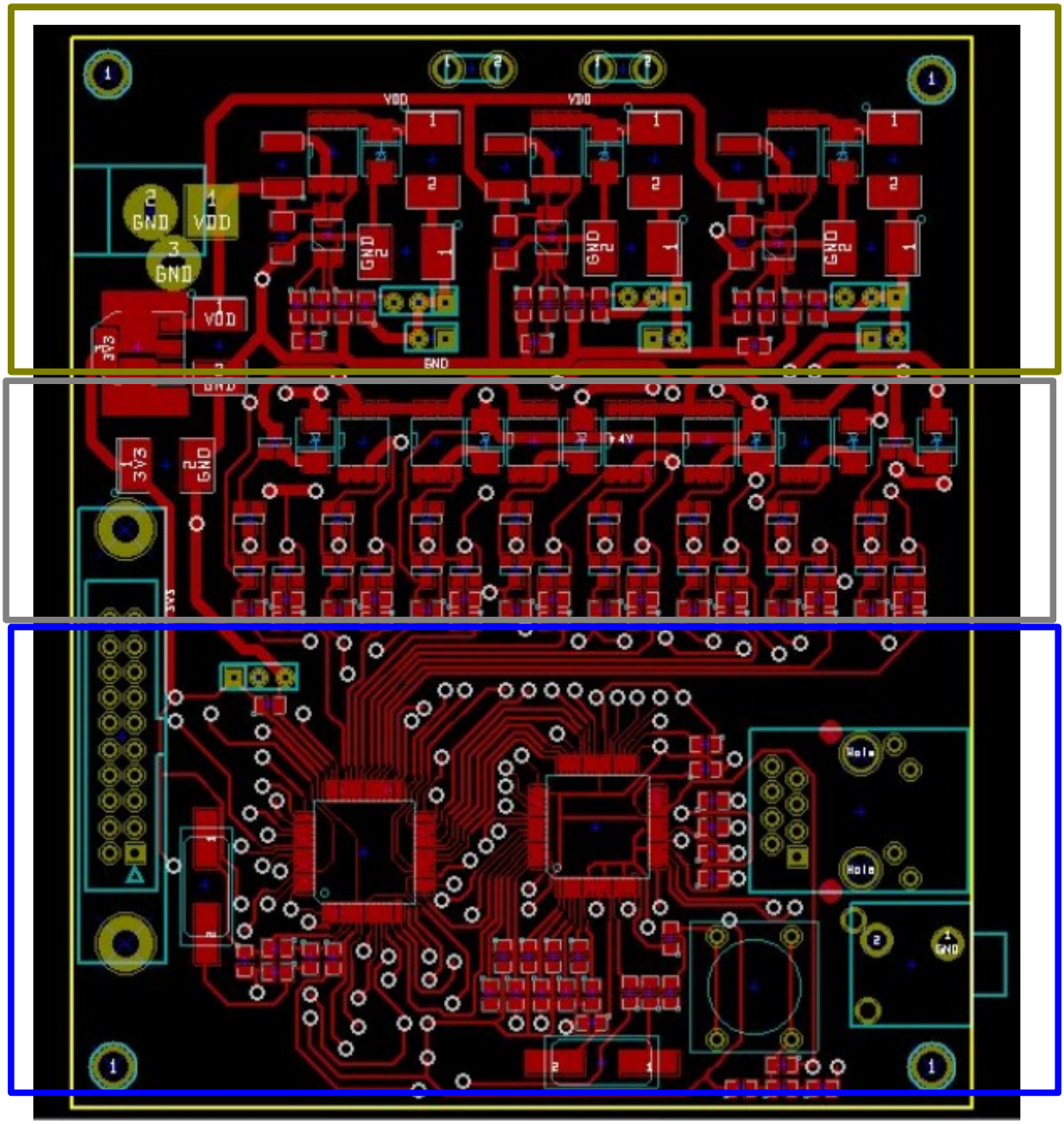


Abbildung 45: Layout

Das Layout enthält drei verschiedene Teile. Oben gibt es den Leistungsteil (braun) mit drei „DC/DC“ Wandlern. Es gibt ebenso die zwei Lautsprecher-Ausgänge ganz oben. Das Layout wird gebaut, damit ein Strom von 3A max. fließen kann. Also müssen die Pisten 1mm dick sein.

Der zweite Teil (grau) enthält das „Speaker Management“ mit den Treibern und den Mostfetstransitoren. Der Teil für den zweiten Kanal wird unten gebaut.

Der letzte Teil (blau) enthält den Mikrocontroller und den Ethernet-Controller.

10.4 NEUE SOFTWARE

Mit diesen neuen Ausgängen (3 positive & 3 negative Spannungen) muss man die Software modifizieren.

Vom Rechner bekommt man Samples auf 16bit und „Signed“. Also gibt es einen Wert zwischen -32768 und +32768. Die Ausgangsmöglichkeiten sind 2V, 4V oder 8V und immer eine positive und eine negative Spannung.

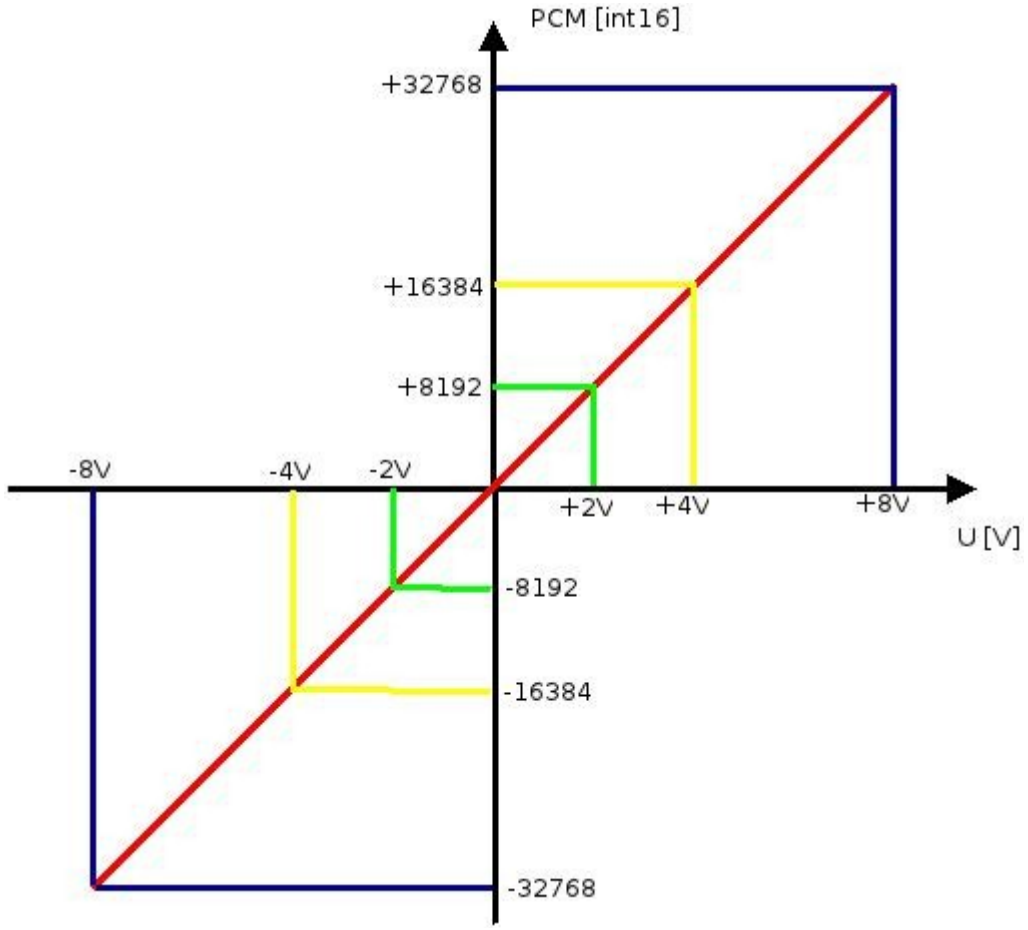


Abbildung 46: Neue Software Verwaltung

Mit diesem Beispiel hat man für alle negativen Werte die Wahl verschiedener PWM-Spannungen (-8V, -4V, -2V).

- Für einen PCM Wert zwischen -32768 und -16384 kann man den -8V PWM-Ausgang benutzen. Der PWM-Wert kann nur zwischen 50% und 100% sein.
- Für ein PCM Wert zwischen -16384 und -8192 kann man den -4V PWM-Ausgang benutzen. Der PWM-Wert kann nur zwischen 50% und 100% sein.
- Für ein PCM Wert zwischen -8192 und 0 kann man den -2V PWM-Ausgang benutzen. So benutzt man der PWM zwischen 0 und 100%.

Das System bleibt das gleiche mit positiven Werten.

10.5 TEST

Für diesen Test hat man ein Testsignal Dreieck benutzt. Dieses Signal hat ein Wert zwischen -32000 und +32000 und der Ausgang ist PWM mit 3 Spannungsebenen.

Mit diesem Test kann man die Präzision der Signale sehen.

Das Resultat auf dem Lautsprecher wird nachfolgend beschrieben.

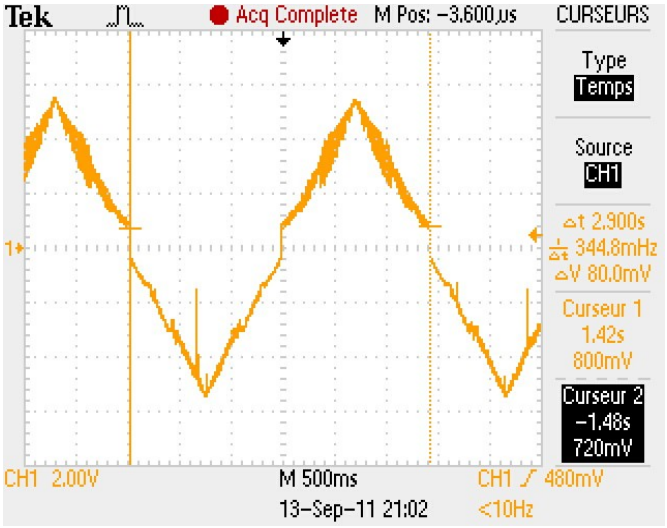


Abbildung 47: Resultat Dreieck

Man kann die 3 Spannungsübergänge sehen.

Unten gibt es den positiven PWM-Ausgang :

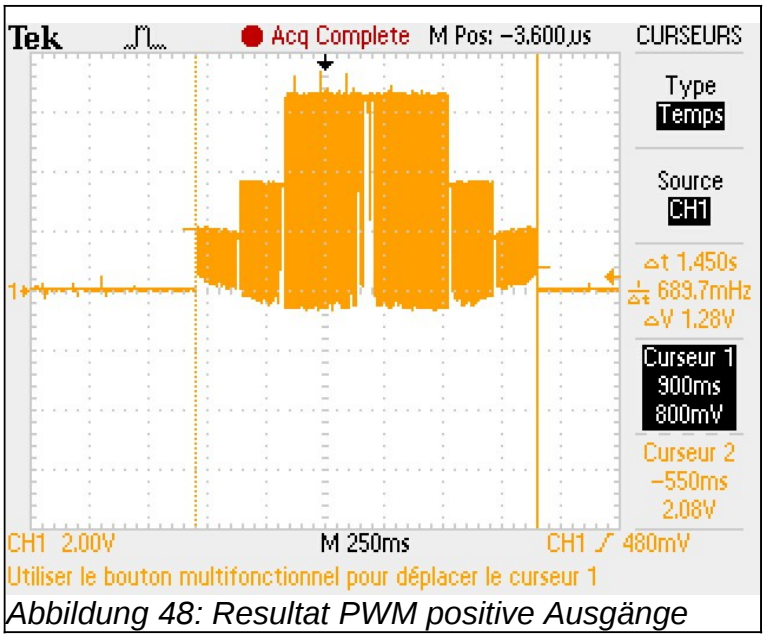


Abbildung 48: Resultat PWM positive Ausgänge

Man kann die 3 Spannungen sehen (2V, 4V und 6V in diesem Fall).

11 ZUSAMMENFASSUNG

Zuerst war der generelle Ablauf des Projekts sehr gut. Die erwünschten Ziele wurden alle erreicht und schlussendlich haben wir ein fertiges und funktionelles Produkt erhalten.

Bei der Entwicklung des Treibers, waren alle Resultate ziemlich befriedigend, aber leider nicht vollständig. Das Prinzip, die Informationen von der Soundarchitektur ALSA aufzunehmen und diese dann via einem anderen Treiber, der irgendeine Netzwerkkarte benutzt, zu verschicken ist erreicht, da die Daten auf der Soundkarte ankommen sind und gespielt werden können.

Ein negativer Punkt ist der Zeitmangel um die „**Capture**“ für den Soundempfang von der Soundkarte zu implementieren. Dieser Punkt ist nicht so wichtig, weil das Hauptziel war, den „Playback“ Modus zu implementieren. Ein anderer enttäuschender Punkt ist, dass das „Phonon“ System vom KDE Büro nicht den Soundkartentyp erkennt. Dies ist wahrscheinlich nicht wichtig, weil der Treiber mit ALSA richtig funktioniert.

Es geht sicherlich um einen Problem mit „Phonon“ und um die Aufnahme des Treibers in der Kernel-Linux Architektur. Der Treiber hingegen verwaltet den Audio-Soundfluss über Ethernet recht gut.

Was die « Firmware » für die Soundkarte selbst betrifft, sind die Ergebnisse sehr erfreulich. Die Schicht TCP/IP geht bestens mit den Angaben um und sendet sie korrekt an die Anwendung. Anschliessend, verwaltet dieselbe Anwendung recht gut das Implementierprotokoll für den Tonversand über TCP/IP. Hinzu kommt, dass die « Firmware » mit zwei verschiedenen Karten kompatibel ist, auf einfachen Wechsel einer Definition hin : mit der Karte OLIMEX, mit der die ersten Tests gemacht wurden, sowie mit der, speziell für dieses spezifische Projekt kreierten Karte.

Die Ausgänge Audio-PWM funktionieren gut und das Timing wurde respektiert, womit ein Ton mit einer maximalen Frequenz von 96 kHz erhalten werden konnte. Jetzt sind, wie beim Driver, die beiden Audio-Eingänge eines Mikrophons zu testen und zu implementieren. Dies sollte nicht zu schwierig sein.

Die Ausarbeitung der Hardware hat sehr viel Zeit gebraucht. Zuerst musste das Schema konzepiert werden. Dafür habe ich Beispiel genommen an der Entwicklung der Karte OLIMEX, die schon mit dem implementierten Programm kompatibel war. Anschliessend wurden aber verschiedene Änderungen vorgenommen, wie die Wahl eines Mikrokontrollers vom gleichen Typ, nur kleiner in Bezug auf unseren Bedarf (Eingang/Ausgang).

Dann war der Ethernet-Controller veraltet und nicht mehr vorhanden. Folglich musste ein Contoller gefunden werden, der ähnlich war, wie der, der mit der OLIMEX Karte gebraucht wurde. Leider war das Kommunikationsprotokoll, das von diesem neuen Chip benutz wurde, verschieden vom ersten (RMII für die Karte OLIMEX und nur MII für den neuen Chip). Also mussten sehr viele Elemente im Schema geändert werden und eine Lösung für einen kleineren Microcontroller gefunden werden.

Der Ausgang über die Lautsprecher war eine noch unentwickelte Neuheit. Aus diesem Grunde mussten zuerst Durchführbarkeitsstests gemacht werden. Anschliessend wurden einige, kleinere Probleme entdeckt und korrigiert, um die Signalsqualität beim Ausgang zu verbessern. Ebenso wurde ein kleines Problem in der Versorgung entdeckt. Sobald man zu viel Strom über die Spannungen zieht, können die Umsetzer DC/DC keinen Strom mehr liefern. Dies liegt sicherlich an der Qualität der Karte und an der Dimensionierung der Resistenz, die nicht angepasst war.

Aus Zeitmangel wurde diese Änderung nur auf dem Schema vorgenommen und nicht auf der Karte selbst. Im Gebiet Ethernet, gab es einige Probleme mit dem Verbindungsstück, das nicht an die Anwendung angepasst war. Im Ganzen funktioniert die Platine aber und gibt einen relativ korrekten Ton her, der mit einer leistungsfähigeren Platine und Steuerung verbessert werden kann.

Was meine persönliche Erfahrung betrifft, konnte ich viele wichtige Elemente kennen lernen. Linux war mir zu Beginn dieses Projektes völlig unbekannt. Heute kann ich sagen, dass ich die Grundlagen dieses Systems, sowie die vielgebrauchten Steuerungen beherrsche. Hinzu kommt, dass ich die Konzeption von Makefile und den Script für Linux beobachten und lernen konnte, was mir in den kommenden Jahr von grossem Nutzen sein wird.

Ebenso konnte ich meine Kenntnisse von Ethernet und TCP/IP verbessern. Schliesslich war die Konzeption der Karte, die in diesem Projekt ein grosses Plus war, sehr interessant, da die Entwicklung eines solchen Systems noch nie ausgeführt wurde.

Das Ziel dieses Projektes war nicht ein 100% funktionelles Produkt zu erhalten, sondern eine gute Basis für eine künftige und komplette Entwicklung eines Netzes von Tonkarten über Ethernet zu schaffen. Die Folge dieses Projektes wird bestimmt enorme Entwicklungsmöglichkeiten bieten.

12 AUSBLICK

12.1 HARDWAREVERBESSERUNG

12.1.1 Schaltplan

- Suche einen Mikrocontroller mit mehreren PWM-Ausgängen. Zum Beispiel STM32 mit 100 Pin. Zurzeit ist das Problem, dass der zweite Kanal nicht zur Verfügung steht, weil alle Ausgänge nicht PWM-kompatibel sind.
- Eine andere Möglichkeit mit den drei Spannungen ist eine Kontrolle der Power-Ausgängen mit dem Mikrocontroller.
- Man kann auch LED's und Switchs hinzufügen damit der Benutzer der Soundkarte mehr Kontrolle hat.

12.1.2 Layout

- Mit einem PWM-Ausgang gibt es viele Parasiten auf der Platine. Eine Lösung ist zwei Schichten hinzufügen, eine für GND und die andere für VCC, +8V, +4V und +2V.
- Während alle Tests gab es viele Modifikation auf der Platine. Für mehr Sicherheit muss man mehr Distanz zwischen den Bauelementen schaffen.
- Für eine bessere Zuverlässigkeit kann man die Stromleitungen dicker bauen.
- Für eine bessere Qualität braucht man eine professionelle Platine.

12.2 SOFTWAREVERBESSERUNG

12.2.1 Treiber

- Zurzeit funktioniert der Treiber nicht mit „Phonon“ das Programm von KDE Büros um den Sound zu verwalten. Also muss man eine Lösung finden.
- Jetzt wird nur „Playback“ Mode implementiert, aber man kann auch „Capture“ Modus (für ein Mikrophon) erschaffen.
- Modifizieren der Pufferverwaltung

12.2.2 Firmware

- Implementieren der Ausgänge für den zweiten Kanal
- Implementieren und testen der Mikrophoneingänge.
- Modifizieren des Algorithmus um die PWM-Ausgänge zu kontrollieren

13 REFERENZ

1. ***Linux Device Drivers, Third Edition*** (2005), Jonathan Corbet, Alessandro Rubini und Greg Kroah-Hartman
2. ***Linux-Treiber entwickeln*** (2011), Eva-Katharina Kunst + Jürgen Quade
3. ***SDB:Sound Concepts*** , OpenSuse.org, http://en.opensuse.org/SDB:Sound_concepts
4. ***Defeating the network security Infrastructure with Socat***, Philippe Bogaerts, http://www.radarhack.com
5. ***Writing an ALSA Driver*** (2002-2005), Takashi Iwai
6. ***Linux MAO / ALSA*** , http://www.linuxmao.org/tikiwiki/tiki-index.php?page=ALSA
7. ***socket programming in KERNEL (TCP/IP)***, http://www.linuxquestions.org/questions/programming-9/socket-programming-in-kernel-tcp-ip-403685/
8. ***Network Programming in the Kernel,*** http://www.scribd.com/doc/25456386/Network-Programming-in-the-Kernel
9. ***ALSA project - the C library reference: PCM (digital audio) interface,*** http://www.alsa-project.org/alsa-doc/alsa-lib/pcm.html
10. ***SOCAT,*** http://www.scribd.com/doc/51786985/45/SOCAT
11. ***System Management Guide : Communications and Networks : TCP/IP Protocols*** , http://ps-2.kev009.com:8081/wisclibrary/aix51/usr/share/man/info/en_US/a_doc_lib/aixbman/commadmn/tcp_protocols.htm
12. ***Ampli Classe D***, http://www.ziggysono.com/htm_effets/index.php?art=ampli2&titre=Amplis_part2
13. ***PWM***, http://www.arduino.cc/en/Tutorial/PWM

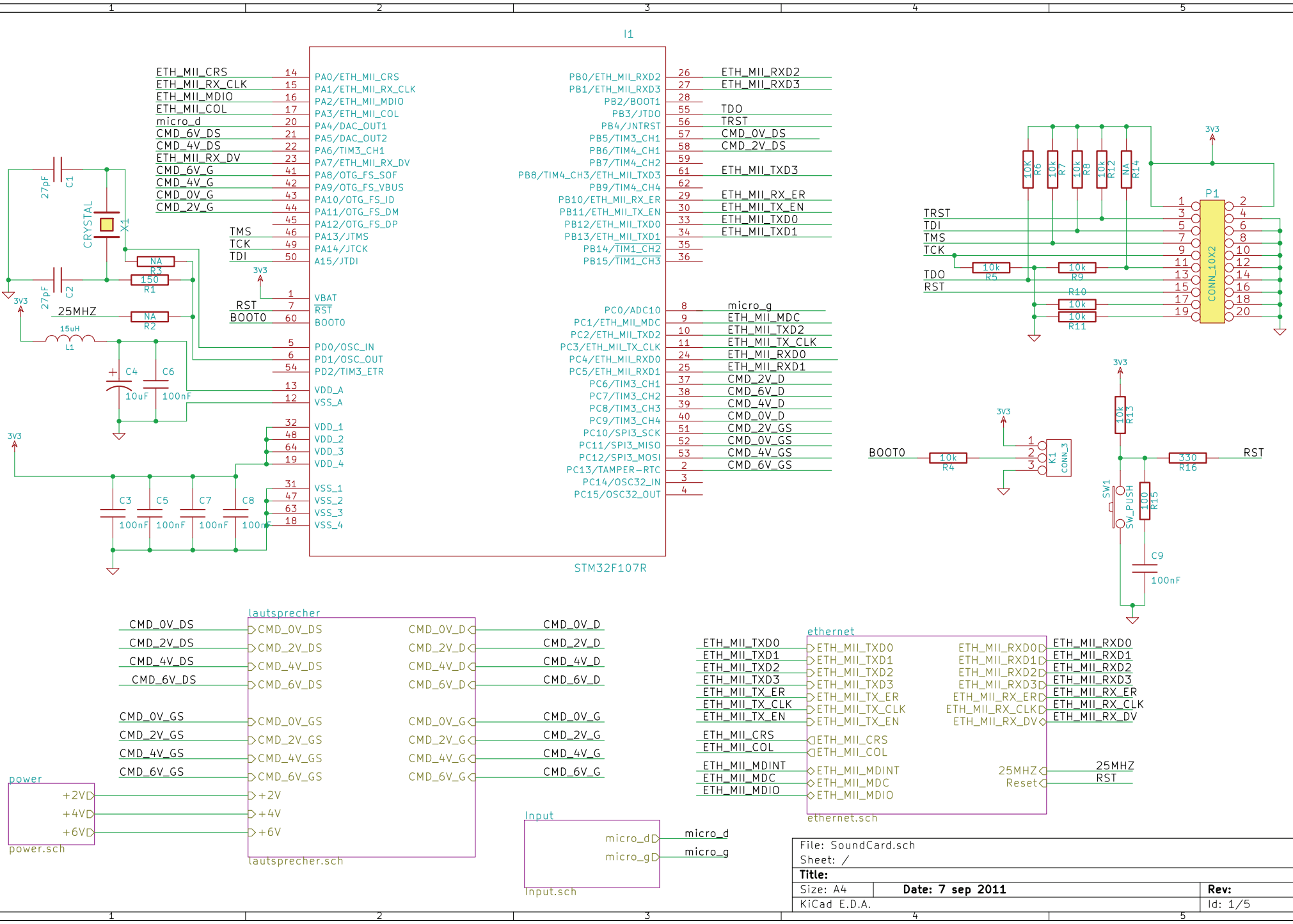
14 ANNEXES

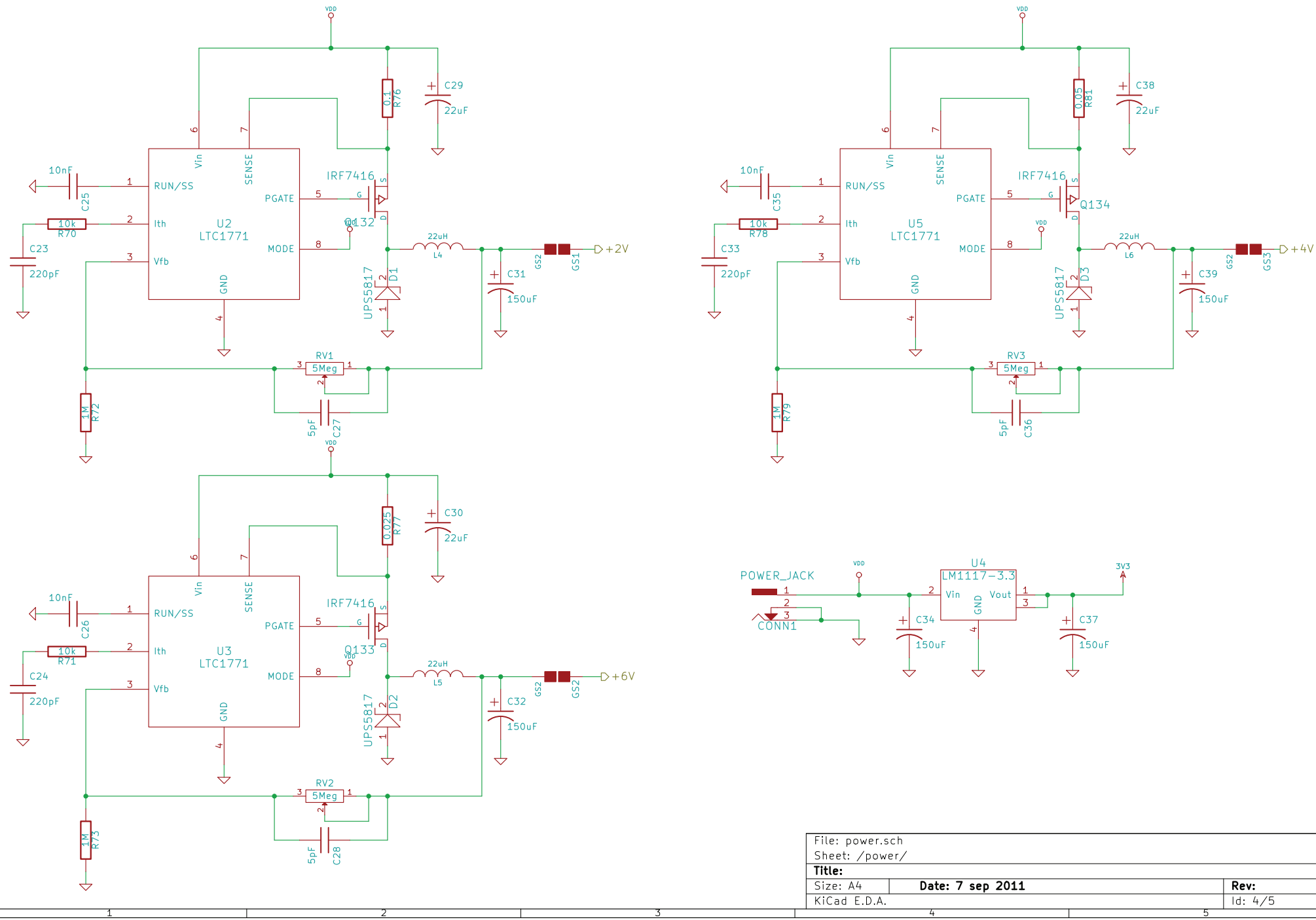
1. Schaltplan
2. Layout
3. Software Treiber
4. Firmware Soundkarte

ANHANG

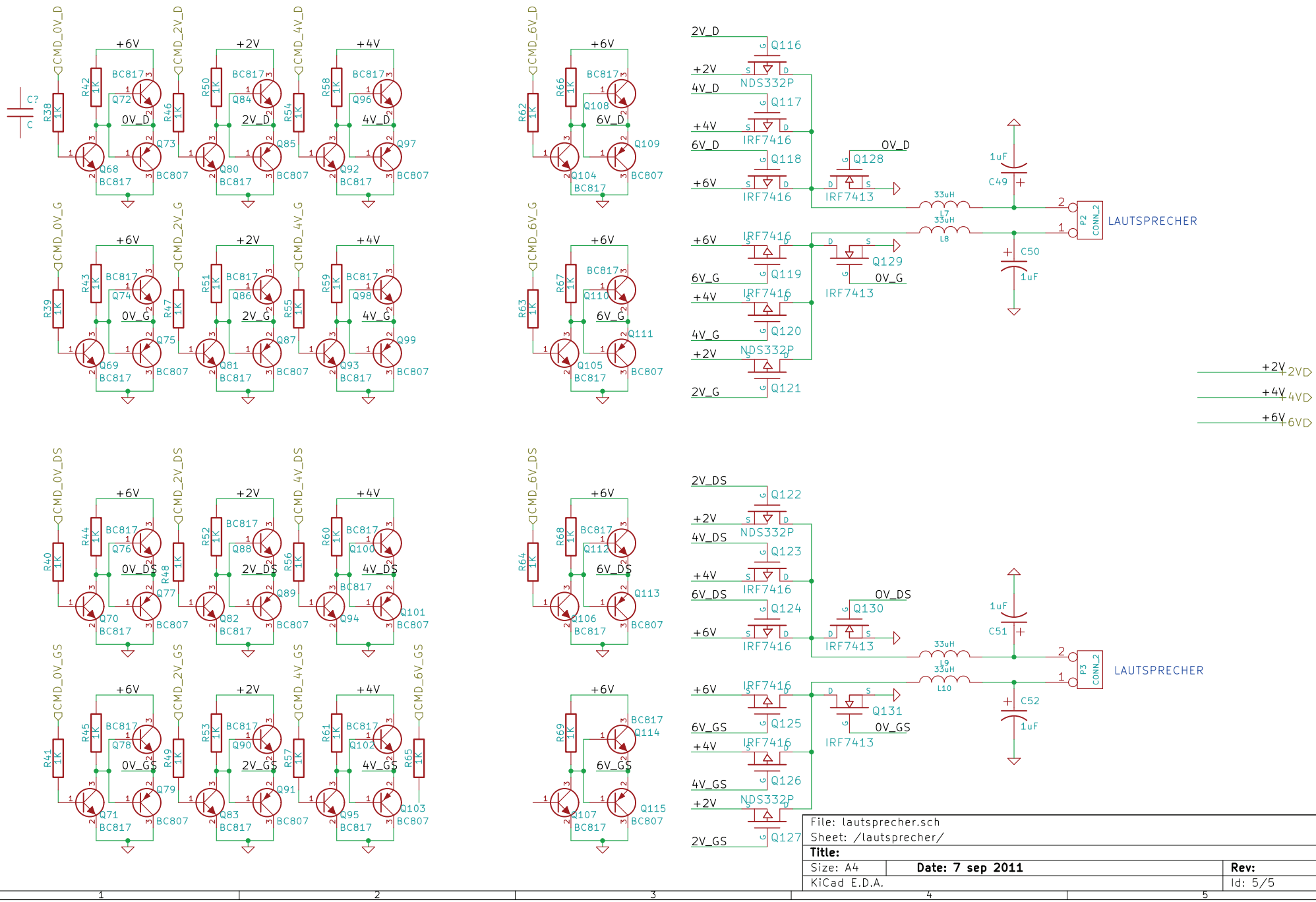
SCHALTPLAN

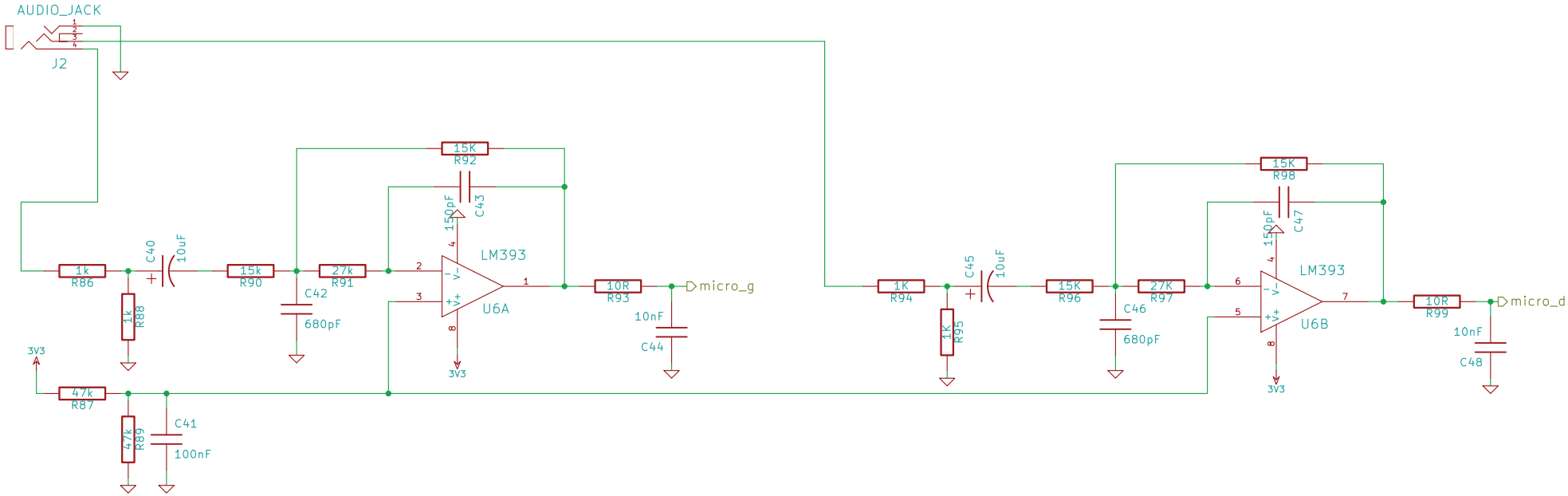
1. Soundkarte CONTROLLER
2. Soundkarte POWER
3. Soundkarte LAUTSPRECHER
4. Soundkarte INPUT
5. Soundkarte ETHERNET


U1
ETH_MII_CRS
ETH_MII_RX_CLK
ETH_MII_MDIO
ETH_MII_COL
micro_d
CMD_6V_DS
CMD_4V_DS
ETH_MII_RX_DV
CMD_6V_G
CMD_4V_G
CMD_0V_G
CMD_2V_G
TMS
TCK
TDI
RST
BOOT0
PA0/ETH_MII_CRS
PA1/ETH_MII_RX_CLK
PA2/ETH_MII_MDIO
PA3/ETH_MII_COL
PA4/DAC_OUT1
PA5/DAC_OUT2
PA6/TIM3_CH1
PA7/ETH_MII_RX_DV
PA8/OTG_FS_SOF
PA9/OTG_FS_VBUS
PA10/OTG_FS_ID
PA11/OTG_FS_DM
PA12/OTG_FS_DP
PA13/JTMS
PA14/JTCK
PA15/JTDI
VBAT
RST
BOOT0
PD0/OSC_IN
PD1/OSC_OUT
PD2/TIM3_ETR
VDD_A
VSS_A
VDD_1
VDD_2
VDD_3
VDD_4
VSS_1
VSS_2
VSS_3
VSS_4
PB0/ETH_MII_RXD2
PB1/ETH_MII_RXD3
PB2/BOOT1
PB3/JTDO
PB4/JNTRST
PB5/TIM3_CH1
PB6/TIM4_CH1
PB7/TIM4_CH2
PB8/TIM4_CH3/ETH_MII_TXD3
PB9/TIM4_CH4
PB10/ETH_MII_RX_ER
PB11/ETH_MII_TX_EN
PB12/ETH_MII_TXD0
PB13/ETH_MII_TXD1
PB14/TIM1_CH2
PB15/TIM1_CH3
PC0/ADC10
PC1/ETH_MII_MDC
PC2/ETH_MII_TXD2
PC3/ETH_MII_TX_CLK
PC4/ETH_MII_RXD0
PC5/ETH_MII_RXD1
PC6/TIM3_CH1
PC7/TIM3_CH2
PC8/TIM3_CH3
PC9/TIM3_CH4
PC10/SPI3_SCK
PC11/SPI3_MISO
PC12/SPI3_MOSI
PC13/TAMPER-RTC
PC14/OSC32_IN
PC15/OSC32_OUT
STM32F107R
ETH_MII_RXD2
ETH_MII_RXD3
TDO
TRST
CMD_0V_DS
CMD_2V_DS
ETH_MII_TXD3
ETH_MII_RX_ER
ETH_MII_TX_EN
ETH_MII_TXD0
ETH_MII_TXD1
micro_g
ETH_MII_MDC
ETH_MII_TXD2
ETH_MII_TX_CLK
ETH_MII_RXD0
ETH_MII_RXD1
CMD_2V_D
CMD_6V_D
CMD_4V_D
CMD_0V_D
CMD_2V_GS
CMD_0V_GS
CMD_4V_GS
CMD_6V_GS
C1
27pF
CRYSTAL
X1
C2
NA
R3
150
R1
25MHZ
NA
R2
L1
15uH
C4
10uF
C6
100nF
C3
C5
C7
C8
3V3
P1
CONN_10X2
R6
R7
R8
R12
R14
R5
R9
R10
R11
TRST
TDI
TMS
TCK
TDO
RST
K1
CONN_3
R4
R13
R16
SW1
SW_PUSH
R15
C9
lautsprecher
lautsprecher.sch
power
power.sch
+2V
+4V
+6V
input
input.sch
micro_d
micro_g
ethernet
ethernet.sch
ETH_MII_TXD0
ETH_MII_TXD1
ETH_MII_TXD2
ETH_MII_TXD3
ETH_MII_TX_ER
ETH_MII_TX_CLK
ETH_MII_TX_EN
ETH_MII_CRS
ETH_MII_COL
ETH_MII_MDINT
ETH_MII_MDC
ETH_MII_MDIO
ETH_MII_RXD0
ETH_MII_RXD1
ETH_MII_RXD2
ETH_MII_RXD3
ETH_MII_RX_ER
ETH_MII_RX_CLK
ETH_MII_RX_DV
25MHZ
Reset
RST
File: SoundCard.sch
Sheet: /
Title:
Size: A4
Date: 7 sep 2011
Rev:
KiCad E.D.A.
Id: 1/5

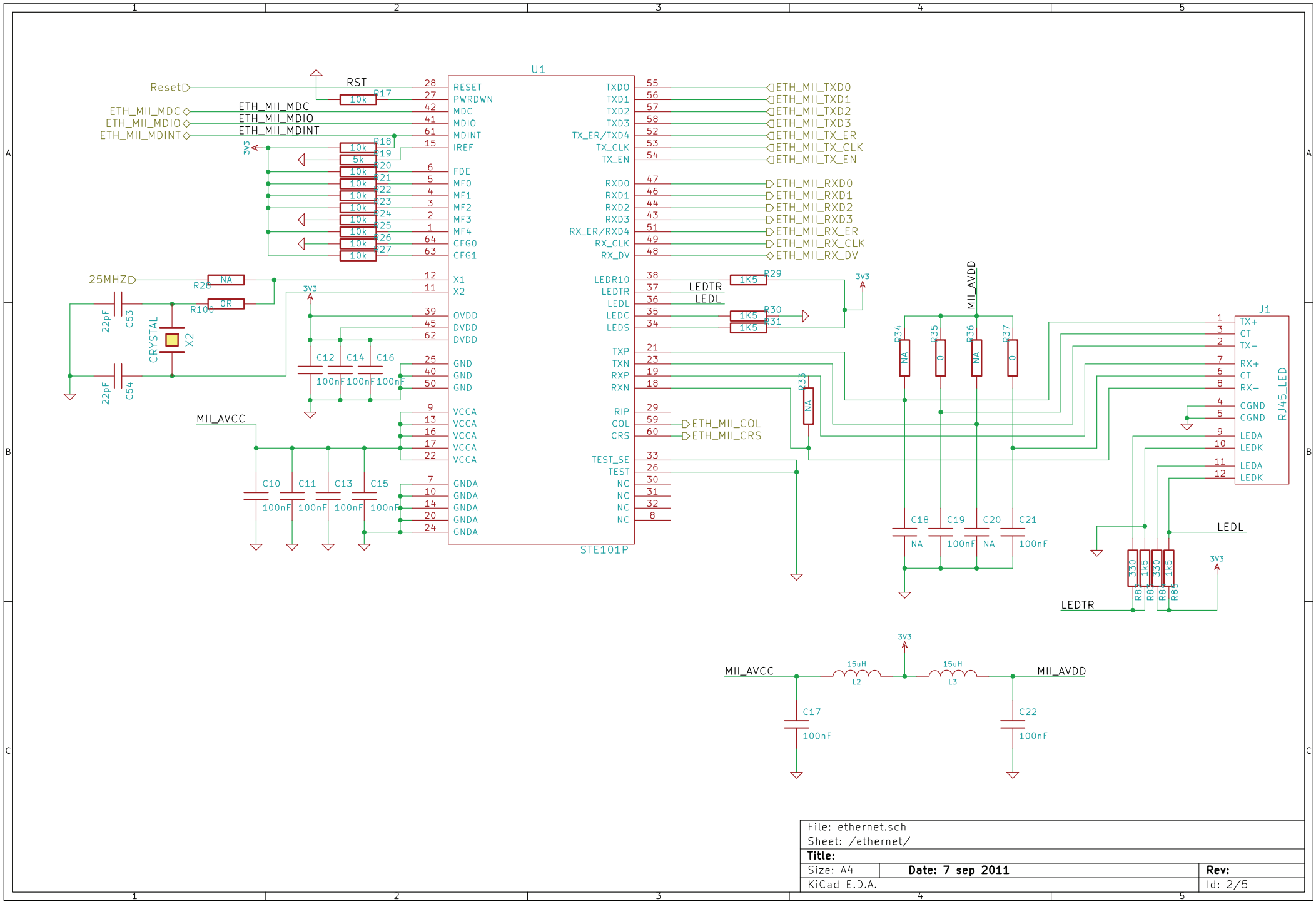


File: power.sch		
Sheet: /power/		
Title:		
Size: A4	**Date: 7 sep 2011**	**Rev:**
KiCad E.D.A.		Id: 4/5


CMD_0V_D
CMD_2V_D
CMD_4V_D
CMD_6V_D
CMD_0V_G
CMD_2V_G
CMD_4V_G
CMD_6V_G
CMD_0V_DS
CMD_2V_DS
CMD_4V_DS
CMD_6V_DS
CMD_0V_GS
CMD_2V_GS
CMD_4V_GS
CMD_6V_GS
+6V
+2V
+4V
0V_D
2V_D
4V_D
6V_D
0V_G
2V_G
4V_G
6V_G
0V_DS
2V_DS
4V_DS
6V_DS
0V_GS
2V_GS
4V_GS
6V_GS
BC817
BC807
NDS332P
IRF7416
IRF7413
33uH
1uF
LAUTSPRECHER
CONN_2
File: lautsprecher.sch
Sheet: /lautsprecher/
Title:
Size: A4
Date: 7 sep 2011
Rev:
KiCad E.D.A.
Id: 5/5

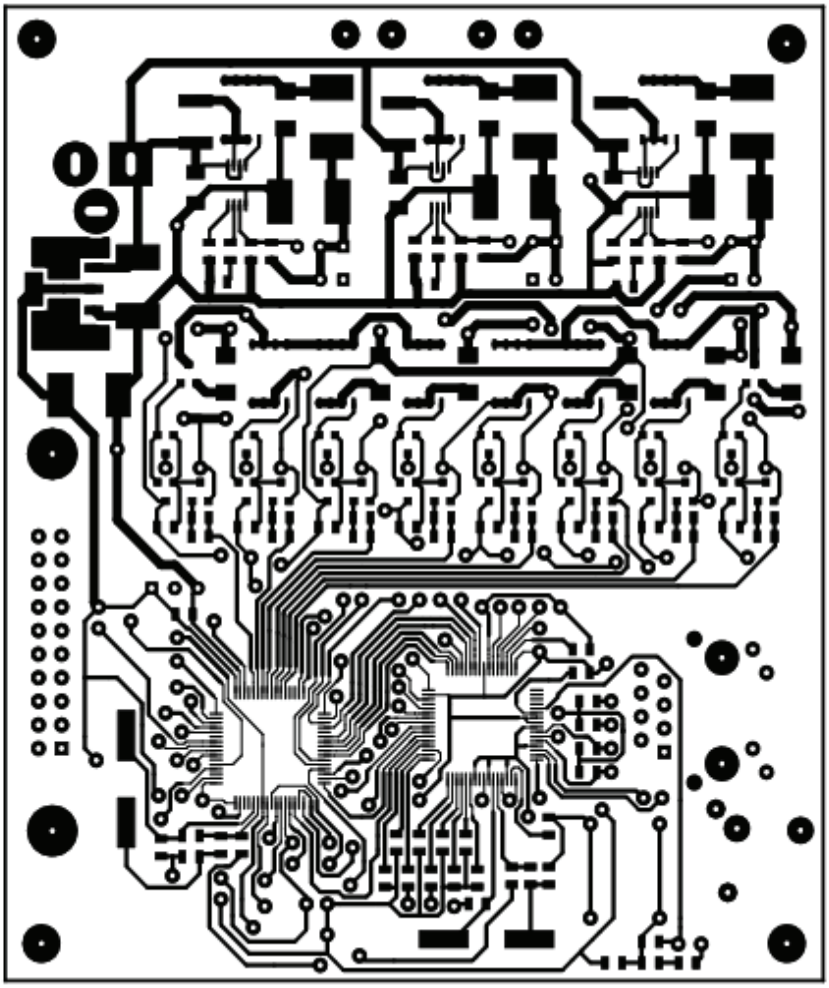


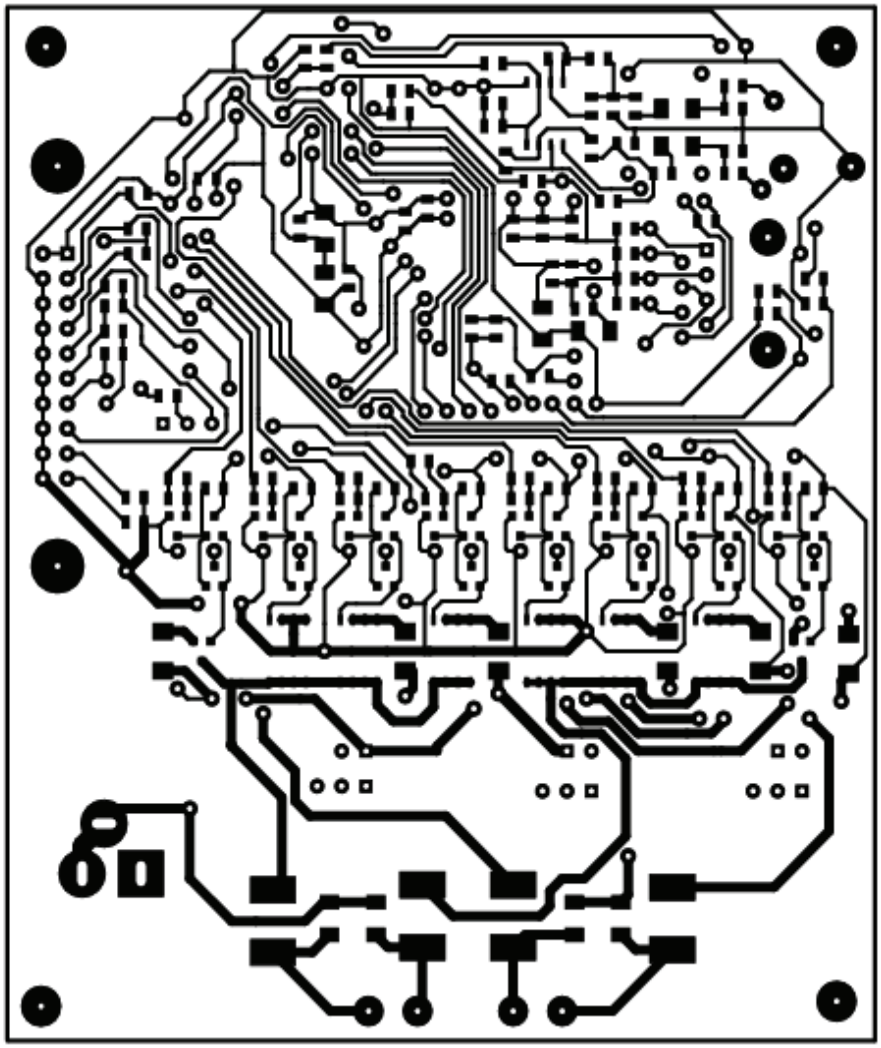
File: Input.sch		
Sheet: /Input/		
Title:		
Size: A4	**Date: 7 sep 2011**	**Rev:**
KiCad E.D.A.		Id: 3/5

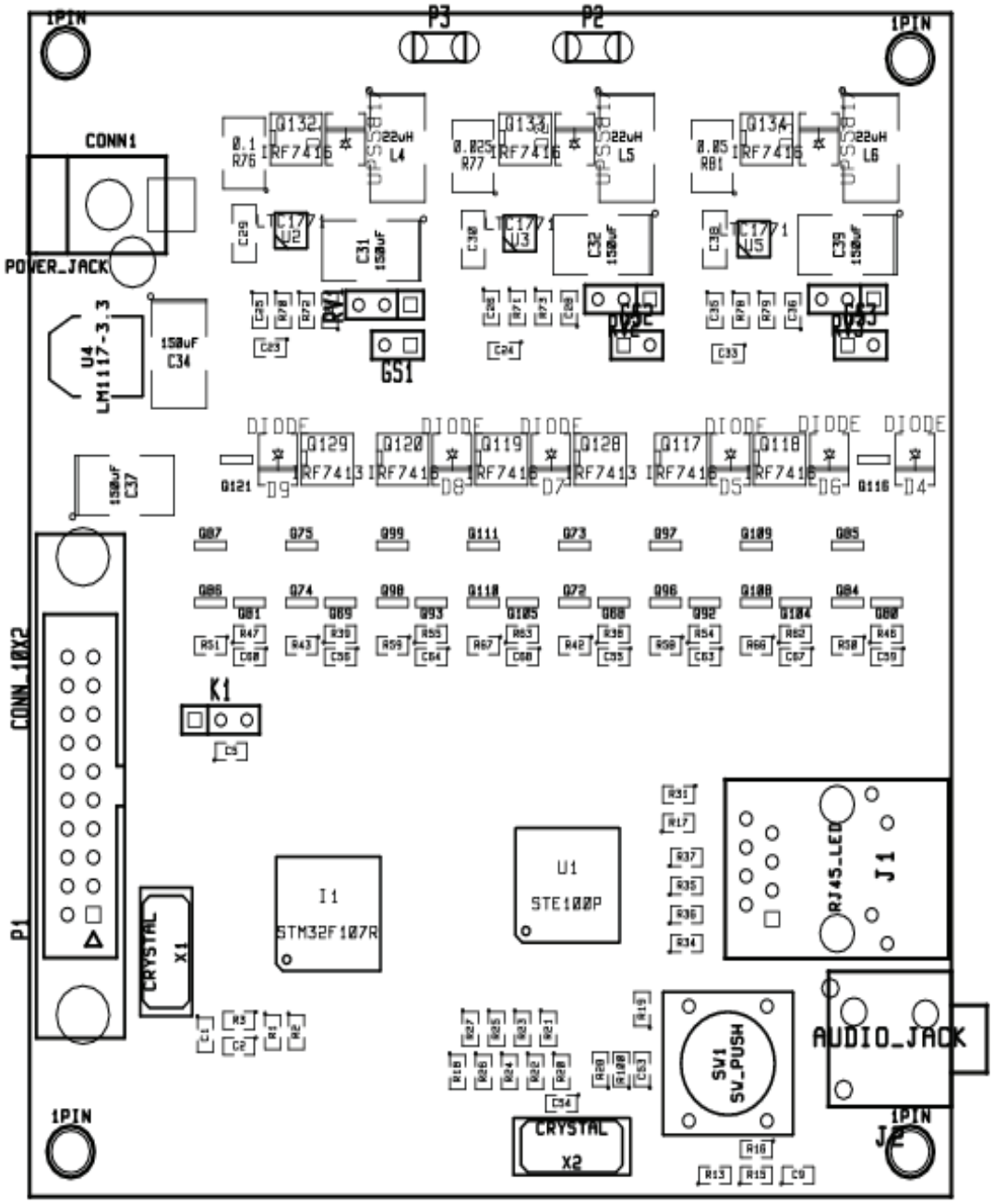

U1
STE101P
RESET
PWRDWN
MDC
MDIO
MDINT
IREF
FDE
MF0
MF1
MF2
MF3
MF4
CFG0
CFG1
X1
X2
OVDD
DVDD
DVDD
GND
GND
GND
VCCA
VCCA
VCCA
VCCA
VCCA
GNDA
GNDA
GNDA
GNDA
GNDA
TXD0
TXD1
TXD2
TXD3
TX_ER/TXD4
TX_CLK
TX_EN
RXD0
RXD1
RXD2
RXD3
RX_ER/RXD4
RX_CLK
RX_DV
LEDR10
LEDTR
LEDL
LEDC
LEDS
TXP
TXN
RXP
RXN
RIP
COL
CRS
TEST_SE
TEST
NC
NC
NC
NC
Reset
RST
ETH_MII_MDC
ETH_MII_MDIO
ETH_MII_MDINT
3V3
25MHZ
CRYSTAL
MII_AVCC
MII_AVDD
LEDTR
LEDL
ETH_MII_TXD0
ETH_MII_TXD1
ETH_MII_TXD2
ETH_MII_TXD3
ETH_MII_TX_ER
ETH_MII_TX_CLK
ETH_MII_TX_EN
ETH_MII_RXD0
ETH_MII_RXD1
ETH_MII_RXD2
ETH_MII_RXD3
ETH_MII_RX_ER
ETH_MII_RX_CLK
ETH_MII_RX_DV
ETH_MII_COL
ETH_MII_CRS
J1
RJ45_LED
TX+
CT
TX−
RX+
CT
RX−
CGND
CGND
LEDA
LEDK
LEDA
LEDK
15uH
L2
L3
C17
C22
100nF
File: ethernet.sch
Sheet: /ethernet/
Title:
Size: A4
Date: 7 sep 2011
Rev:
KiCad E.D.A.
Id: 2/5

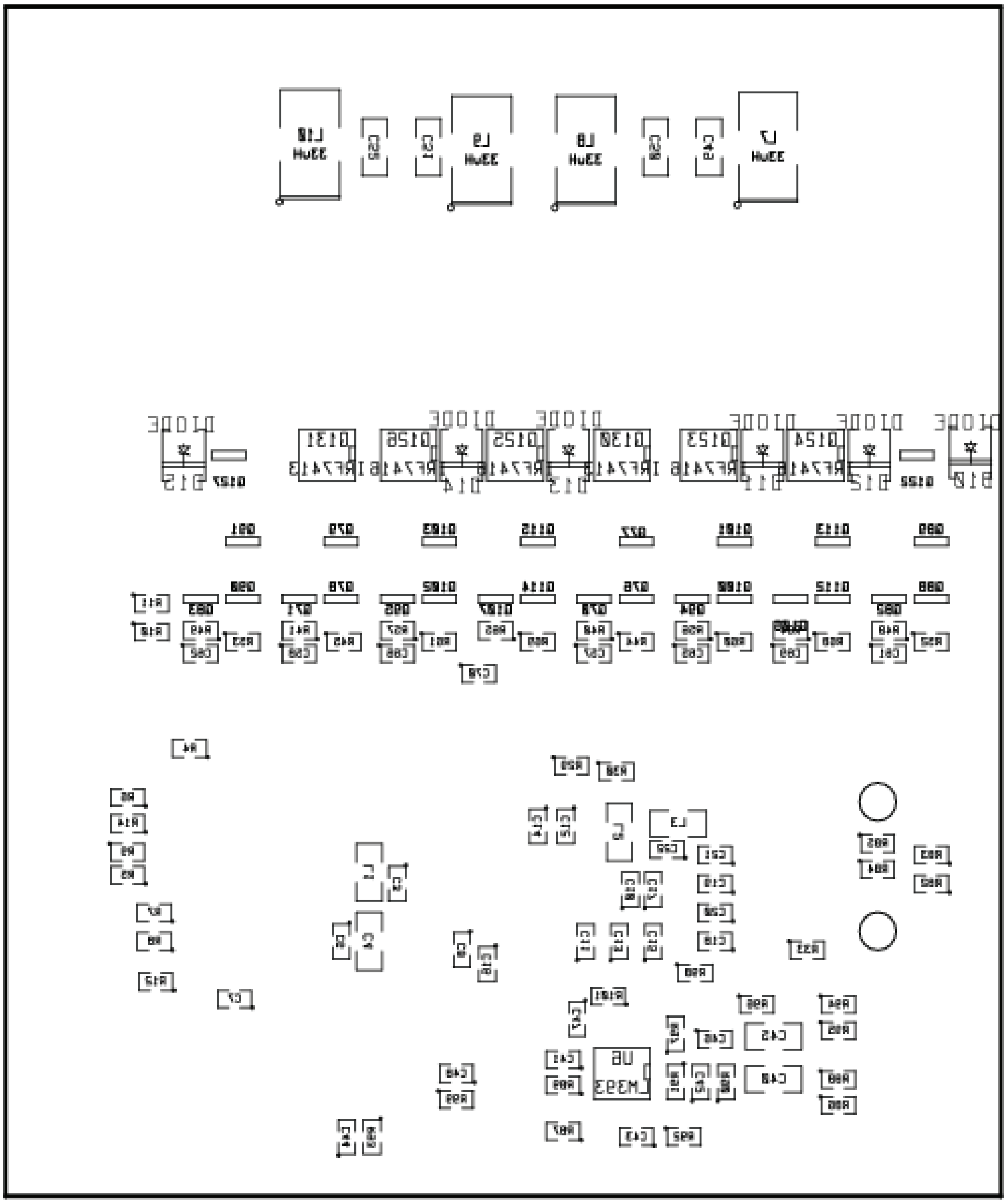
LAYOUT

1. Component
2. Copper
3. Silks Front
4. Silks Back






1PIN
P3
P2
1PIN
CONN1
POWER_JACK
U4
LM1117-3.3
150uF
C34
150uF
C37
DIODE
Q121
D9
Q129
IRF7413
Q120
IRF7416
D8
Q119
IRF7416
D7
Q128
IRF7413
Q117
IRF7416
D5
Q118
IRF7416
D6
Q116
D4
CONN_10X2
P1
K1
CRYSTAL
X1
I1
STM32F107R
U1
STE100P
RJ45_LED
J1
SV1
SV_PUSH
AUDIO_JACK
J2
CRYSTAL
X2
1PIN
1PIN



L10
L9
L8
L7
33uH
DIODE
Q131
Q126
Q125
Q130
Q123
Q124
IRF7413
IRF7416
U6
LM393

SOFTWARE TREIBER

1. snd_ethSndCard.c

```c
/*-------------------------------------------------------------------------------
 * Name:    snd_ethSndCard.c
 * Purpose: Ethernet Sound Card Driver
 * Author:  Solioz Baptiste
 * Version: V1.1
 *-------------------------------------------------------------------------------
 *
 *
 *-------------------------------------------------------------------------------*/

/*-------------------------Include File-----------------------------------------*/
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/interrupt.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/socket.h>
#include <linux/in.h>
#include <linux/net.h>
#include <linux/platform_device.h>
#include <linux/delay.h>

#include <sound/control.h>
#include <sound/core.h>
#include <sound/initval.h>
#include <sound/pcm.h>

/*-------------------------Private Definition-----------------------------------*/
/* Sending Packet definition */
#define ETH_INFO_SIZE   5
#define ETH_INFO_TYPE   1

#define ETH_CTRL_SIZE   4
#define ETH_CTRL_TYPE   2

#define ETH_DATA_HEAD   5
#define ETH_DATA_SIZE   480
#define ETH_DATA_TYPE   3

#define ETH_STOPPED_SIZE 5
#define ETH_STOPPED_TYPE 4

#define ETH_REQUEST_SIZE 2

#define SND_ETHSNDCARD_DRIVER "EthSndCard"


/*-------------------------Struct Definition------------------------------------*/
/*************************************************************************************/
/*  Structure definiton for the specific SoundCard and Hardware                      */
/*                                                                                   */
/*************************************************************************************/
static int index = SNDRV_DEFAULT_IDX1;  /* Index 0-MAX */
static char *id = SNDRV_DEFAULT_STR1;   /* ID for this card */
static DECLARE_COMPLETION(on_exit);     /* Tread Declaration */
static char *ip = "192.168.000.114";    /* Default IP Address */
static int port = 1234;                 /* Default Port Number */


/*-------------------------Chip Specific Record---------------------------------*/
struct ethSndCard
{
    struct  snd_card            *card;
    struct  snd_pcm             *pcm;
    struct  snd_pcm_substream   *substream;
    struct  socket              *ethSock;
    struct  msghdr              msg;
    struct  sockaddr_in         addr;
    wait_queue_head_t           wq;
    char    kernelStop;
    char    connected;
    int     port;
```

```
    char    *ipAddr;
    char    start;
    char    thread_id;
    char    *actualPointer;
    int     Volume;
};

/*-------------------------Hardware Definition-----------------------------------*/
static struct snd_pcm_hardware snd_ethSndCard_playback_hw =
{
  .info = ( SNDRV_PCM_INFO_INTERLEAVED |
            SNDRV_PCM_INFO_BLOCK_TRANSFER),
  .formats =    (SNDRV_PCM_FMTBIT_S16_LE |
                 SNDRV_PCM_FMTBIT_S8 |
                 SNDRV_PCM_FMTBIT_S24_LE |
                 SNDRV_PCM_FMTBIT_S32_LE),
  .rates =  SNDRV_PCM_RATE_8000_96000,
  .rate_min =   8000,
  .rate_max =   96000,
  .channels_min = 1,
  .channels_max = 2,
  .buffer_bytes_max = 64*1024,
  .period_bytes_min = 480,
  .period_bytes_max = 32*1024,
  .periods_min = 1,
  .periods_max = 2,
};

/*-------------------------SOCKET INTERFACE--------------------------------------*/
/*****************************************************************************/
/*  Thread for the Socket and Communication management                       */
/*                                                                           */
/*****************************************************************************/
/*-------------------------Thread------------------------------------------------*/
static int ethSndCard_thread(void *data)
{
    struct snd_pcm_substream *substream = (struct snd_pcm_substream*) data;
    struct ethSndCard *chip = snd_pcm_substream_chip(substream);
    struct snd_pcm_runtime *runtime = substream->runtime;
    struct kvec txVec;
    struct kvec rxVec;

    char *txBuffer;
    char rxBuffer[ETH_REQUEST_SIZE];
    unsigned int offset;
    unsigned int int_offset;
    unsigned int err;
    unsigned long DelayUSecs, DelayIncrease;
    unsigned int packetNo;

    allow_signal(SIGTERM);

    offset = 0;
    int_offset = 0;
    packetNo = 0;
    DelayUSecs = 0;
    DelayIncrease = 0;
    printk(KERN_INFO "ethSndCard-alsa: Begin thread\n");

    while(1)
    {
        if(chip->start)
        {
            if(DelayUSecs == 0)
            {
                /* First Time Init. the delay and wait 15ms */
                DelayUSecs = (ETH_DATA_SIZE/2)*1000000/(runtime->rate)/(runtime->channels);
                DelayIncrease = DelayUSecs / 10;
                msleep(15);
            }
            txBuffer = (char *) kmalloc(ETH_DATA_SIZE + ETH_DATA_HEAD,GFP_KERNEL);
            memset(txBuffer, 0, ETH_DATA_SIZE + ETH_DATA_HEAD);
```

```
            /* Copy the data into a local buffer */
            if(ETH_DATA_SIZE < (runtime->dma_bytes - offset))
            {
                memcpy(txBuffer+ETH_DATA_HEAD, runtime->dma_area + offset, ETH_DATA_SIZE);
                offset = offset + ETH_DATA_SIZE;
            }
            else
            {
                int_offset = runtime->dma_bytes - offset;
                memcpy(txBuffer+ETH_DATA_HEAD, runtime->dma_area + offset, int_offset);
                offset = ETH_DATA_SIZE - int_offset;
                memcpy(txBuffer+ETH_DATA_HEAD+int_offset, runtime->dma_area, offset);
            }
            txBuffer[0] = ETH_DATA_TYPE;
            memcpy(txBuffer+1, &packetNo, sizeof(int));
            packetNo ++;
            txVec.iov_len = ETH_DATA_SIZE + ETH_DATA_HEAD;
            txVec.iov_base = txBuffer;
            rxVec.iov_len = ETH_REQUEST_SIZE;
            rxVec.iov_base = rxBuffer;
            chip->actualPointer = runtime->dma_area + offset;
            if(chip->connected)
            {
                /* Send the data using TCP/IP stack */
                err = kernel_sendmsg(chip->ethSock, &(chip->msg), &txVec, 1, sizeof(chip->addr));
                if(err >= 0)
                    snd_pcm_period_elapsed(substream);
                err =  kernel_recvmsg(chip->ethSock, &(chip->msg), &rxVec, 1, sizeof(chip->addr), 0);
                if(rxBuffer[0] == ETH_DATA_TYPE && rxBuffer[1] != 0)
                {
                    /* MORE/LESS value : 0 = normal delay
                                         1 = no delay (send right now the next Packet)
                                         -1 = two delay (forget a sending)*/
                    if(rxBuffer[1] == 0)
                    {
                        udelay(DelayUSecs);
                    }
                    if(rxBuffer[1] == -1)
                    {
                        udelay(2*DelayUSecs);
                    }
                }

            }
            kfree(txBuffer);
        }
        else
        {
            offset = 0;
            int_offset = 0;
            packetNo = 0;
        }
        /* END of sending */
        if(chip->kernelStop)
        {
          printk(KERN_INFO "ethSndCard-alsa: Thread breaking \n");
          break;
        }
    }
    chip->thread_id = 0;
    printk(KERN_INFO "ethSndCard-alsa: Complete and exit thread \n");
    complete_and_exit(&on_exit, 0);
}
/*-------------------------Initialization thread --------------------------------*/
static int ethSndCard_init(struct snd_pcm_substream *substream)
{
    struct ethSndCard *chip = snd_pcm_substream_chip(substream);

    printk(KERN_INFO "ethSndCard-alsa: thread_init\n");
    chip->thread_id = 0;
    init_waitqueue_head(&(chip->wq));
    chip->kernelStop = 0;
```

```
    chip->thread_id = kernel_thread(ethSndCard_thread, (void*) substream, CLONE_KERNEL);
    if(chip->thread_id == 0)
    {
        return -EIO;
    }

    return 0;
}
/*-------------------------Exit thread-----------------------------------------*/
static void ethSndCard_exit(struct snd_pcm_substream *substream)
{
    struct ethSndCard *chip = snd_pcm_substream_chip(substream);

    printk(KERN_INFO "ethSndCard-alsa: thread_exit\n");
    if(chip->thread_id)
    {
        chip->kernelStop = 1;
        kill_pid(find_pid_ns(chip->thread_id, &init_pid_ns), SIGTERM, 1);
        wait_for_completion(&on_exit);
    }
}

/*-------------------------PCM INTERFACE---------------------------------------*/
/*****************************************************************************/
/*  All PCM Function for the Playback mode                                   */
/*  Include the PCM constructor                                              */
/*****************************************************************************/

/*-------------------------Open CallBack PlayBack------------------------------*/
static int snd_ethSndCard_playback_open(struct snd_pcm_substream *substream)
{
    struct ethSndCard *chip = snd_pcm_substream_chip(substream);
    struct snd_pcm_runtime *runtime = substream->runtime;
    int err;

    printk(KERN_INFO "ethSndCard-alsa: snd_Playback_Open Enter\n");
    runtime->hw = snd_ethSndCard_playback_hw;
    err = sock_create(AF_INET, SOCK_STREAM, IPPROTO_TCP, &(chip->ethSock));
    if(err<0)
    {
        printk(KERN_ALERT "socket failed %d\n",err);
        return err;
    }

    chip->start = 0;
    memset(&(chip->addr), 0, sizeof(struct sockaddr_in));

    chip->addr.sin_family = AF_INET;
    chip->addr.sin_addr.s_addr = in_aton(chip->ipAddr);
    chip->addr.sin_port = htons(chip->port);

    //socket, socket addr, addrlen, flag
    err = kernel_connect(chip->ethSock, (struct sockaddr *)
                         &(chip->addr), sizeof(struct sockaddr), 0);
    if(err < 0)
    {
        printk(KERN_INFO "ethSndCard-alsa: Socket not connected with err = %d\n", err);
        chip->connected = 0;
        sock_release(chip->ethSock);
        return err;
    }
    chip->connected = 1;
    ethSndCard_init(substream);
    return 0;
}

/*-------------------------Close CallBack PlayBack-----------------------------*/
static int snd_ethSndCard_playback_close(struct snd_pcm_substream *substream)
{
    struct ethSndCard *chip = snd_pcm_substream_chip(substream);

    printk(KERN_INFO "ethSndCard-alsa: snd_PlayBack_Close Enter\n");
```

```
    /* the hardware-specific codes will be here */
    ethSndCard_exit(substream);
    if(chip->connected)
    {
        chip->connected = 0;
        kernel_sock_shutdown(chip->ethSock, SHUT_RDWR);
        sock_release(chip->ethSock);
    }
    return 0;
}

/*-------------------------HW_params CallBack-------------------------------------*/

static int snd_ethSndCard_pcm_hw_params(struct snd_pcm_substream *substream,
                                        struct snd_pcm_hw_params *hw_params)
{
    int err;
    printk(KERN_INFO "ethSndCard-alsa: snd_Pcm_Hw_Params Enter/Exit\n");
    err = snd_pcm_lib_malloc_pages(substream,params_buffer_bytes(hw_params));
    return err;
}

/*-------------------------HW_free CallBack---------------------------------------*/
static int snd_ethSndCard_pcm_hw_free(struct snd_pcm_substream *substream)
{
    printk(KERN_INFO "ethSndCard-alsa: snd_Pcm_Hw_Free Enter/Exit\n");
    return snd_pcm_lib_free_pages(substream);
}

/*-------------------------Prepare CallBack---------------------------------------*/
static int snd_ethSndCard_pcm_prepare(struct snd_pcm_substream *substream)
{
    struct ethSndCard *chip = snd_pcm_substream_chip(substream);
    struct snd_pcm_runtime *runtime = substream->runtime;
    struct kvec vec;
    int result;
    char *send_info;

    printk(KERN_INFO "ethSndCard-alsa: snd_Pcm_Prepare Enter\n");

    send_info = (char *) kmalloc(ETH_INFO_SIZE,GFP_KERNEL);
    memset(send_info, 0x00, ETH_INFO_SIZE);

    chip->msg.msg_name = &(chip->addr);
    chip->msg.msg_namelen = sizeof(chip->addr);
    chip->msg.msg_flags = 0;

    switch(runtime->rate)
    {
        case 8000:      send_info[1] = 1;
            break;
        case 11025:     send_info[1] = 2;
            break;
        case 16000:     send_info[1] = 3;
            break;
        case 22050:     send_info[1] = 4;
            break;
        case 32000:     send_info[1] = 5;
            break;
        case 44100:     send_info[1] = 6;
            break;
        case 48000:     send_info[1] = 7;
            break;
        case 64000:     send_info[1] = 8;
            break;
        case 88200:     send_info[1] = 9;
            break;
        case 96000:     send_info[1] = 10;
            break;
        case 176400:    send_info[1] = 11;
            break;
        case 192000:    send_info[1] = 12;
```

```
            break;
        default:
            break;
    }
    switch(runtime->format)
    {
        case SNDRV_PCM_FORMAT_S8:   send_info[2] = 1;
            break;
        case SNDRV_PCM_FORMAT_S16:  send_info[2] = 2;
            break;
        case SNDRV_PCM_FORMAT_S24:  send_info[2] = 3;
            break;
        case SNDRV_PCM_FORMAT_S32:  send_info[2] = 4;
            break;
        default:
            break;
    }
    send_info[3] = (char)runtime->channels;
    send_info[0] = ETH_INFO_TYPE;

    vec.iov_base = send_info;
    vec.iov_len = ETH_INFO_SIZE;

    if(chip->connected)
    {
        result = kernel_sendmsg(chip->ethSock, &(chip->msg), &vec, 1, sizeof(chip->addr));
    }
    return 0;
}

/*-------------------------Trigger CallBack---------------------------------------*/
static int snd_ethSndCard_pcm_trigger(struct snd_pcm_substream *substream, int cmd)
{
    struct ethSndCard *chip = snd_pcm_substream_chip(substream);
    struct kvec vec;
    int result;
    char *send_stopped;

    printk(KERN_INFO "ethSndCard-alsa: snd_Pcm_Trigger Enter with Cmd: %d \n", cmd);
    switch (cmd)
    {
        case SNDRV_PCM_TRIGGER_START:
            chip->actualPointer = substream->runtime->dma_area;
            chip->start = 1;
            break;
        case SNDRV_PCM_TRIGGER_STOP:
            chip->start = 0;

            send_stopped = (char *) kmalloc(ETH_STOPPED_SIZE,GFP_KERNEL);
            memset(send_stopped, 0x00, ETH_STOPPED_SIZE);

            send_stopped[0] = ETH_STOPPED_TYPE;

            vec.iov_base = send_stopped;
            vec.iov_len = ETH_STOPPED_SIZE;
            if(chip->connected)
            {
                result = kernel_sendmsg(chip->ethSock, &chip->msg, &vec, 1, sizeof(chip->addr));
                printk(KERN_INFO "ethSndCard-alsa: Send Stopped\n");
            }
            kfree(send_stopped);
            break;
        default:
            return -EINVAL;
    }
    return 0;
}

/*-------------------------Pointer CallBack---------------------------------------*/
static snd_pcm_uframes_t snd_ethSndCard_pcm_pointer(struct snd_pcm_substream *substream)
{
    struct ethSndCard *chip = snd_pcm_substream_chip(substream);
```

```
    struct snd_pcm_runtime *runtime = substream->runtime;
    snd_pcm_uframes_t pos;
    unsigned long bytes;

    /* get the current hardware pointer */
    bytes = (unsigned long)chip->actualPointer - (unsigned long)runtime->dma_area;
    pos = bytes_to_frames(runtime, bytes);
    if (pos >= runtime->buffer_size)
    {
        pos -= runtime->buffer_size;
    }

    return pos;
}

/*-------------------------Operators PlayBack-----------------------------------*/
static struct snd_pcm_ops snd_ethSndCard_playback_ops =
{
    .open = snd_ethSndCard_playback_open,
    .close = snd_ethSndCard_playback_close,
    .ioctl = snd_pcm_lib_ioctl,
    .hw_params = snd_ethSndCard_pcm_hw_params,
    .hw_free = snd_ethSndCard_pcm_hw_free,
    .prepare = snd_ethSndCard_pcm_prepare,
    .trigger = snd_ethSndCard_pcm_trigger,
    .pointer = snd_ethSndCard_pcm_pointer,
};

/*-------------------------PCM Construtor-------------------------------------------*/
static int __devinit snd_ethSndCard_new_pcm(struct ethSndCard *chip)
{
    struct snd_pcm *pcm;
    int err;

    printk(KERN_INFO "ethSndCard-alsa: snd_New_Pcm Enter\n");
    err = snd_pcm_new(chip->card, "Ethernet Sound Card", 4, 1, 0, &pcm);
    if (err < 0)
    {
        return err;
    }
    pcm->private_data = chip;
    strcpy(pcm->name, "EthSndCard");
    chip->pcm = pcm;
    /* Set operators */
    snd_pcm_set_ops(pcm, SNDRV_PCM_STREAM_PLAYBACK, &snd_ethSndCard_playback_ops);
    /* pre-allocation of buffers */
    snd_pcm_lib_preallocate_pages_for_all(  pcm, SNDRV_DMA_TYPE_CONTINUOUS,
                                            snd_dma_continuous_data(GFP_KERNEL),
                                            64*1024, 64*1024);
    return 0;
}

/*-------------------------SoundCard Control-----------------------------------*/
/*****************************************************************************/
/*  All Function for the Control Management (Volume)                         */
/*                                                                           */
/*****************************************************************************/

/*-------------------------Control Information-----------------------------------*/
static int ethSndCard_ctrl_info(struct snd_kcontrol *kcontrol,
                                struct snd_ctl_elem_info *uinfo)
{
    printk(KERN_INFO "ethSndCard-alsa: Kcontrol_info\n");
    uinfo->type = SNDRV_CTL_ELEM_TYPE_INTEGER;
    uinfo->count = 1;
    uinfo->value.integer.min = 0;
    uinfo->value.integer.max = 255;
    return 0;
}

/*-------------------------Control Getting-----------------------------------*/
static int ethSndCard_ctrl_get( struct snd_kcontrol *kcontrol,
```

```
                                struct snd_ctl_elem_value *ucontrol)
{
    struct ethSndCard *chip = snd_kcontrol_chip(kcontrol);
    printk(KERN_INFO "ethSndCard-alsa: Kcontrol_get\n");
    ucontrol->value.integer.value[0] = chip->Volume;
    return 0;
}


/*-------------------------Control Putting---------------------------------------*/
static int ethSndCard_ctrl_put( struct snd_kcontrol *kcontrol,
                                struct snd_ctl_elem_value *ucontrol)
{
    struct ethSndCard *chip = snd_kcontrol_chip(kcontrol);
    int changed = 0;
    struct kvec vec;
    int result;
    char *send_control;

    printk(KERN_INFO "ethSndCard-alsa: Kcontrol_put\n");
    if (chip->Volume != ucontrol->value.integer.value[0])
    {
        chip->Volume = ucontrol->value.integer.value[0];
        send_control = (char *) kmalloc(ETH_STOPPED_SIZE,GFP_KERNEL);
        memset(send_control, 0x00, ETH_STOPPED_SIZE);

        send_control[0] = ETH_CTRL_TYPE;
        send_control[1] = 0;
        memcpy(send_control+2, &(chip->Volume), sizeof(int));

        vec.iov_base = send_control;
        vec.iov_len = ETH_CTRL_SIZE;
        if(chip->connected)
        {
            result = kernel_sendmsg(chip->ethSock, &chip->msg, &vec, 1, sizeof(chip->addr));
        }
        kfree(send_control);
        changed = 1;
    }
    return changed;
}

/*-------------------------Control Structure---------------------------------------*/
static struct snd_kcontrol_new ethSndCard_ctrl __devinitdata =
{
    .iface = SNDRV_CTL_ELEM_IFACE_MIXER,
    .name = "Master Playback Volume",
    .index = 0,
    .access = SNDRV_CTL_ELEM_ACCESS_READWRITE,
    .private_value = 0xFFFF,
    .info = ethSndCard_ctrl_info,
    .get = ethSndCard_ctrl_get,
    .put = ethSndCard_ctrl_put
};

/*-------------------------SoundCard Specification---------------------------------*/
/*****************************************************************************/
/*  Driver constructor and destructor                                        */
/*  Include Module init and exit                                             */
/*****************************************************************************/

/*-------------------------Specific ethSndCard Desctructor-----------------------*/
static int snd_ethSndCard_free(struct ethSndCard *chip)
{
    printk(KERN_INFO "ethSndCard-alsa: snd_Free Enter\n");

    /* release the data */
    kfree(chip);
    return 0;
}

/*-------------------------Component Destructor----------------------------------*/
```

```c
static int snd_ethSndCard_dev_free(struct snd_device *device)
{
    printk(KERN_INFO "ethSndCard-alsa: snd_Dev_Free Enter/Exit\n");
    return snd_ethSndCard_free(device->device_data);
}

/*-------------------------Specific ethSndCard Constructor-----------------------*/
static int __devinit snd_ethSndCard_create( struct snd_card *card,
                                            struct platform_device *myDev,
                                            struct ethSndCard **rchip)
{
    struct ethSndCard *chip;
    int err;
    char* myIP = ip;
    static struct snd_device_ops ops =
    {
        .dev_free = snd_ethSndCard_dev_free,
    };

    printk(KERN_INFO "ethSndCard-alsa: snd_Create Enter\n");
    chip = kzalloc(sizeof(*chip), GFP_KERNEL);
    if (chip == NULL)
    {
        return -ENOMEM;
    }
    *rchip = NULL;

    /* initialize the stuff */
    chip->card = card;
    chip->ethSock = 0;

    chip->ipAddr = (char*)kzalloc(sizeof(myIP), GFP_KERNEL);
    chip->ipAddr = myIP;
    chip->port = port;
    chip->connected = 0;
    chip->kernelStop = 0;

    err = snd_device_new(card, SNDRV_DEV_LOWLEVEL, chip, &ops);
    if (err < 0)
    {
        snd_ethSndCard_free(chip);
        return err;
    }
    snd_card_set_dev(card, &myDev->dev);
    *rchip = chip;
    return 0;
}

/*-------------------------SoundCard Constructor-----------------------------------*/
static int __devinit snd_ethSndCard_probe(struct platform_device *myDev)
{
    struct snd_card *card;
    struct ethSndCard *chip;
    int err;

    printk(KERN_INFO "ethSndCard-alsa: snd_Probe Enter\n");
    /* Create a card instance */
    err = snd_card_create(index, id, THIS_MODULE, 0, &card);
    if (err < 0)
    {
        return err;
    }

    /* Create a main component */
    err = snd_ethSndCard_create(card, myDev, &chip);
    if (err < 0)
    {
        snd_card_free(card);
        return err;
    }
    snd_card_set_dev(card, &myDev->dev);
```

```
    /* Set the drivers ID and the name strings */
    strcpy(card->driver, "EthSndCardDriver");
    strcpy(card->shortname, "snd_ethSndCard");
    sprintf(card->longname, "%s", card->shortname);

    err = snd_ethSndCard_new_pcm(chip);
    if (err < 0)
    {
        snd_card_free(card);
        return err;
    }
    /*Initialize Sound Control*/
    err = snd_ctl_add(card, snd_ctl_new1(&ethSndCard_ctrl, chip));
    if (err < 0)
    {
        snd_card_free(card);
        return err;
    }

    /* Register the card instance */
    err = snd_card_register(card);
    if (err < 0)
    {
        snd_card_free(card);
        return err;
    }
    platform_set_drvdata(myDev, card);
    return 0;
}

/*-------------------------SoundCard Destructor---------------------------------------*/
static void __devexit snd_ethSndCard_remove(struct platform_device *myDev)
{
    struct snd_card *card = platform_get_drvdata(myDev);
    printk(KERN_INFO "ethSndCard-alsa: snd_Remove Enter\n");
    snd_card_free(card);
    platform_set_drvdata(myDev, NULL);
}

/*-------------------------Driver Description-----------------------------------------*/
static struct platform_driver snd_ethSndCard_driver = {
            .probe = snd_ethSndCard_probe,
            .remove = __devexit_p(snd_ethSndCard_remove),
            .driver =   {
                        .name = SND_ETHSNDCARD_DRIVER,
                        .owner = THIS_MODULE,
                        }
};

static struct platform_device myDevice;

/*-------------------------Module Init------------------------------------------------*/
static int __init snd_ethSndCard_init(void)
{
    int err;
    memset(&myDevice, 0, sizeof(struct platform_device));
    myDevice.name = SND_ETHSNDCARD_DRIVER;
    myDevice.id = -1;
    err = platform_device_register(&myDevice);
    err = platform_driver_register(&snd_ethSndCard_driver);
    printk(KERN_INFO "ethSndCard-alsa: module loading. err = %d\n", err);
    return err;
}

/*-------------------------Module Clean Up--------------------------------------------*/
static void __exit snd_ethSndCard_exit(void)
{
    printk(KERN_INFO "ethSndCard-alsa: module erase...\n");
    platform_driver_unregister(&snd_ethSndCard_driver);
    platform_device_unregister(&myDevice);
}
```

```c
/*-------------------------Module Specification-----------------------------------*/
module_init(snd_ethSndCard_init);
module_exit(snd_ethSndCard_exit);
module_param(ip, charp, S_IRUGO);
module_param(port, int, S_IRUGO);

MODULE_AUTHOR("Baptiste Solioz");
MODULE_DESCRIPTION("Sound driver for Ethernet Sound Card");
MODULE_LICENSE("GPL");
```

FIRMWARE : SOUNDKARTE

1. EthSndCard.c
2. snd.c
3. snd.h
4. main.c

```
/**
********************************************************************************************
* file :   ethSndCard.h
* author : Solioz Baptiste <baptiste.solioz@gmail.com>
* version: V1.0
* date :   21.11.11
* brief :  Sound Application Definiton (Structure + Function)
********************************************************************************************
*/

#ifndef __ETHSNDCARD_H__
#define __ETHSNDCARD_H__

#include "snd.h"

/* Give the Specific Application Structure */
typedef struct snd_state uip_tcp_appstate_t;

/* UIP_APPCALL: the name of the application function. This function
   must return void and take no arguments (i.e., C type "void
   appfunc(void)"). */
#ifndef UIP_APPCALL
#define UIP_APPCALL                    snd_appcall
#endif

#endif /* __ETHSNDCARD_H__ */
```

```c
/**
******************************************************************************************************
* file :   snd.c
* author : Solioz Baptiste <baptiste.solioz@gmail.com>
* version: V1.0
* date :   21.11.11
* brief :  Sound Application
******************************************************************************************************
*/

#include "stm32_io.h"
#include "uip.h"
#include "snd.h"
//#include "FreeRTOSConfig.h"


/*****************************************************************************************************/
/* Name : LedTester                                                                                  */
/* Description : Debugging function for invert Output                                                */
/*****************************************************************************************************/
void ledTester()
{
    static char test = 0;
    if(test == 0)
    {
        test = 1;
        portSet(GPIOA, 5);
    }
    else
    {
        test = 0;
        portClr(GPIOA, 5);
    }
}

/*****************************************************************************************************/
/* Name : Handle Information                                                                         */
/* Description : Modify Format and rate                                                              */
/*****************************************************************************************************/
static void handle_info(struct snd_state *s)
{
    unsigned int frequency;
    switch(s->info.format)
    {
        case SND_FORMAT_8B:    format = 8; break;
        case SND_FORMAT_16B:   format = 16; break;
        case SND_FORMAT_24B:   format = 24; break;
        case SND_FORMAT_32B:   format = 32; break;
        default:                break;
    }

    TIM_Cmd( TIM5, DISABLE );
    switch(s->info.rate)
    {
        case SND_RATE_8K:       frequency = 8000; break;
        case SND_RATE_11K:      frequency = 11025; break;
        case SND_RATE_16K:      frequency = 16000; break;
        case SND_RATE_22K:      frequency = 22050; break;
        case SND_RATE_32K:      frequency = 32000; break;
        case SND_RATE_44K:      frequency = 44100; break;
        case SND_RATE_48K:      frequency = 48000; break;
        case SND_RATE_64K:      frequency = 64000; break;
        case SND_RATE_88K:      frequency = 88200; break;
        case SND_RATE_96K:      frequency = 96000; break;
```

```
        case SND_RATE_176K:     frequency = 176400; break;
        case SND_RATE_192K:     frequency = 192000; break;
        default:                frequency = 32000; break;
    }
    TIM5->ARR = configCPU_CLOCK_HZ/(frequency); // modifiy the interrupt frequency
    TIM5->EGR = TIM_PSCReloadMode_Immediate;
    TIM_Cmd( TIM5, ENABLE );
}

/*******************************************************************************************/
/* Name :  Handle data copy                                                                */
/* Description : Copy the data from the receive Buffer into a local Buffer                 */
/* Parameters : Length of the data for copying                                             */
/*******************************************************************************************/
static void handle_copy(uint16_t len)
{
    uint16_t int_offset;
    len >>= 1;
    if(len < (BUFFER_SIZE - indexIn))
    {
        memcpy(&(transBuffer[indexIn]), ((uint8_t*)(uip_appdata+5)), (len<<1));
        indexIn += len;
    }
    else
    {
        int_offset = BUFFER_SIZE - indexIn;
        memcpy(&(transBuffer[indexIn]), ((uint8_t*)(uip_appdata+5)), (int_offset<<1));
        indexIn = len - int_offset;
        memcpy(transBuffer,((uint8_t*)(uip_appdata+5))+(int_offset<<1), (indexIn<<1));
        int_offset = 0;
    }
}

/*******************************************************************************************/
/* Name : Handle control data                                                              */
/* Description : Modify Volume and mute Value                                              */
/*******************************************************************************************/
static void handle_control(struct snd_state *s)
{
    s->len = 5;
    s->outputbuf[0] = SND_CONTROL;
    s->control.mute = s->inputbuf[1];
    s->control.volume = (s->inputbuf[3]<<8) | s->inputbuf[2];
    Volume = s->control.volume;
    memcpy(&(s->outputbuf[1]), &(s->control), s->len-1);
    uip_send(s->outputbuf, s->len);
}

/*******************************************************************************************/
/* Name : Handle Data                                                                      */
/* Description : Copy the data and control the data stream                                 */
/*******************************************************************************************/
static void handle_data(struct snd_state *s)
{
    int8_t  requestMore = 0;
    uint16_t counterBuffer = 0;
    uint16_t actPacketNo = 0;

    handle_copy(SND_REQUEST_DATA);
    actPacketNo = (s->inputbuf[2]<<8) | s->inputbuf[1];
    counterBuffer = (indexIn-indexOut) & (BUFFER_SIZE-1);
    if(counterBuffer < 1460)
    {
        requestMore = 1;
```

```
        ledTester();
    }
    if(counterBuffer > (BUFFER_SIZE-1460))
        requestMore = -1;
    memset(&(s->outputbuf), 0, sizeof(s->outputbuf));
    s->len = 2;
    s->outputbuf[0] = SND_DATA;
    s->outputbuf[1] = requestMore;
    uip_send(s->outputbuf, s->len);
}

/*---------------------------------------------------------------------------*/
/* Name : Handle Start and Stop                                              */
/* Description : Stop all activity of the application when the playback is finished       */
/*---------------------------------------------------------------------------*/
static void handle_stopped(struct snd_state *s)
{
    begin = 0;
    TIM_Cmd( TIM5, DISABLE );
    indexIn = 0;
    indexOut = 0;
    TIM3->CCER = 0x0000;    //disable all Negativ PWM
    TIM1->CCER = 0x0000;    //disable all Negativ PWM
    portSet(GPIOC, 9);      //Enable 0V Negativ
    portSet(GPIOA, 10);     //Disable 0V Positiv
}

/*---------------------------------------------------------------------------*/
/* Name : Handle Input                                                       */
/* Description : Control the protocol and call the right Function                       */
/*---------------------------------------------------------------------------*/
static void handle_input(struct snd_state *s)
{
    s->inputbuf = (uint8_t *)uip_appdata;

    switch(s->inputbuf[0])
    {
    case SND_INFO:      s->info.rate = s->inputbuf[1];
        s->info.format = s->inputbuf[2];
        s->info.channel = s->inputbuf[3];
        channel = s->info.channel;
        handle_info(s);
        break;
    case SND_DATA:      handle_data(s);
        begin = 1;
        break;
    case SND_CONTROL:   handle_control(s);
        break;
    case SND_STOPPED:   handle_stopped(s);
        break;
    default:            break;
    }
}

/*---------------------------------------------------------------------------*/
/* Name : Call Sound Application Function                                    */
/* Description : Called by TCP/IP stack and test the Connection Statment                */
/*---------------------------------------------------------------------------*/
void snd_appcall(void)
{
    struct snd_state *s = (struct snd_state *)&(uip_conn->appstate);

    //test the connection statment
    if(uip_closed() || uip_aborted() || uip_timedout())
```

```
    {
        handle_stopped(s);
    }
    else if(uip_connected())
    {
        s->timer = 0;
    }
    else if(s != NULL)
    {
        if(uip_poll())
        {
            ++s->timer;
            if(s->timer >= 20)
            {
                uip_abort();
            }
        }
        else
        {
            s->timer = 0;
        }
        if(uip_newdata())
        {
            handle_input(s);
        }
    }
    else
    {
        uip_abort();
    }
}

/*---------------------------------------------------------------------------------------------*/
/* Name :  Sound Init                                                                          */
/* Description : Give the right Port for this Application. Called at the initalization    */
/*---------------------------------------------------------------------------------------------*/
void httpd_init(void)
{
    uip_listen(HTONS(1234));
}
/*-----------------------------------------------------------------------------*/
```

```c
/**
****************************************************************************************************
* file :   snd.h
* author : Solioz Baptiste <baptiste.solioz@gmail.com>
* version: V1.0
* date :   21.11.11
* brief :  Sound Header
****************************************************************************************************
*/

#ifndef __SND_H__
#define __SND_H__

#include "string.h"

//Define protocole
#define SND_INFO                0x01
#define SND_CONTROL             0x02
#define SND_DATA                0x03
#define SND_STOPPED             0x04

//Define rate information
#define SND_RATE_8K             0x01
#define SND_RATE_11K            0x02
#define SND_RATE_16K            0x03
#define SND_RATE_22K            0x04
#define SND_RATE_32K            0x05
#define SND_RATE_44K            0x06
#define SND_RATE_48K            0x07
#define SND_RATE_64K            0x08
#define SND_RATE_88K            0x09
#define SND_RATE_96K            0x0A
#define SND_RATE_176K           0x0B
#define SND_RATE_192K           0x0C

//Define format information
#define SND_FORMAT_8B           0x01
#define SND_FORMAT_16B          0x02
#define SND_FORMAT_24B          0x03
#define SND_FORMAT_32B          0x04

//Define channels information
#define SND_CHANNEL_1CH         0x01
#define SND_CHANNEL_2CH         0x02

//request data
#define SND_REQUEST_DATA        480

//Buffer Size
#define BUFFER_SIZE             2048*4

// extern variable from the main.c
extern int16_t  transBuffer[BUFFER_SIZE];
extern uint16_t indexIn;
extern uint16_t indexOut;
extern uint8_t  begin;
extern uint8_t  format;
extern uint8_t  channel;
extern uint16_t Volume;

/* Specific structure for the Protocole */
struct snd_info
{
    uint8_t     rate;
```

```
    uint8_t     format;
    uint8_t     channel;
};

struct snd_control
{
    uint8_t     mute;
    uint16_t    volume;
};

struct snd_state
{
    uint8_t     timer;
    uint8_t     outputbuf[5];
    uint8_t*    inputbuf;
    uint32_t    len;
    uint16_t    oldPacketNo;
    struct snd_info info;
    struct snd_control control;
};

/* Function of snd.c */
void httpd_init(void);
void snd_appcall(void);
void ledTester();

#endif /* __SND_H__ */
```

```c
/**
********************************************************************************************************
* file :   main.c
* author : Solioz Baptiste <baptiste.solioz@gmail.com>
* version: V1.0
* date :   21.11.11
* brief :  Main program body
********************************************************************************************************
*/

/* Includes ------------------------------------------------------------------------------------------*/
#include "stm32_io.h"
#include "stm32f10x_conf.h"
#include "stm32f10x.h"
#include "stm32_eth.h"
#include "ethernet.h"
#include "stm32f10x_tim.h"

/* Scheduler includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"

/*-----------------------------------------------------------------------------------------------------*/
/* Defines -------------------------------------------------------------------------------------------*/
/* The time between cycles of the 'check' functionality (defined within the
tick hook. */
#define mainCHECK_DELAY                ( ( portTickType ) 5000 / portTICK_RATE_MS )

/* Task priorities. */
#define mainQUEUE_POLL_PRIORITY        ( tskIDLE_PRIORITY + 2 )
#define mainSEM_TEST_PRIORITY          ( tskIDLE_PRIORITY + 1 )
#define mainBLOCK_Q_PRIORITY           ( tskIDLE_PRIORITY + 2 )
#define mainUIP_TASK_PRIORITY          ( tskIDLE_PRIORITY + 3 )
#define mainFLASH_TASK_PRIORITY        ( tskIDLE_PRIORITY + 2 )
#define mainLCD_TASK_PRIORITY          ( tskIDLE_PRIORITY + 3 )
#define mainINTEGER_TASK_PRIORITY       ( tskIDLE_PRIORITY )
#define mainGEN_QUEUE_TASK_PRIORITY    ( tskIDLE_PRIORITY )

#define mainBASIC_WEB_STACK_SIZE        ( configMINIMAL_STACK_SIZE * 4 )

/* The period of the system clock in nano seconds.  This is used to calculate
the jitter time in nano seconds. */
#define mainNS_PER_CLOCK ( ( unsigned long ) ( ( 1.0 / \
                           ( double ) configCPU_CLOCK_HZ ) * 1000000000.0 ) )

#define RCC_PLLSource_HSE_Div2           ((u32)0x00030000)


/* Define the Board OLIMEX Test board or Specific Soundcard Hardware */
//#define OLIMEX_VERSION

/*-----------------------------------------------------------------------------------------------------*/
/* Private define ------------------------------------------------------------------------------------*/
#define DIV             1 //division pro 4
#define PWM_FREQ       96000
#define PERIOD         512//configCPU_CLOCK_HZ/(PWM_FREQ)
/* Private macro -------------------------------------------------------------------------------------*/
#define stereo_6V_p        GPIOA, 5
#define stereo_4V_p        GPIOA, 6
#define stereo_2V_p        GPIOB, 6
#define stereo_0V_p        GPIOB, 5
#define stereo_6V_m        GPIOC, 13
```

```
#define stereo_4V_m        GPIOC, 12
#define stereo_2V_m        GPIOC, 10
#define stereo_0V_m        GPIOC, 11
/*-----------------------------------------------------------------------------------------*/
/* Variables sharing with snd.c ------------------------------------------------------------*/

int16_t         transBuffer[BUFFER_SIZE];
uint16_t        indexIn;
uint16_t        indexOut;
uint8_t         begin;
uint8_t         format;
uint8_t         channel;
uint16_t        Volume;

/*-----------------------------------------------------------------------------------------*/
/* Private function prototypes -------------------------------------------------------------*/
void Tim5Handler(void);
void RCC_Init(void);
void Ethernet_Init(void);
void Output_Init(void);
void TIM_Init(void);


/* Private functions -----------------------------------------------------------------------*/
/*****************************************************************************************/
/* Name : Interrupt Routine of Timer 5                                                   */
/* Description :  Management of all PWM Output                                           */
/*****************************************************************************************/
void Tim5Handler (void)
{
    static uint8_t  wait = 0;
    static uint16_t waitCounter = 0;
    int32_t         outVal = 0;
#ifdef OLIMEX_VERSION // Code Version for olimex board with 2 PWM Output
    if(begin == 1 && wait == 1)
    {
        // Take the Value from buffer
        outVal = (int64_t) transBuffer[indexOut];
        // Volume Management
        outVal *= Volume;
        outVal >>= 8;
        // Only positiv Side
        outVal  += 32768;
        if(format < 10)
            TIM3->CCR3 = (uint16_t)(outVal<<2); // modifiy PWM value
        else
            TIM3->CCR3 = (uint16_t)((outVal*PERIOD)/0xFFFF);
        if(channel == 2)
        {

            indexOut++;
            outVal = (int64_t) transBuffer[indexOut];
            outVal *= Volume;
            outVal >>= 8;
            outVal  += 32768;
            if(format < 10)
                TIM4->CCR4 = (uint16_t)(outVal & 0x01FF); // modifiy PWM value
            else
                TIM4->CCR4 = (uint16_t)((outVal*PERIOD/0xFFFF));
        }
        indexOut++;
        indexOut &= (BUFFER_SIZE-1);
    }
#else // Specific board version with 6 PWM Outputs
```

```
    if(begin == 1 && wait == 1)
    {
        // Take the Value from buffer
        outVal = (int32_t) transBuffer[indexOut];
        // Volume Management
        outVal *= Volume;
        outVal >>= 7;

        //outVal += 32768;            //Only Positiv Side

        if(outVal >= 0)
        {
            TIM3->CCER = 0x0000;    //disable all Negativ PWM
            portClr(GPIOC, 9);      //Enable 0V Negativ
            portSet(GPIOA, 10);     //Disable 0V Positiv
            if(outVal < 16384)     //Voltage 2V
            {
                TIM1->CCR4 = (uint16_t)((outVal*512)/16384);
                TIM1->CCER = 0x1000; //Select 2V PWM Output
            }
            else if(outVal < 32768)     //Voltage 4V
            {
                TIM1->CCR2 = (uint16_t)((outVal*512)/32768);
                TIM1->CCER = 0x0010; //Select 4V PWM Output
            }
            else
            {
                //Voltage 6V
                TIM1->CCR1 = (uint16_t)((outVal*512)/65536);
                TIM1->CCER = 0x0001;    //Select 6V PWM Output
            }
        }
        else // if(outVal < 0)
        {
            TIM1->CCER = 0x0000;
            portClr(GPIOA, 10);
            portSet(GPIOC, 9);
            outVal *= -1;   // Absolute Value
            if(outVal < 16384)
            {
                TIM3->CCR1 = (uint16_t)((outVal*512)/16384);
                TIM3->CCER = 0x0001;
            }
            else if(outVal < 32768)
            {
                TIM3->CCR3 = (uint16_t)((outVal*512)/32768);
                TIM3->CCER = 0x0100;
            }
            else
            {
                TIM3->CCR2 = (uint16_t)((outVal*512)/65536);
                TIM3->CCER = 0x0010;
            }
        }
        //Modify Index Value
        indexOut++;
        if(channel == 2) // Canal 2 not implemented
            indexOut++;
        indexOut &= (BUFFER_SIZE-1);
    }
#endif
    // Wait 2KByte Data (BUFFER_SIZE / 2) for begin
    if(begin == 1 && wait == 0)
    {
```

```
        waitCounter ++;
        if(waitCounter >= BUFFER_SIZE/2)
        {
            wait = 1;
        }
    }
    TIM5->SR = (uint16_t)0xFFFE;
}

/*******************************************************************************************/
/* Name : Init RCC                                                                         */
/* Description : Use Function of RCC Driver stm32f10x_rcc.h                                */
/*******************************************************************************************/
void RCC_Init()
{
    /* Enable ETHERNET clock  */
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_ETH_MAC | RCC_AHBPeriph_ETH_MAC_Tx |
    RCC_AHBPeriph_ETH_MAC_Rx, ENABLE);

    /* Enable GPIOs clocks */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |
                           RCC_APB2Periph_GPIOC |RCC_APB2Periph_AFIO, ENABLE);
#ifdef OLIMEX_VERSION
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE, ENABLE);
    RCC_APB1PeriphClockCmd( RCC_APB1Periph_TIM4, ENABLE );
#else
    RCC_APB2PeriphClockCmd( RCC_APB2Periph_TIM1, ENABLE );
#endif
    RCC_APB1PeriphClockCmd( RCC_APB1Periph_TIM3, ENABLE );
    RCC_APB1PeriphClockCmd( RCC_APB1Periph_TIM5, ENABLE );
}

/*******************************************************************************************/
/* Name : Init Ethernet                                                                    */
/* Description : Use Function of ETH Driver stm32_eth.h                                    */
/*******************************************************************************************/
void Ethernet_Init()
{
    ETH_InitTypeDef     ETH_InitStructure;
    GPIO_InitTypeDef    GPIO_InitStructure;

    //Select Media Interface
#ifdef OLIMEX_VERSION
    GPIO_ETH_MediaInterfaceConfig(GPIO_ETH_MediaInterface_RMII);
#else
    GPIO_ETH_MediaInterfaceConfig(GPIO_ETH_MediaInterface_MII);
#endif
    /* set PLL3 clock output to 50MHz (25MHz /5 *10 =50MHz) */
    RCC_PLL3Config(RCC_PLL3Mul_10);
    /* Enable PLL3 */
    RCC_PLL3Cmd(ENABLE);
    /* Wait till PLL3 is ready */
    while (RCC_GetFlagStatus(RCC_FLAG_PLL3RDY) == RESET)
    {}
#ifdef OLIMEX_VERSION
    /* Get clock PLL3 clock on PA8 pin */
    RCC_MCOConfig(RCC_MCO_PLL3CLK);

    /* MCO pin configuration------------------------------------------------- */
    /* Configure MCO (PA8) as alternate function push-pull */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
```

```
  /* ETHERNET pins configuration */
  /* AF Output Push Pull:
  - ETH_RMII_MDIO:    PA2
  - ETH_RMII_MDC:     PC1
  - ETH_RMII_TX_EN:   PB11
  - ETH_RMII_TXD0:    PB12
  - ETH_RMII_TXD1:    PB13
  - ETH_RMII_PPS_OUT: PB5
  */

  /* Configure PA2 as alternate function push-pull */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  /* Configure PC1, PC2 and PC3 as alternate function push-pull */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  /* Configure PB5, PB8, PB11, PB12 and PB13 as alternate function push-pull */
  GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_11 |
  GPIO_Pin_12 | GPIO_Pin_13;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  /*****************************************************************/
  /*               For Remapped Ethernet pins                      */
  /*****************************************************************/
  /* Input (Reset Value):
  - ETH_RMII_REF_CLK: PA1
  - ETH_RMII_CRS_DV: PD8
  - ETH_RMII_RXD0: PD9
  - ETH_RMII_RXD1: PD10
  */

  /* ETHERNET pins remapp in STM3210C-EVAL board: RX_DV and RxD[3:0] */
  GPIO_PinRemapConfig(GPIO_Remap_ETH, DISABLE);

  /* Configure PA0, PA1 and PA3 as input */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_7 ;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  /* Configure PB10 as input */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  /* Configure PC3 as input */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  /* Configure PD8, PD9, PD10, PD11 and PD12 as input */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
```

```
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOC, &GPIO_InitStructure); /**/
#else
  //RCC_MCOConfig(RCC_MCO_HSE);
    /* ETHERNET pins configuration */
    /* AF Output Push Pull:
    - ETH_MII_MDIO:     PA2
    - ETH_MII_MDC:      PC1
    - ETH_MII_TXD2:     PC2
    - ETH_MII_TX_EN:    PB11
    - ETH_MII_TXD0:     PB12
    - ETH_MII_TXD1:     PB13
    - ETH_MII_PPS_OUT:  PB5
    - ETH_MII_TXD3:     PB8
        Input (Reset Value):
    - ETH_MII_CRS CRS:  PA0
    - ETH_MII_RX_CLK:   PA1
    - ETH_MII_COL:      PA3
    - ETH_MII_RX_DV:    PD8
    - ETH_MII_TX_CLK:   PC3
    - ETH_MII_RXD0:     PD9
    - ETH_MII_RXD1:     PD10
    - ETH_MII_RXD2:     PD11
    - ETH_MII_RXD3:     PD12
    - ETH_MII_RX_ER:    PB10
    */

    /* All Pins Speed are 50MHz */
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    /* Configure PA2 as alternate function push-pull */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* Configure PC1, PC2 and PC3 as alternate function push-pull */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    /* Configure PB5, PB8, PB11, PB12 and PB13 as alternate function push-pull */
    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_8 | GPIO_Pin_11 | GPIO_Pin_12 | GPIO_Pin_13;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    /* Configure PA0, PA1, PA3 And PA7 as input */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_3 | GPIO_Pin_7 ;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* Configure PB0, PB1, PB10 as input */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    /* Configure PC3, PC4, PC5 as input */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
#endif
    /* Reset ETHERNET on AHB Bus */
    ETH_DeInit();

    /* Software reset */
    ETH_SoftwareReset();
```

```
    /* Wait for software reset */
    while(ETH_GetSoftwareResetStatus()==SET);

    /* ETHERNET Configuration ------------------------------------------------------*/
    /* Call ETH_StructInit if you don't like to configure all ETH_InitStructure parameter */
    ETH_StructInit(&ETH_InitStructure);

    /* Fill ETH_InitStructure parameters */
    /*-------------------------   MAC   ----------------------------------*/
    ETH_InitStructure.ETH_AutoNegotiation = ETH_AutoNegotiation_Disable;
    ETH_InitStructure.ETH_LoopbackMode = ETH_LoopbackMode_Disable;
    ETH_InitStructure.ETH_RetryTransmission = ETH_RetryTransmission_Disable;
    ETH_InitStructure.ETH_AutomaticPadCRCStrip = ETH_AutomaticPadCRCStrip_Disable;
    ETH_InitStructure.ETH_ReceiveAll = ETH_ReceiveAll_Enable;
    ETH_InitStructure.ETH_BroadcastFramesReception = ETH_BroadcastFramesReception_Disable;
    ETH_InitStructure.ETH_PromiscuousMode = ETH_PromiscuousMode_Disable;
    ETH_InitStructure.ETH_MulticastFramesFilter = ETH_MulticastFramesFilter_Perfect;
    ETH_InitStructure.ETH_UnicastFramesFilter = ETH_UnicastFramesFilter_Perfect;
    ETH_InitStructure.ETH_Mode = ETH_Mode_FullDuplex;
    ETH_InitStructure.ETH_Speed = ETH_Speed_100M;

    unsigned int PhyAddr;

#ifdef OLIMEX_VERSION
    for(PhyAddr = 1; 32 >= PhyAddr; PhyAddr++)
    {
        if((0x0006 == ETH_ReadPHYRegister(PhyAddr,2)) &&
           (0x1c50 == (ETH_ReadPHYRegister(PhyAddr,3)&0xFFF0))) break;
    }
#else
    for(PhyAddr = 1; 32 >= PhyAddr; PhyAddr++)
    {
        if((0x1C04 == ETH_ReadPHYRegister(PhyAddr,2)) &&
           (0x0010 == (ETH_ReadPHYRegister(PhyAddr,3)&0xFFF0))) break;
    }
#endif

    /* Configure Ethernet */
    ETH_Init(&ETH_InitStructure, PhyAddr);
}

/*******************************************************************************************/
/* Name : Init Timer                                                                       */
/* Description : Use Function of TIM Driver stm32f10x_tim.h                                */
/*******************************************************************************************/
void TIM_Init()
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    NVIC_InitTypeDef        NVIC_InitStructure;

    /* Initialise data. */
    TIM_DeInit( TIM5 );
    NVIC_Init(&NVIC_InitStructure);
    TIM_TimeBaseStructInit( &TIM_TimeBaseStructure );

    /* Time base configuration for timer 5 - which generates the interrupts. */

    TIM_TimeBaseStructure.TIM_Period = (configCPU_CLOCK_HZ / 32000) & 0xFFFF;
    TIM_TimeBaseStructure.TIM_Prescaler = 0;
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit( TIM5, &TIM_TimeBaseStructure );
    TIM_ARRPreloadConfig( TIM5, ENABLE );
```

```
    /* Enable TIM5 IT.  */
    NVIC_InitStructure.NVIC_IRQChannel = TIM5_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init( &NVIC_InitStructure );
    TIM_ITConfig( TIM5, TIM_IT_Update, ENABLE );

    /* Initialization Timer Value */
    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;
    TIM_TimeBaseStructInit( &TIM_TimeBaseInitStruct );
    TIM_TimeBaseInitStruct.TIM_ClockDivision = 0;
    TIM_TimeBaseInitStruct.TIM_Period = PERIOD;
    TIM_TimeBaseInitStruct.TIM_Prescaler = 0;
    TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Down;
    TIM_TimeBaseInit( TIM3, &TIM_TimeBaseInitStruct );
#ifdef OLIMEX_VERSION
    TIM_TimeBaseInit( TIM4, &TIM_TimeBaseInitStruct );
#else
    TIM_TimeBaseInitStruct.TIM_RepetitionCounter = 0xF;
    TIM_TimeBaseInit( TIM1, &TIM_TimeBaseInitStruct );
#endif

    /* Initialization PWM Value */
    TIM_OCInitTypeDef TIM_OCInitStruct;
    TIM_OCStructInit( &TIM_OCInitStruct );
    TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Disable;
    TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
    TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OCInitStruct.TIM_Pulse = 0;

#ifdef OLIMEX_VERSION
    TIM_OC3Init( TIM3, &TIM_OCInitStruct );
    TIM_OC4Init( TIM4, &TIM_OCInitStruct );
#else
    TIM_OC1Init( TIM3, &TIM_OCInitStruct );
    TIM_OC2Init( TIM3, &TIM_OCInitStruct );
    TIM_OC3Init( TIM3, &TIM_OCInitStruct );

    TIM_OC1Init( TIM1, &TIM_OCInitStruct );
    TIM_OC2Init( TIM1, &TIM_OCInitStruct );
    TIM_OC4Init( TIM1, &TIM_OCInitStruct );

    TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStruct.TIM_Pulse = PERIOD>>1;
    TIM_OC1Init( TIM1, &TIM_OCInitStruct );
#endif
    /* Init DMA for all Timer */
    TIM_DMAConfig(TIM3, TIM_DMABase_CCR1, TIM_DMABurstLength_5Transfers);
    TIM_DMACmd(TIM3, TIM_DMA_CC1, ENABLE);
    TIM_Cmd( TIM3, ENABLE );
#ifdef OLIMEX_VERSION
    TIM_DMAConfig(TIM4, TIM_DMABase_CCR1, TIM_DMABurstLength_5Transfers);
    TIM_DMACmd(TIM4, TIM_DMA_CC1, ENABLE);
    TIM_Cmd( TIM4, ENABLE );
#else
    TIM_DMAConfig(TIM1, TIM_DMABase_CCR1, TIM_DMABurstLength_5Transfers);
    TIM_DMACmd(TIM1, TIM_DMA_CC1, ENABLE);
    TIM_Cmd( TIM1, ENABLE );
    TIM_CtrlPWMOutputs( TIM1, ENABLE);
#endif

    /* uIP stack main loop */
```

```
    TIM_Cmd( TIM5, DISABLE);
}

/*******************************************************************************************/
/* Name :        GPIO Initialization                                                       */
/* Description : Initialize all Input/Output use stm32f10x_gpio.h                         */
/*******************************************************************************************/
void Output_Init()
{
    GPIO_InitTypeDef GPIO_InitStructure;

    /*Initialization Stereo Output*/
    initPort(stereo_6V_m, GPIO_Mode_Out_PP);
    initPort(stereo_4V_m, GPIO_Mode_Out_PP);
    initPort(stereo_2V_m, GPIO_Mode_Out_PP);
    initPort(stereo_0V_m, GPIO_Mode_Out_PP);

    initPort(stereo_6V_p, GPIO_Mode_Out_PP);
    initPort(stereo_4V_p, GPIO_Mode_Out_PP);
    initPort(stereo_2V_p, GPIO_Mode_Out_PP);
    initPort(stereo_0V_p, GPIO_Mode_Out_PP);

    portClr(stereo_6V_m);
    portClr(stereo_4V_m);
    portClr(stereo_2V_m);
    portSet(stereo_0V_m);

    portClr(stereo_6V_p);
    portClr(stereo_4V_p);
    portClr(stereo_2V_p);
    portSet(stereo_0V_p);

    /* All Pins Speed are 50MHz */
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;            // Alt Function - Push Pull

    /* Select Output Pin for PWM */
#ifdef OLIMEX_VERSION

    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_8;
    GPIO_Init( GPIOC, &GPIO_InitStructure );
    GPIO_PinRemapConfig( GPIO_FullRemap_TIM3, ENABLE );        // Map TIM3_CH3 to GPIOC.Pin8
    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_15;
    GPIO_Init( GPIOD, &GPIO_InitStructure );
    GPIO_PinRemapConfig( GPIO_Remap_TIM4, ENABLE );        // Map TIM3_CH3 to GPIOC.Pin8

#else
    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9;
    GPIO_Init( GPIOC, &GPIO_InitStructure );
    GPIO_PinRemapConfig( GPIO_FullRemap_TIM3, ENABLE );        // Map TIM3_CH3 to GPIOC.Pin8

    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 | GPIO_Pin_11;
    GPIO_Init( GPIOA, &GPIO_InitStructure );

    initPort(GPIOA, 10, GPIO_Mode_Out_PP);
    portSet(GPIOA, 10);
    initPort(GPIOC, 9, GPIO_Mode_Out_PP);
    portSet(GPIOC, 9);
#endif
}

/*******************************************************************************************/
/* Name :        Main Function                                                             */
/* Description : Call at the Beginning and call all Initialize Function                    */
```

```c
/*******************************************************************************************/
int main(void)
{
    /* Setup STM32 system (clock, PLL and Flash configuration) */
    SystemInit();

    /* Create the uIP task.  The WEB server runs in this task. */
    /*xTaskCreate(Ethernet_Test,                          // function to start as thread
                (const signed char*) "EthernetApp",       // thread name (just for
debugging)
                (size_t) 2*mainBASIC_WEB_STACK_SIZE,      // stack size
                (void *) NULL,                            // passed as argument to
taskLEDs()
                (unsigned portBASE_TYPE) mainUIP_TASK_PRIORITY,        // task priority
                (xTaskHandle*) NULL);                // can return a handle to created
task*/

    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_OTG_FS | RCC_AHBPeriph_ETH_MAC |
    RCC_AHBPeriph_ETH_MAC_Tx | RCC_AHBPeriph_ETH_MAC_Rx , DISABLE);
    RCC_APB2PeriphClockCmd(~0xFFFF0002,DISABLE);
    RCC_APB1PeriphClockCmd(~(0xC10137C0 | RCC_APB1Periph_USART3),DISABLE);

    /* Start the scheduler. */
    //vTaskStartScheduler();
    RCC_Init();
    Ethernet_Init();
    Output_Init();
    TIM_Init();

    begin = 0;
    indexIn = 0;
    indexOut = 0;
    Volume = 150;
    uIPMain();
}
/*-------------------------------------------------------------------------------------------*/
/************************END OF FILE*****************************************************/
```