UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

MP3-Player mit einem Mikrocontroller

Studienarbeit

vorgelegt von

Steffen Buchner
und
Johannes Schmitz

formatc@uni-koblenz.de, schmitzj@uni-koblenz.de

Betreuer: Dr. M. Joost
Institut für integrierte Naturwissenschaften
Abteilung Physik

Koblenz, im August 2010

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Vorwort

In einer Zeit auf dem Weg zur Rechnerallgegenwart finden immer mehr Kleingeräte ihren Weg in den Alltag. Einen großen Anteil machen dabei Unterhaltungsgeräte aus. Da diese klein und stromsparend sein sollen, bieten sich für viele Teilbereiche Mikrocontroller an. Wo diese Controller an ihre Grenzen stoßen, können sie durch integrierte Schaltkreise unterstützt werden, die bestimmte Teilaufgaben in Hardware realisieren.

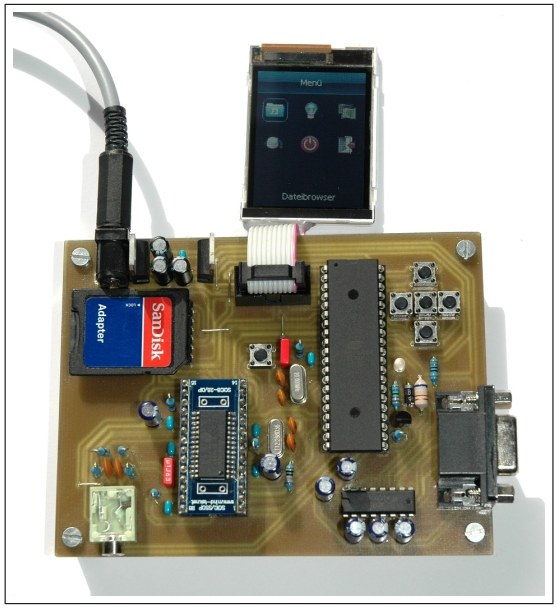


Abbildung 1.1: Die Entwicklungsplatine des MP3-Players

1.2 Inhalt und Ziel der Arbeit

Das Ziel dieser Arbeit ist die Entwicklung eines MP3-Players mit SD-Karten-Slot. Zudem soll er ein Farbdisplay besitzen und über eine komfortable Bedienmöglichkeit verfügen. In dieser Dokumentation wird die verwendete Hardware und ihr Zusammenspiel erklärt. Die für die Realisierung notwendige Software wird vorgestellt und deren Benutzung mithilfe von Codebeispielen verdeutlicht. Insbesondere wird auf die extra für dieses Projekt entworfenen Codebausteine eingegangen. Dies soll die Wiederverwendung der Bausteine in anderen Projekten vereinfachen.

1.3 Verfügbare Lösungen

Im Internet sind bereits einige MP3-Player auf der Basis von ATmega-Controllern in Zusammenspiel mit einem Chip der VS10xx-Reihe zu finden. Eine beim Einstieg besonders hilfreiche Quelle ist [Din06], in der ein Testaufbau mit einer kurzen Software zu finden ist. Ein bereits fertig ausgereifter und vor allem sehr kompakter MP3-Player wurde unter dem Namen yampp (Yet Another MP3 Player) entwickelt und ist in [Han06] zu finden. Ebenso gibt es eine Lösung auf der Basis des AVR Butterflys, welche in [Lot08] zu finden ist. Eine industrielle Platine auf der Basis eines ATmega644p und eines VS1011 ist für 89 Euro[1] auf [EA09a] zu finden. Diese Platine hat aber weder ein Display, noch eine Tastatur.

Um trotz der vielen vorhandenen Lösungen noch ein interessantes Produkt entwickeln zu können, haben wir eigene Bibliotheken entwickelt und dabei eine bedeutungsvolle Alternative zu der bei Heimentwicklungen mit dem Siemens S65-Display geläufigen, jedoch in manchen Punkten nachteilhaften glcd-Library[2] von Christian Kranz und Hagen Reddmann geschaffen.

[1] Preis vom 17.03.2010
[2] Siehe [Kra05]

Kapitel 2

Theoretische Grundlagen

2.1 MP3

MP3 steht für *MPEG-1 Audio Layer 3* und ist ein verlustbehaftetes Kodierungsformat für digitale Audiodaten.

Der Begriff *MP3* ist im umgangssprachlichen Gebrauch fast zum Synonym für Musik in digitalisierter Form geworden. Dies ist wohl vor allem durch die Vorreiterrolle von MP3 als verlustbehaftetes Audioformat mit hoher Kompression begründet. Obwohl es mittlerweile genügend technisch mindestens ebenbürtige Alternativen gibt, kann sich MP3 immer noch gut gegen seine „Konkurrenz“ behaupten: In vielen Online-Musikläden wird es als Standardformat für Musikdownloads genutzt[1]. Alternativen stellen u.a. die Audioformate *AAC* (*Advanced Audio Coding*[2]) oder *OGG Vorbis*[3] dar. Da es sich bei MP3 um ein proprietäres und patentgeschütztes Format bzw. Verfahren handelt, für das entsprechende Lizenzgebühren verrichtet werden müssen, würde sich gerade das lizenz- und patentfreie *OGG Vorbis* Format als Alternative eignen. Viele Hardware-Audioplayer auf dem Markt unterstützen aber nur eines der Audioformate und setzten dabei oft nach wie vor auf MP3.

Für den Anwender sind solche Lizenzaspekte aber letztlich wenig relevant, vielmehr interessiert ihn hingegen der Kompressionsgrad, der sich mithilfe von MP3 erreichen lässt. Sehr deutlich wird dieser beim Umkodieren von Musikstücken von Audio-CDs (16-Bit PCM-Audiodaten[4] mit 44,1kHz Abtastrate) in MP3: Verwendet man hierzu eine konstante Bitrate von 192 Kilobit pro Sekunde (bei gleicher Abtastrate), so benötigt man nur noch rund 14% des Spei-

[1] Z.B. bei Amazon, http://www.amazon.de/mp3/
[2] Standardisiert durch die ISO/IEC Standards ISO/IEC 13818-7 und ISO/IEC 14496-3
[3] Siehe http://xiph.org/vorbis/
[4] PCM: Pulse-code modulation

cherplatzes[5], obwohl die Klangqualität für die meisten Hörer als unverändert empfunden werden sollte.

Dies ermöglicht zusammen mit den stetig günstiger werdenden Preisen von Speicherplatz (vor allem von Flash-Speichermedien), komplette Musiksammlungen für geringe Kosten im mobilen Umfeld ständig zur Verfügung zu haben.

Im folgenden Abschnitt soll ein kurzer Überblick über die Entwicklung der MP3-Kodierung gegeben werden (nach [Fra09a]):

Die Entwicklung des MP3-Codecs geht bis in den Anfang der 1970er Jahre zurück, als Professor Dieter Seitzer an der Friedrich-Alexander Universität Erlangen-Nürnberg beginnt, an der Übertragung von Sprache in hoher Qualität über Telefonleitungen zu forschen. Mit der wenige Jahre späteren Einführung von ISDN und Glasfaserkabel in der Telekommunikation verliert die Verbesserung der Sprachkodierung an Priorität und das Team von Professor Seitzer entscheidet sich, an der Kodierung von Musiksignalen zu forschen. Im Jahr 1979 entwickelt die Forschergruppe einen ersten digitalen Signalprozessor zur Audiokodierung. In diesem Jahr beginnt auch der damalige Student Karlheinz Brandenburg, psychoakustische Prinzipien in Audiokodierverfahren anzuwenden.

Den Beginn der größten Entwicklungsphase erreichte der MP3-Codec im Jahr 1987, als im Rahmen des EU-geförderten Projektes EU147 „EUREKA“ für Digital Audio Broadcasting (DAB) eine Forschungsallianz der Universität Erlangen-Nürnberg und des Fraunhofer IIS gebildet wurde. Die Erkenntnisse der Doktorarbeit über den „Optimum Coding in the Frequency Domain“-Algorithmus von Karlheinz Brandenburg im Jahr 1989 bildeten eine weitere Grundlage für die Weiterentwicklung des MP3-Codecs.

Seitdem ist es „[...] dank unermüdlicher Entwicklungsarbeit [...] letztlich zu dem [geworden], was es heute ist: Ein kulturelles Phänomen made in Germany.“ (Heinz Gerhäuser, MP3-Entwickler, siehe [Fra09b]).

Die wichtigste Idee hinter dem MP3-Codec ist, die Teile der Musik, die das menschliche Ohr kaum wahrnimmt, ungenauer darzustellen und diejenigen, die gar nicht wahrgenommen werden, komplett zu entfernen. Gut hörbare Anteile der Musik werden hingegen sehr genau und möglichst originalgetreu dar-

[5] Berechnet sich durch: $(192 * 1000)/(44100 * 16 * 2)$

gestellt. Diese Idee entstammt dem Wissenschaftsbereich der Psychoakustik. Ein gutes Beispiel aus [Fra09b] ist folgendes: Hört man ein leises Flötenspiel und es wird gleichzeitig kräftig in eine Trompete geblasen, so kann das Ohr das Flötenspiel nicht mehr wahrnehmen. Die Flöte wird durch die Trompete „maskiert“. Zu beachten ist bei der Nutzung dieser psychoakustischen Effekte, dass die akustische Wahrnehmung von Mensch zu Mensch variiert und dadurch ebenso der Klangeindruck von MP3-Kodierungen generell unterschiedlich empfunden werden kann.

2.2 MP3 als Dateiformat

MP3 kodiert Audiodaten in sogenannten *elementary streams* (ER), die durch *MPEG-1* spezifiziert werden[6]. Ein solcher Datenstrom besteht aus einer Aneinanderreihung von *frames*. Ein frame wiederum besteht dabei aus einem 4 Byte großen *header* und einem *Datenteil*. Die frames können dabei jeweils mit unterschiedlichen Bitraten kodiert werden, wodurch zusätzliche Kodierungsformen wie *variable bitrate* (VBR) oder *average bitrate* (ABR) möglich sind. Außerdem existieren erweiterte Kodierungsformen für die Stereokanäle, z.B. das *mid/side stereo* Verfahren (oft auch einfach als *joint stereo* verallgemeinert): Hierbei wird ein Mittelwert-Kanal gespeichert sowie jeweils die Differenz des linken und rechten Stereokanals zu diesem Mittelwert. Da man meist eine relativ starke Korrelation beider Stereokanäle annehmen kann (d.h. kleine Differenzwerte), lässt sich auf diese Art die Kompression oft zusätzlich verbessern.

Beim Dekodieren müssen zur fehlerfreien Wiedergabe (ohne Störgeräusche) die frame-Grenzen eingehalten werden, d.h. der Codec benötigt jeweils einen kompletten frame.

Eine MP3-Datei enthält im einfachsten Fall den kompletten elementary stream. Oft befinden sich vor oder nach diesem Datenstrom noch Metadaten (*ID3-Tags*, siehe Abschnitt 2.3). Die gängige Dateiendung ist *.mp3*.

[6] in ISO/IEC-11172-1

2.3 ID3

Das Dateiformat MP3 beinhaltet nur MP3-spezifische Rohdaten bestehend aus Frames, jeweils mit einem Header und einem Datenblock. Dieser Datenstrom kann so direkt an Hard- oder Software-Decoder weitergereicht werden, da er keinerlei dateispezifische Informationen enthält.
Durch die Beliebtheit von MP3 zur komprimierten Speicherung von Musikstücken kam jedoch schnell das Bedürfnis auf, auch Metainformationen über den enthaltenen Datenstrom in der MP3-Datei selbst unterbringen zu können [Wik10]. Bis dahin waren Annotationen im Dateinamen die einzige Möglichkeit, Metainformationen mit der MP3-Datei selbst zu verbinden.

Diesem vom MP3-Dateiformat nicht abgedeckten Aspekt wurde Mitte der 1990er Jahre pragmatisch durch das Anfügen von sogenannten *ID3*-Tags entgegnet [Wik10]. Ein ID3-Tag ist dabei lediglich ein Datenblock bestehend aus Textfeldern fester Länge, der an das Ende der MP3-Datei angefügt wird. Durch die Platzierung am Dateiende werden Irritationen bei der Dekodierung weitestgehend vermieden: Der Decoder kann den Datenstrom regulär dekodieren und verwirft im Normalfall den ID3-Block (da er keine MP3-Frame Kennzeichnung bestehend aus der Bitfolge FF-FB besitzt), ansonsten wird ein nur relativ kurzes Störsignal am Ende erzeugt. Gleichzeitig lassen sich die Metainformationen von auslesender Software leicht auffinden.

Aber auch ID3-Tags weisen Unzulänglichkeiten auf, die sich in der Praxis ebenfalls schnell bemerkbar machten: Zum einen sind die Längen der Textfelder begrenzt (30 Zeichen in den meisten Fällen), zum anderen sind nur sechs Eigenschaften des Musikstückes annotierbar (Titel, Interpret, Album, Erscheinungsjahr, Kommentar und Genre). Zudem ist die Platzierung der Metadaten am Dateiende nachteilig für Streaming-Anwendungen. Daher entwickelte sich schnell ein neues Format zum De-Facto-Standard für MP3-Metadaten-Annotation: Der von Martin Nilsson entwickelte *ID3v2*-Tag [ea09b].

Der Aufbau unterscheidet sich dabei grundlegend vom einfachen ID3-Tag (im Zusammenhang mit ID3v2 wird dieser auch häufig als ID3v1 bezeichnet): Statt aus einer festen Anzahl fixer Felder besteht ein ID3v2-Tag aus einem Header, gefolgt von einer beliebigen Anzahl von unterschiedlichen Frames (begrenzt auf eine maximale Gesamtgröße). Diese Frames sind dabei unterschiedlich beschaffen, es existieren u.a. einfache Textfelder und Binärdatenfelder mit definierter Semantik. Im Gegensatz zu ID3v1-Tags, in denen Zeichenketten meist in ASCII oder ISO-8859-1 kodiert werden, erlaubt ID3v2 für Text-

felder unterschiedliche Kodierungen innerhalb eines Tags, neben ISO-8859-1 auch Unicode (UTF-8 und teils UTF-16, je nach ID3v2 Version). Außer Textfeldern existieren beispielsweise Frames zur Einbettung von Bildern, empfohlenen Einstellungen für Equalizing und Hall oder auch Informationen über den MP3-Encoder und dessen beim Kodieren verwendete Einstellungen.
Ein weiterer grundlegender Unterschied zu ID3v1 liegt in der Positionierung des Tags innerhalb der Datei: Generell kann ein ID3v2-Tag überall zwischen den MP3-Frames eingebettet werden, in der Praxis befindet er sich aber in fast allen Fällen am Dateianfang (was wiederum dem Streaming-Einsatz zu Gute kommt, da Metainformationen vor dem Senden des eigentlichen Datenstroms übertragen werden können). Damit man beim Editieren eines vorhandenen ID3v2-Tags nicht jedes Mal eine neue Datei anlegen muss, falls sich die Größe des Tags erhöht und er dadurch nicht mehr ohne Weiteres abänderbar ist, fügt man meist eine gewisse Anzahl an Null-Bytes an das Ende des ID3v2 Tags an. Eine gängige Größe für dieses *Padding* ist oft 4048 Byte. Dadurch bleibt der Aufwand für Änderungen an Textfeldern in den meisten Fällen relativ gering.

Ein Großteil der Software, die Metadaten in MP3-Dateien speichert, verwendet dazu ID3v1 und ID3v2-Tags parallel, um eine möglichst gute Kompatibilität zu erreichen. Die auslesende Software muss zunächst prüfen, ob einer der beiden Tags vorhanden ist. Bei ID3v2-Tags kann sie die Metainformationen Frame für Frame auslesen und dabei unbenötigte Frames überspringen, was einen relativ schnellen Zugriff ermöglicht. Um die Zugriffszeit weiter zu verbessern schlägt die ID3v2-Spezifikation vor, wichtigere Informationen (d.h. solche, die gewöhnlich von auslesender Software zuerst benötigt werden) möglichst an den Anfang des Tags zu platzieren.

Kapitel 3

Hardware

3.1 Einleitung

Im diesem Abschnitt soll auf die Hardwarekomponenten, die zur Realisierung des MP3-Players verwendet wurden, sowie deren Zusammenspiel eingegangen werden. Außerdem wird die zur Entwicklung verwendete Hardwareplatine beschrieben.

Neben den nachfolgend beschriebenen Hardwarekomponenten ist vor allem das *Serial Peripheral Interface (SPI)*, das verwendete Bus-System zur Kommunikation zwischen den Komponenten, als wichtiger Hardwarebestandteil hervorzuheben. SPI ist eine ursprünglich von Motorola entwickelte Bus-Technologie, die die Kommunikation der Teilnehmer über drei gemeinsame Leitungen (Takt, Ein- und Ausgangsleitung des Masters) sowie sternförmig zum Master verbundene *Slave Select* Leitungen für jeden Bus-Teilnehmer realisiert. Der verwendete Mikrocontroller ATmega644 unterstützt neben dem SPI-Bus-System auch das I^2C Bus-System. Die Auswahl des verwendeten Bus-Systems wurde jedoch durch das verwendete Display eingeschränkt, das nur am SPI-Bus zu betreiben ist.

3.2 Konzept

Die in dieser Arbeit vorgestellte Platine diente als Entwicklungsplatine für das Projekt und bietet deshalb eine RS232-Schnittstelle, die als Hilfe beim Debuggen und als Möglichkeit zum schnellen Beschreiben des Programmspeichers dient. Dazu kam der Bootloader von Peter Dennegger mit dem Namen *FastBoot*[1] zum Einsatz, der problemlos das „Flashen“ mit einer Geschwindigkeit bis zu 115200 Baud ermöglicht und die Verifikation mithilfe einer CRC-32-Prüfsumme durchführen kann. Der Bootloader ist für alle gängigen ATmegas verfügbar. Ein weiterer entscheidender Punkt beim Entwurf der Platine war der Wunsch nach einfacher Produzierbarkeit auch für solche mit wenig Löterfahrung, weshalb weitgehend auf SMD-Komponenten verzichtet wurde und stattdessen ICs im DIP-Gehäuse, oder falls nicht anders möglich im SOIC-Gehäuse, eingesetzt wurden. Doch auch das Löten von SOIC-Gehäusen sollte mit entsprechendem Werkzeug kein großes Problem darstellen.

Die wichtigsten Komponenten der Hardware sind der Mikrocontroller selbst, ein IC zur Dekodierung der MP3-Daten, eine SD-Karte und ein Display, welche in Kapitel 3.3 genauer erklärt werden. Da diese Komponenten jedoch verschiedene Spannungspegel benötigen (5 V und 3,3 V) mussten zwei Spannungsregler zum Einsatz kommen. Die Auswahl fiel auf die beiden Low-Drop Spannungsregler *LF50CV* und *LF33CV*, welche hintereinander geschaltet wurden (Ausgangsspannung des LF50CV als Eingangsspannung den LF33CV). Jedoch sind damit noch nicht alle Probleme gelöst, da der Spannungspegel auch bei der Kommunikation zwischen den Komponenten eine Rolle spielt. Hier kommt der in Kapitel 3.3.3 vorgestellte Pegelwandler *HC4050* zum Einsatz, der neben dem IC zur Dekodierung der MP3-Daten die einzige Komponente im SOIC-Package ist. Zur SPI-Kommunikation vom Mikrocontroller zu den Komponenten werden deren Input-Pins zum Pegelwandler geführt, während ihre Ausgangs-Pins direkt mit dem ATmega verbunden sind, da dieser auch 3,3 V bereits als High-Pegel erkennt.

Der ATmega wurde mit einem 20 MHz-Quarz beschaltet, um eine flüssige Wiedergabe von hochauflösender Musik und das gleichzeitige Blättern durch die Menüs zu ermöglichen. Auf dem Entwicklungsboard ist zusätzlich ein Reset-Schalter angebracht, der benötigt wird, um zurück in den Bootloader zu springen. Eine am ATmega angebrachte zweifarbige LED (rot/grün) ermöglicht zusätzlich das Informieren über den aktuellen Status (bspw. Bootloader, Fehler, ...), für den Fall, dass das LCD nicht richtig funktioniert.

[1] Siehe [Don J]

Der Decoder-Chip mit dem Namen *VS1011* hingegen wurde mit einem 12,228 MHz-Quarz beschaltet, sodass mit dem internen Taktverdoppler die benötigte Taktrate von 24,456 MHz erreicht wird. Der IC benötigt zusätzlich noch einige Kondensatoren zur Spannungsstabilisierung sowie einige Bauteile für den analogen Schaltungsteil, d.h. für Endstufe und Kopfhörerbuchse.

Das LCD-Display wird wie die anderen Komponenten über den Pegelwandler verbunden, benötigt jedoch zwei Spannungsversorgungen. Eine davon dient zur Hintergrundbeleuchtung und beträgt etwa mindestens 8,5 V. Diese Spannung wird mithilfe der in Kapitel 3.3.5 gezeigten Schaltung generiert.

Zur Steuerung sind 5 Taster angebracht, von denen vier zur Navigation durch die Menüs und ein zentrierter Taster als Bestätigungsknopf dient.

3.3 Hauptbestandteile

3.3.1 ATmega644

Der 8-Bit RISC Mikrocontroller *ATmega644* des Herstellers Atmel bildet das Herzstück des MP3-Players. Er stellt den Master im SPI-Bus-System dar und übernimmt die Ansteuerung aller angeschlossenen Geräte, d.h. des Displays, der SD-Karte und des VS1011e-Decoders.

Die Hauptaufgaben des ATmega644 sind neben der Steuerung des allgemeinen Programmflusses, d.h. der Realisierung von Programmablaufsträngen sowie einem bedienbaren grafischen Menü, das Befördern von MP3-Daten an den Decoder-Chip und die Realisierung eines FAT-Dateisystem-Zugriffs von einer Partition der SD-Karte. Zudem realisiert der Controller eine alphabetisch sortierte Dateiauflistung des FAT-Dateisystems. Eine weitere Aufgabe ist das Auffinden und Auslesen von Metadaten in Form von ID3-Tags vor dem Übermitteln der MP3-Audiodaten an den Decoder. Der Programmspeicher des ATmega644 wird zur Generierung der grafischen Oberfläche auch zur Speicherung von Grafikelementen wie Bildern oder Pixelschriftarten genutzt. Zu Debugging-Zwecken realisiert der ATmega644 über seine USART-Hardware und einen MAX232 IC eine serielle RS232 Kommunikationsschnittstelle.

Ein Großteil der Entwicklung wurde auf dem *ATmega32* Controller durchgeführt, der nur die Hälfte des Programm- und Arbeitsspeichers besitzt. Durch das Hinzufügen umfangreicher Funktionen und benötigter Grafiken war der Programmspeicher jedoch (bei der angestrebten Funktionalität) nicht mehr ausreichend. Aus diesem Grund haben wir uns für den Umstieg auf den pinkompatiblen ATmega644 entschieden. Dieser bietet uns 64 KB Programmspeicher, 2 KB EEPROM und 4 KB SRAM, sowie zwei SPI-Interfaces[1] und genug Input- und Output-Pins, um eine kleine Tastatur anzubringen. Der verfügbare Programmspeicher reicht aus, um einen MP3-Player zu realisieren, bei dem auf kleine Extras wie grafische Menüs nicht verzichtet werden muss. Zudem erlaubt der ATmega644 eine höhere Taktrate von 20 MHz (statt 16 MHz beim ATmega32), was eine etwas schnellere Übertragung der Pixeldaten an das Display ermöglicht. Der MP3-Player kann trotzdem problemlos mit 16 MHz betrieben werden, wodurch generell eine Kompatibilität zum ATmega32 gewährleistet ist.

3.3.2 VS1011e

Der VS1011e des finnischen Herstellers VLSI Solution ist ein MP3-Decoder in Form eines integrierten Schaltkreises. Er ist in der Schaltung zuständig für die Dekodierung der MP3-Daten sowie deren Verstärkung zur Ausgabe auf Kopfhörern oder als Line-Level-Ausgang. Der VS1011e-IC wurde für dieses Projekt ausgesucht, da er vergleichsweise günstig ist und zudem am SPI-Bus betrieben werden kann. Viele andere Lösungen unterstützen nur den I^2C-Bus [mik10]. Da aber für dieses Projekt ohnehin eine SD-Karte per SPI-Bus betrieben wird, fiel dieses Kriterium besonders ins Gewicht, um nicht auf zwei verschiedene Bus-Systeme angewiesen zu sein, da dies ein weiteres Problem bezüglich eines kompakten Layouts dargestellt hätte. Zusätzlich verfügt der IC über einen Bass / Treble Enhancer, mit dem sich Höhen- und Bassanteile des Signals anpassen lassen. Es besteht auch die Möglichkeit, eigene Applikationen auf dem DSP des ICs auszuführen (soweit der Dekoder nicht mit hohen Bitraten ausgelastet ist), welche aber bei diesem diesem Projekt nicht genutzt wurde. Außer der Dekodierung von MP3 wird auch die Wiedergabe von einigen WAV (RIFF) Formaten unterstützt, diese Option wurde aber bisher ebenfalls nicht genutzt.

[1] USART kann im SPI-Modus benutzt werden.

Angesteuert wird der Decoder über zwei getrennte Bus-Schnittstellen, die beide per SPI realisiert sind: Das *Serial Command Interface (SCI)*, das Steuerbefehle entgegennimmt und u.a. Daten über den dekodierten MP3-Datenstrom zur Verfügung stellt, sowie das *Serial Data Interface (SDI)*, das die zu dekodierenden Daten entgegennimmt. Das jeweilige Interface lässt sich über die beiden Pins *XCS* (SPI slave select Pin des SCI) und *XDCS* (slave select des SDI) anwählen. Hierbei ist auch ein *shared mode* möglich, bei dem nur einer der beiden Pins benötigt wird.

Wenn das Serial Data Interface ausgewählt ist, nimmt der VS1011e Audiodaten entgegen, die er in einem internen FIFO-Puffer zwischenspeichert. Empfangsbereitschaft signalisiert der Decoder durch High-Pegel am *DREQ* Pin, er nimmt dann mindestens 32 Bytes an Daten entgegen.

Betrieben wird der IC auf einer Versorgungsspannung von 2,5 bis 3,6 V sowie einem Takt von 24,456 MHz. Für den Takt stellt der VS1011e zudem einen internen Taktverdoppler zur Verfügung, falls benötigt [VLS09].

3.3.3 Pegelwandlung im SPI Bus mittels HC4050

Bei der Konzeption der Schaltung für den MP3-Player ergab sich anfangs folgendes Problem: Die SPI Slaves, d.h. Display, SD-Karte und VS1011-Decoder, erlauben alle nur eine Betriebsspannung in einem Bereich von ca. 3,3 V (maximal 3,6 V). Würde man den ATmega644 als SPI Master mit 3,3 V statt 5 V betreiben, so erhielte man aber nur eine maximale Taktfrequenz von 10 MHz statt möglichen 20 MHz. Die daraus resultierende langsamere SPI Taktfrequenz (5 MHz maximal) wäre für die Ansteuerung des Displays sehr von Nachteil, da sich dadurch der Bildaufbau drastisch verlangsamen würde (um eben 50%).

Um den ATmega644 auf seiner maximalen Taktfrequenz von 20 MHz betreiben zu können, entschlossen wir uns, den ATmega644 auf 5 V zu betreiben (die SPI Slaves dabei weiterhin auf 3,3 V) und die Pegel der SPI-Ausgangspins (MOSI, SCK sowie alle Slave Select Pins) mit Hilfe von HD4050 CMOS TTL-Treiber ICs von 5 V auf 3,3 V zu reduzieren. Der HD4050 Treiber-IC hat dabei die wichtige Eigenschaft, dass er Eingangspegel toleriert, die über seiner Betriebsspannung von hier 3,3 V liegen. Zudem schaltet er schnell genug, um die geforderte SPI-Taktfrequenz zu realisieren. Die niedrigen Ausgangsspannungen der SPI Slaves stellen für den ATmega644 Controller kein Problem dar, da dieser die Spannungen von 3,3 V als High-Pegel erkennt.

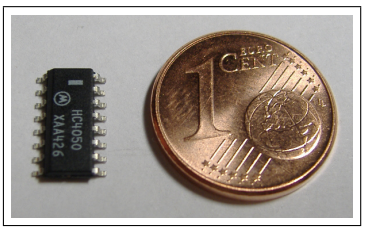


Abbildung 3.1: Foto eines SMD HC4050

3.3.4 UART über MAX232

Zu Debugging-Zwecken wurde in die Entwicklungsversion der Hardware eine serielle Schnittstelle integriert, die eine einfache Anbindung der Testplatine an einen PC ermöglicht.

Der ATmega644 verfügt über eine *Universal Synchronous/Asynchronous Receiver Transmitter (USART)* Hardware-Einheit, welche serielle Kommunikation im synchronen oder asynchronen Modus realisiert. Die Spannungspegel unterliegen dabei der Betriebsspannung des ATmega644. Ein MAX232 Pegelwandler IC wird hier ergänzend verwendet, um die 0 V / 5 V Spannungspegel auf -/+ 12 V des RS232 Standards umzuwandeln. Neben dem MAX232 IC selbst werden fünf Kondensatoren zur Spannungsstabilisierung benötigt.

3.3.5 Siemens S65 LCD

Nach einigen Überlegungen fiel unsere Auswahl des Displays auf das des Siemens S65-Handys. Es überzeugt vor allem durch die Farbqualität, jedoch spielte auch der Preis eine entscheidende Rolle. Zum aktuellen Zeitpunkt sind solche Displays für 11,80 Euro[2] zu erhalten. Ebenso war die bereits vorhandene GLCD von Christian Kranz[3] ein Anstoß, dieses Display zu benutzen.

Das S65-Display gibt es mit verschiedenen Chipsätzen. Auf [Kra05] werden drei verschiedene Modelle gezeigt. Diese entscheiden sich markant durch den internen Aufbau und verwenden Controller unterschiedlicher Hersteller. Die Displays sind mit einem Typenschild gekennzeichnet, mithilfe dessen sie sich klassifizieren lassen. Die Typenschilder zeigen die Bezeichnungen LS020xxx, LPH88xxxx oder L2F50xxx. Unsere Wahl fiel auf Displays der LS020-Reihe, da für diese bereits eine Ansteuerung mit Grafikfunktionalität sowohl in Assembler-Sprache als auch C existiert.

[2] Preis vom 06.05.2010
[3] http://www.superkranz.de/christian/S65_Display/DisplaySoftware.html

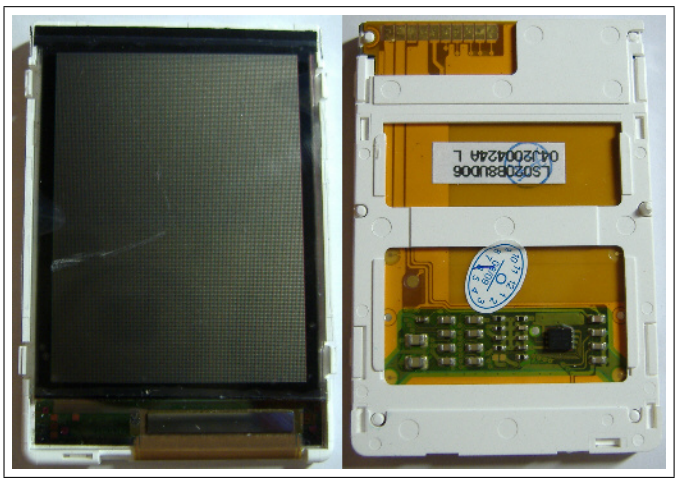


Abbildung 3.2: S65-LCD mit LS020xxx-Controller

Das Display besitzt die Maße 38 * 55 * 3 mm, die sichtbare Fläche ist 31 * 42 mm groß. Die Auflösung beträgt 132 * 176 Pixel zu je 24 Bit Farbtiefe. Es wird über SPI angebunden und kann somit ohne großen Aufwand am ATmega betrieben werden. Da es allerdings eine Versorgungsspannung von etwa 3,3 V benötigt, müssen bei einem mit höherer Voltzahl betriebenen ATmega Pegelwandler vorgeschaltet werden. An dem anfänglich eingesetzten ATmega32L mit einer Betriebsspannung von 3,3 V konnte das Display problemlos direkt am SPI-Bus betrieben werden.

Da zu dem Display keine offenen Dokumentationen oder Datenblätter existieren, mussten die Befehle durch Reverse Engineering in Erfahrung gebracht werden. Einen Meilenstein legte dabei [Kra05], der mithilfe eines digitalen HP (Agilent) 16-Kanal-Oszilloskop einen wichtigen Anteil der Befehle ermittelte und die Befehlssequenzen analysierte. Einige weitere Befehle, von dessen Existenz auszugehen ist, sind jedoch zum aktuellen Zeitpunkt noch unbekannt. So erreicht das Siemens S65 Handy mit dem Display z.B. flüssige Wiedergabe von Videos, trotz ebenfalls serieller Anbindung über SPI Bus, was auf eine „bessere“ Ansteuerung zurück zu führen sein könnte.

Zuletzt ist die eingebaute LED-Hintergrundbeleuchtung zu erwähnen. Diese besitzt allerdings den Nachteil, dass sie eine Versorgungsspannung von etwa 10,4 V bei 20 mA benötigt. Die Spannung kann man jedoch mit einer kleinen Schaltung über die im ATmega integrierte Pulsweitenmodulations-Hardware erzeugen. In unserem Projekt wurde die in [Kra05] vorgestellte Schaltung in leicht abgewandelter Form benutzt (siehe Abbildung 3.3). Mit ihr ist eine Re-

gulation der Hintergrundbeleuchtung in zahlreichen Stufen möglich. Die von uns geschriebene S65-Bibliothek besitzt bereits den Quelltext zur Ansteuerung dieser Schaltung.

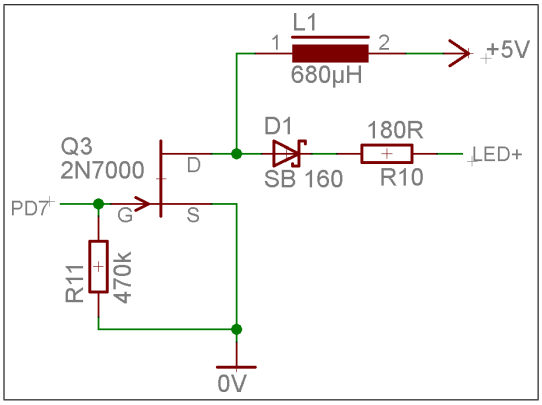


Abbildung 3.3: Schaltung für die Hintergrundbeleuchtung des S65-LCD

Der große Nachteil des S65-LCD besteht wie bereits zuvor erwähnt darin, dass keine offene Dokumentation besteht. Somit wird die Darstellungsgeschwindigkeit wie bei den Siemens-Handys bekannt, vorerst nicht erreicht werden können. Allerdings stellt die neu implementierte Bibliothek zur Ansteuerung des Displays schon einen Fortschritt dar, da mit ihr durch gezielte Reduzierung des SPI-Datenverkehrs ein Geschwindigkeitszuwachs erreicht werden kann. Ein weiterer Nachteil ist das abnehmende Angebot an S65-Displays am Markt.

Weitere ausführliche Informationen sind in einer Arbeit zum Seminar „Machbarkeitsstudie: Einchipcomputer“ unter [Jak07] zu finden.

3.3.6 SD-Karte

Verwendbar sind theoretisch sowohl MMC- als auch SD- und SDHC-Karten. Die SD- und SDHC-Karten werden dabei im Kompatibilitätsmodus für MMC-Karten verwendet, sodass im Quelltext keine gesonderte Unterscheidung nötig ist. Die Karten sind einfach über SPI ansteuerbar und benötigen eine Versorgungsspannung von 3,3 V. Bei Mikrocontrollern mit höherer Versorgungsspannung haben wir zur Erzeugung der Spannung von 3,3 V auf dem SPI erfolgreich Spannungsteiler und HD4050 CMOS TTL-Treiber ICs benutzt. In Richtung von der SD-Karte zum Mikrocontroller wird keine Spannungserhöhung benötigt, da der ATmega eine Spannung von 3,3 V bereits als High-Pegel

erkennt. Als Kartenhalterung haben wir einen microSD-Adapter benutzt, der fest verlötet ist. Somit ist die Benutzung von microSD-Karten in einer Halterung für einen sehr geringen Preis möglich.

Zur Ansteuerung der SD-Karte benutzen wir die MMC/SD/SDHC card Quelltextbibliothek von Roland Riegel [1]. Sie befindet sich nun bereits seit etwa 4 Jahren in aktiver Entwicklung und bietet low-level-Routinen zum Lesen und Beschreiben von MMC-, SD- und SDHC-Karten sowie Unterstützung einer Partitionstabelle und eine einfache FAT16/FAT32-Implementation mit Lese- und Schreibzugriff. Diese Bibliothek wird von unserem MP3-Player in Kombination mit dem selbst geschriebenen FAT File Browser Quelltext-Modul benutzt, um durch (alphabetisch) sortierten Listen von Ordnern und Dateien in beliebiger Richtung iterieren zu können.

Abbildung 3.4: SD-Kartenhalterung

Weitere ausführliche Informationen sind in einer Arbeit zum Seminar „Machbarkeitsstudie: Einchipcomputer“ unter [Max08] zu finden.

[1] Siehe [Rie10]

3.4 Schaltplan und Layout

Abbildung 3.6 zeigt den Schaltplan der Entwicklungsplatine, Abbildung 3.7 das zugehörige Layout. Beide wurden mithilfe der Software EAGLE[1] erstellt. Das Layout ist einlagig ausgelegt, d.h. es werden nur Leiterbahnen auf der Platinenunterseite benötigt. Teilweise wurden auf der Platinenoberseite Drahtbrücken verwendet (angedeutet durch den roten Top-Layer im EAGLE Layout).

Die beiden Spannungsregler *IC1* und *IC2* bilden zusammen mit den Kondensatoren *C1* bis *C4* das Netzteil. Bei beiden Spannungsreglern handelt es sich um Low-Drop-Typen, die mit einer geringen Differenzspannung arbeiten können. Dadurch kann die Platine mit einer Gleichspannung von 5V bis maximal 16V betrieben werden, wobei höhere Spannungen entsprechende Verlustleistungen mit sich bringen. Zu beachten ist der Kondensator *C3* zwischen den beiden Spannungsteilern, der Oszillationen verhindert.

Die Grundbeschaltung des ATmega644 besteht zunächst nur aus dem Abblock-Kondensator *C13*, dem Quarz *Q1* mit Kondensatoren *C10* und *C11* sowie der Reset-Beschaltung (*S1*, *C9* und *R5*). Weitere Peripherie sowie Komponenten sind über die I/O Ports verbunden.

Die Schaltung zur Erzeugung der Display-Hintergrundspannung per Pulsweitenmodulation besteht aus dem FET-Transistor *Q3*, der Spule *L1*, der Zener-Diode *D1* sowie den Widerständen *R10* und *R11*. Die Wahl der Bauteile ist dabei nicht sonderlich kritisch.

Bei der Beschaltung des VS1011e ist auf eine saubere Masseführung zu achten, um Signalkopplungen über die Masseleitung („Brummschleifen“) im Audio-Schaltungsteil zu verhindern [Sol08]. Hierzu wurden die entsprechenden Masseleitungen auf einem Punkt unter dem IC zusammengeführt und zum zentralen Netzteilmassepunkt weiterverbunden.
Die Widerstände *R1* und *R2* der Kopfhörerbeschaltung dienen als zusätzlicher Schutz vor Kurzschlüssen der Endstufe.

Für die Pegelwandler-ICs wurden SMD-Ausführungen gewählt, da dies die Layoutführung deutlich vereinfachte. Die ICs wurden dabei nur jeweils auf einer Gehäuseseite verschaltet, wobei auf jeder Gehäuseseite drei Eingänge und drei Ausgänge vorhanden sind. Somit wurden nur drei der sechs uni-

[1] „Easy Applicable Graphical Layout Editor“, http://www.cadsoft.de/

direktionalen Treiber pro IC verwendet, die ungenutzten Eingänge liegen auf Masse. Abbildung 3.5 zeigt die Pinbelegung eines HC4050. Es könnten in einem anderen Layout also alternativ auch zwei (voll beschaltete) ICs in größeren Bauformen genutzt werden.

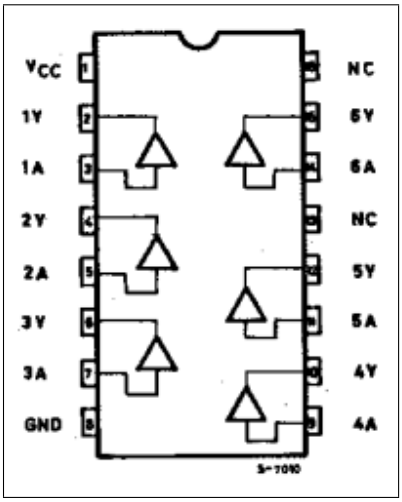


Abbildung 3.5: Pinbelegung eines HC4050

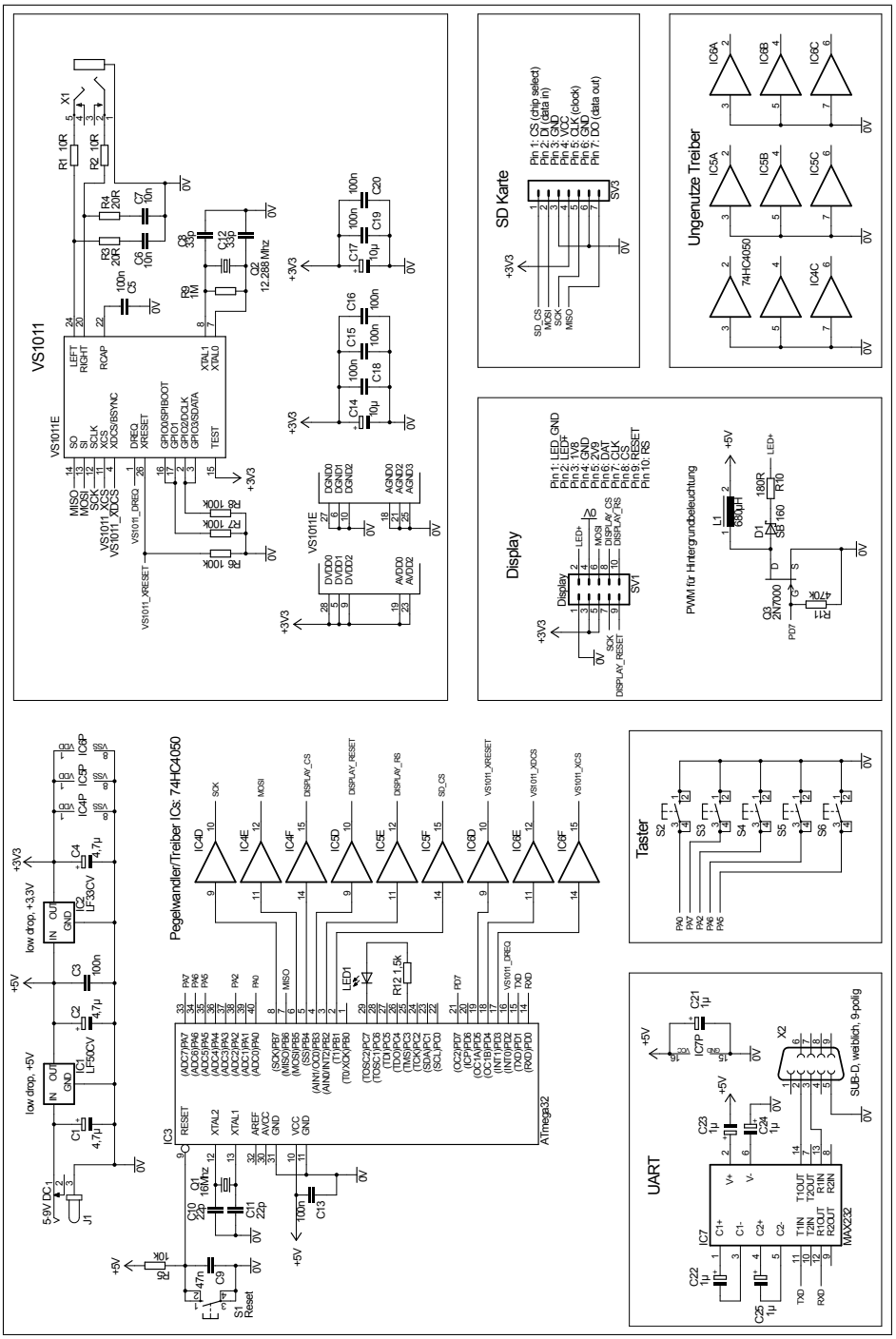


Abbildung 3.6: Schaltplan der Entwicklungsplatine

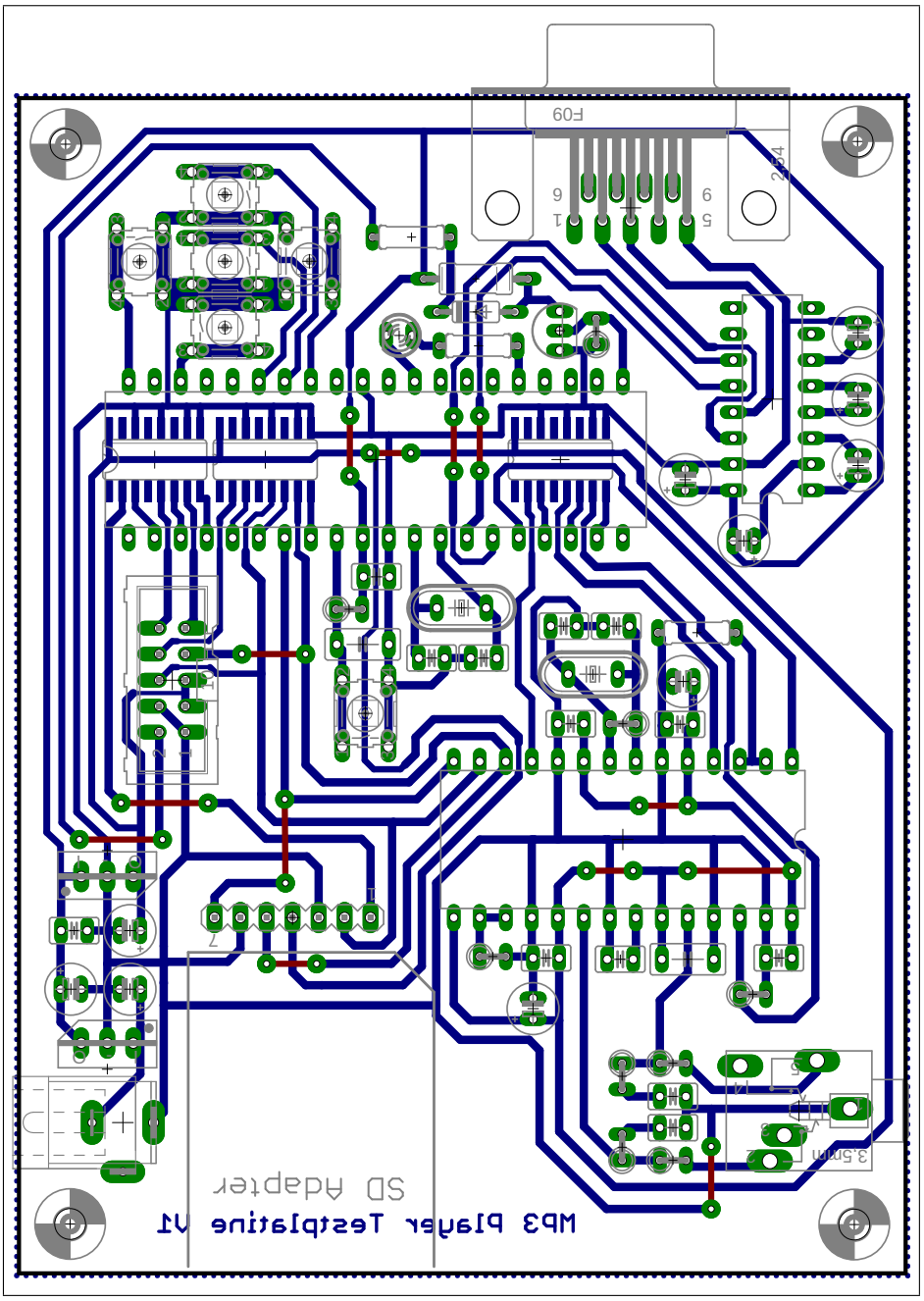


Abbildung 3.7: Layout der Entwicklungsplatine

Kapitel 4

Software

4.1 Einleitung

Die Software des MP3-Players ist komplett in C geschrieben und wurde in mehrere Module aufgeteilt, um Wiederverwendbarkeit und Austauschbarkeit verschiedener Codestücke zu ermöglichen. Besonders interessant ist dort zum Beispiel die LCD-Library, welche auch problemlos in andere Softwareprojekte eingebunden werden kann und in Kombination mit der Bibliothek *GUI-Elements* benutzt werden kann. Ebenso ist eine Bibliothek zum Auslesen der ID3-Tags und zur komfortablen Ansteuerung einer SD-Karte mithilfe der *MMC/SD/SDHC card library* von Roland Riegel [Rie10] enthalten. In Kapitel 4.1 wird die Software im Allgemeinen erklärt, während in Kapitel 4.5 auf die einzelnen Bibliotheken genau eingegangen wird.

4.2 Anforderungsliste

Bei der Erstellung der Software standen folgende Anforderungen im Vordergrund:

- Es müssen MP3-Daten abgespielt werden können.
- Der Player muss die MP3-Dateien von einer SD-Karte laden.
- Es sollen lange Dateinamen verwendet werden können.
- Es muss ein Farb-LCD-Display angesteuert werden können.
- Die Hintergrundbeleuchtung soll regelbar sein und sich automatisch abblenden.
- Die Steuerung soll komfortabel und schnell erlernbar sein.
- Es soll ein Dateibrowser enthalten sein.
- Die Software muss modular aufgebaut sein.
- Die GUI soll farbige grafische Menüs enthalten.
- Möglichst viele Elemente sollen durch Grafiken angezeigt werden.
- Der Player soll ID3-Tags auslesen können.
- Alle Einstellungen müssen im EEPROM abgespeichert werden.
- Alle numerischen Konfigurationen sollen durch einen Regler einstellbar sein.
- Es sollen Funktionen zur komfortablen Erstellung von GUI-Elementen verfügbar sein.
- Die Display-Ansteuerung soll komfortabel sein und soll Funktionen für Linien, Pixel, Rechtecke und Textausgabe beinhalten.

4.3 Architektur

Die Software besteht aus folgenden Komponenten:

Komponente	**Aufgabe**
Main / Global	Bootprozedur inkl. Steuerung der Einrichtung der Hardwarekomponenten, Speichern und Laden der Einstellungen, Steuerung der Interrupts und der Threads, globale Variablen
LCD	Bibliothek zur Ansteuerung des Siemens S65-LCD
ID3	Quelltextbibliothek zum Auslesen von ID3-Tags aus MP3-Daten
GUI	Realisierung der kompletten GUI bis auf die Playeroberfläche, enthält den Hauptzustandsautomat und ist für das Auslösen der Threads verantwortlich
Player	Versorgen des VS1011 mit Daten und Steuerung desselben, Verwaltung der Abspielliste, Bereitstellen von Equalizer-Funktionalitäten
Player-GUI	Oberfläche des Players (außerhalb sonstiger Menüs) inklusive des Player-Minimenüs
VS1011	Quelltextbibliothek zur Ansteuerung des VS1011
File-Browser	Der Dateibrowser innerhalb des MP3-Players zum Durchsuchen der Ordner inklusive dessen GUI
FAT-File-Browser	Backend zum komfortablen Durchsuchen der SD-Karte, alphabetische Sortierung der Dateiinhalte, Verwaltung der kompletten Dateidatenhaltung in Zusammenspiel mit der MMC/SD/SDHC card library von Roland Riegel
EEPROM	Wrapper zur Speicherung der Daten im EEPROM
UART	Stellt über die Standardausgabe eine RS232-Schnittstelle über die UART-Hardware des ATmegas zur Verfügung

Der vereinfachte Graph in Abbildung 4.1 gibt einen Überblick darüber, wie die einzelnen Komponenten voneinander abhängen. Hierzu wurden die wichtigsten Header-Einbindungen über die *#include* Präprozessordirektive betrachtet.

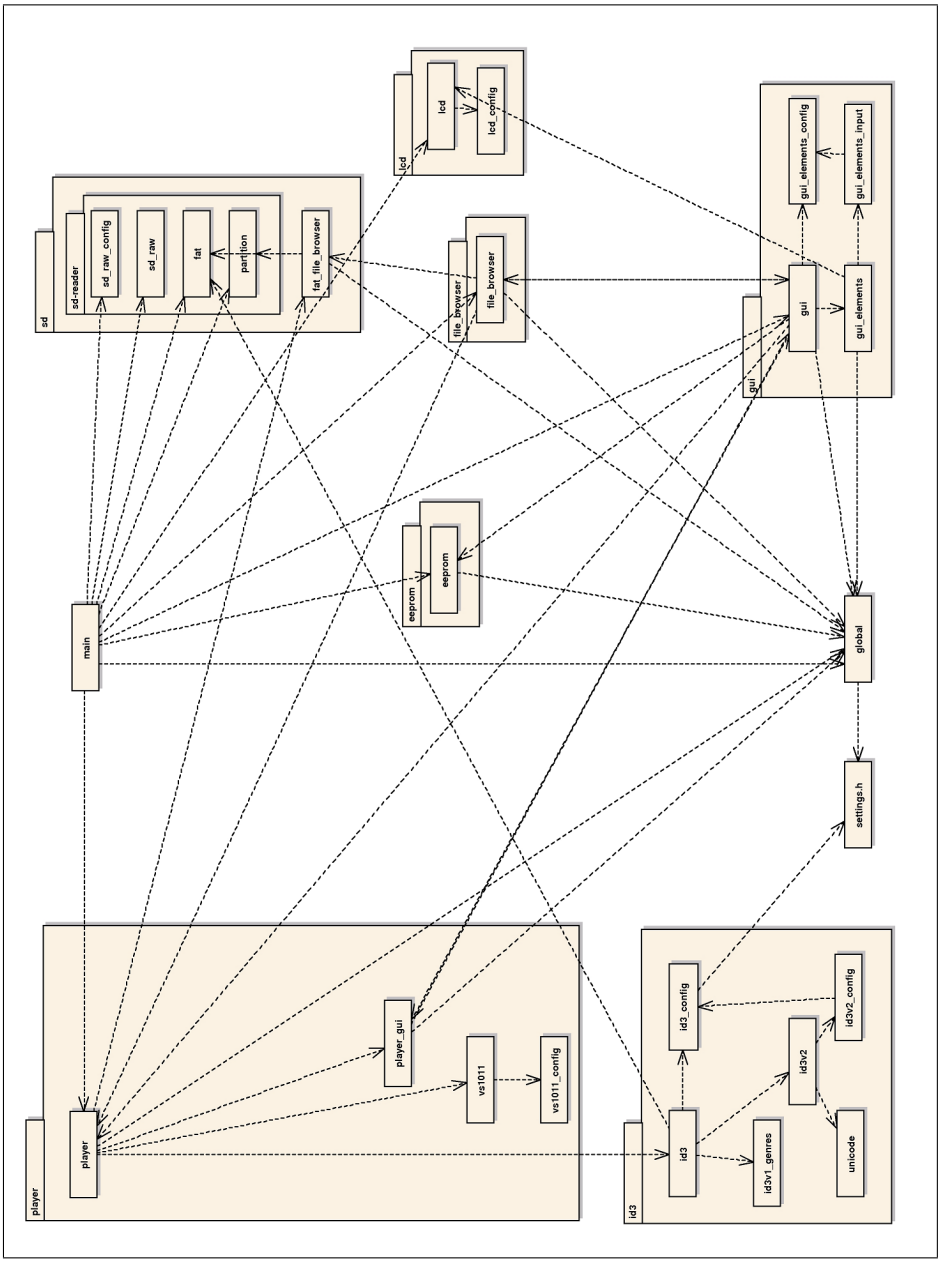


Abbildung 4.1: Graph der Quelltext-Abhängigkeiten

4.4 Zustandsautomaten

Einen wichtigen Teil der Software bilden die beiden enthaltenen Zustandsautomaten. Der Zustand in Abbildung 4.2 ist hauptsächlich für die Benutzerschnittstelle (GUI) zuständig und speichert, welche Ansicht momentan dargestellt wird.

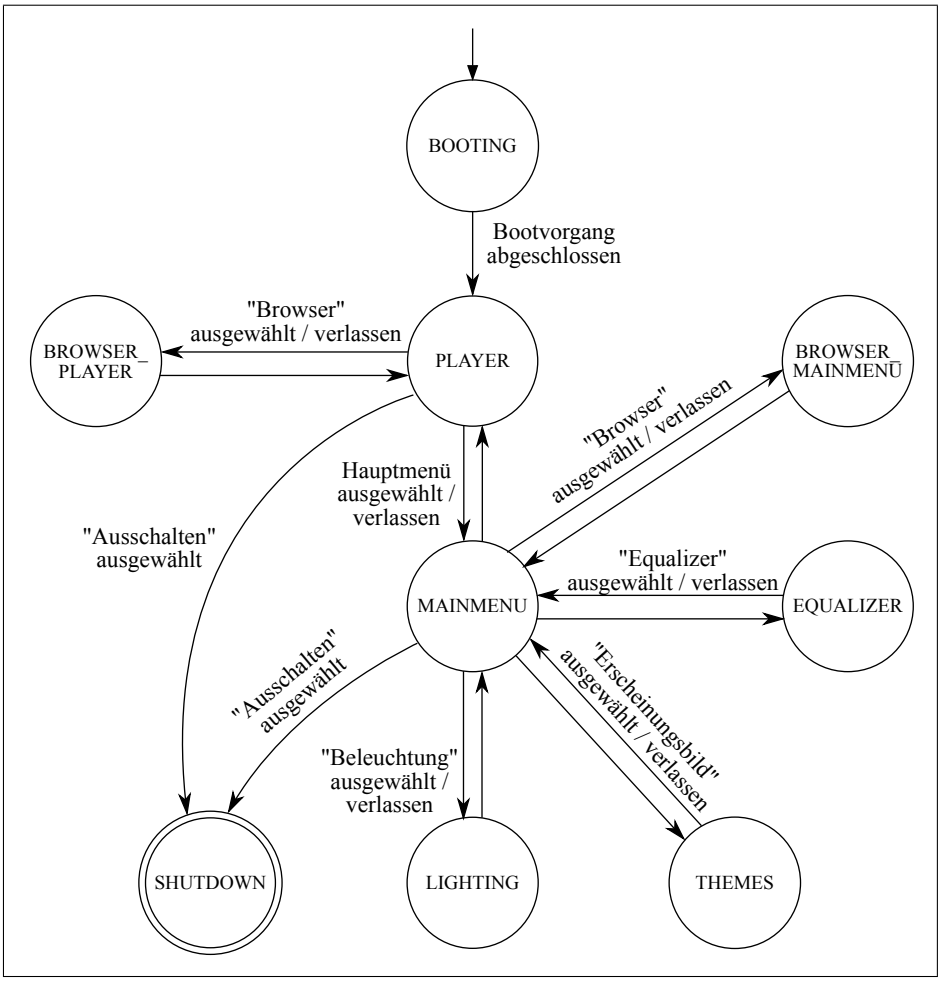


Abbildung 4.2: Zustandsautomat der Benutzerschnittstelle

Die Zustände *BROWSER_PLAYER* und *BROWSER_MAINMENU* repräsentieren beide das Anzeigen des Dateibrowsers innerhalb der Benutzerschnittstelle. Es werden zwei separate Zustände verwendet, damit anhand dieser festgestellt werden kann, zu welchem GUI-Zustand beim Verlassen des Dateibrowsers zurückgekehrt werden soll.

Der zweite Automat speichert das aktuell angezeigte Untermenü der Player-Ansicht (Abbildung 4.3).

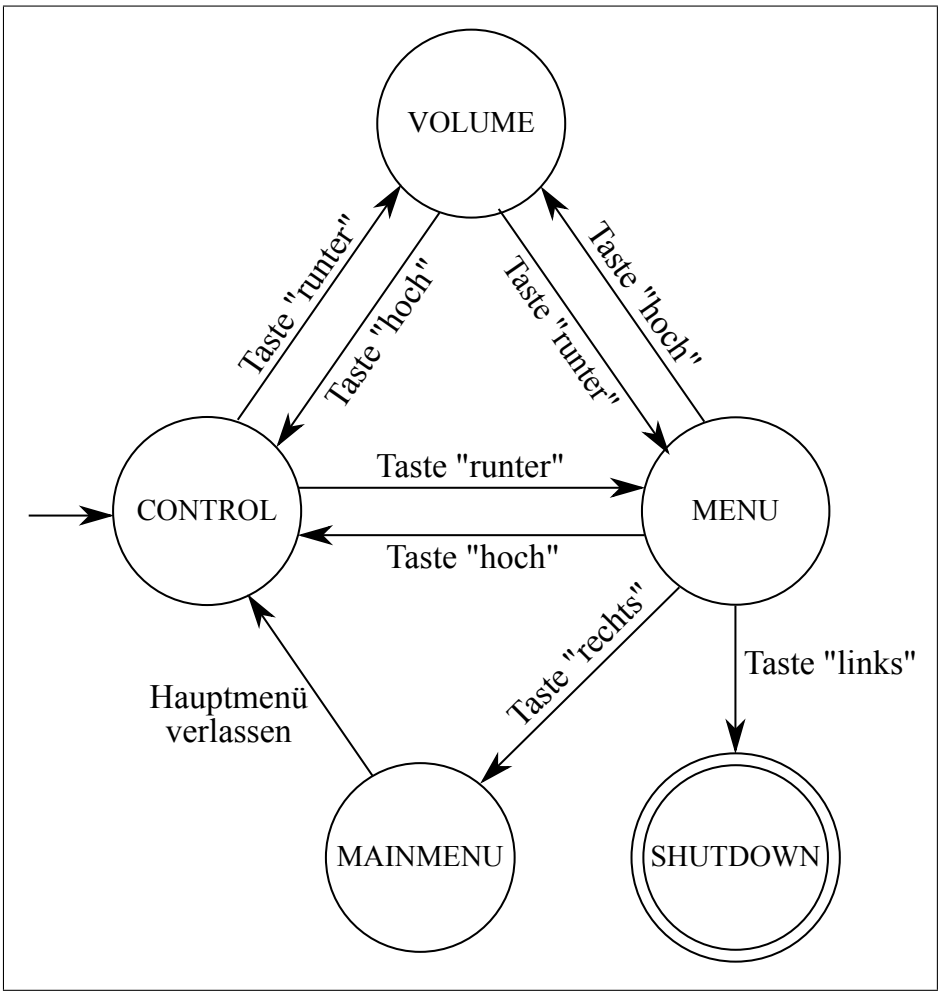


Abbildung 4.3: Zustandsautomat der Player-GUI

Die Zustände *MAINMENU* und *SHUTDOWN* sind dabei Zustände des GUI-Zustandsautomaten und sollen die Interaktion zwischen beiden Automa-

ten andeuten.

4.5 Bibliotheken

4.5.1 LCD

Bei der Entwicklung der Software wurde zur Ansteuerung des Displays zunächst die in Assembler geschriebene *Siemens S65 GLCD* Library von Christian Kranz verwendet [Kra05]. Dabei stellten sich aber schnell einige unvorteilhafte Aspekte heraus:

- Die Konfiguration von Display-spezifischen Pins und SPI-Hardware kann nicht getrennt durchgeführt werden, was eine saubere Eingliederung im Projekt erschwert. Zudem setzt die Library nicht vor jeder SPI-Kommunikation die entsprechende Taktrate neu, dies erschwert die Verwendung zusammen mit anderen SPI-Slaves etwas.
- Die Library unterstützt den 8-Bit-Datenmodus des S65-Displays nicht, auf den die Autoren durch weitere Recherchen aufmerksam wurden[1]. Die Library überträgt pro Pixel zwei Byte per SPI-Bus (bei Nutzung einer 16-Bit-Pixelkodierung). Durch die Nutzung des 8-Bit Datenmodus könnte man für Operationen, die mit weniger Farbtiefe auskommen, die Übertragungsrate verdoppeln. Dies ist vor Allem deshalb relevant, weil die serielle Übertragung der Pixeldaten über den SPI-Bus einen Flaschenhals bei der Aktualisierungsgeschwindigkeit des Display-Inhalts darstellt. Außerdem deckt die Library von Christian Kranz nicht alle bekannten Fähigkeiten des Display-Controllers ab (z.B. das Scrollen von Bildschirmbereichen sowie das Deaktivieren des Displayinhaltes).
- Die Library verfügt über keine Funktionalitäten zur Darstellung von Bildern (was z.B. für Icons und Ähnliches wünschenswert wäre).
- Die Assembler-Implementation macht Erweiterungen und Anpassung des Codes aufwändiger, außerdem ist die Qualität des vorliegenden Codes nur recht mühsam kontrollierbar.

[1] Siehe Ausführungen auf http://www.juras-projects.org/eng/lcd.php

Aus diesem Grund wurde eine eigene (statische) Bibliothek zur Ansteuerung des S65-Displays in C implementiert, die die oben aufgeführten Nachteile beheben sollte. Dabei wurde auf folgende Eigenschaften besonderen Wert gelegt:

- Saubere Konfiguration getrennt für Hardware und Software über entsprechende Header-Dateien
- Getrennte Initialisierung von betroffener Controller-Hardware (I/O Pins, SPI) und LCD-Software (Initialisierung des Display-Controllers etc.)
- Vor jeder SPI-Übertragung soll ggf. die passende SPI-Taktrate eingestellt werden
- Einfache Anwendbarkeit
- Unterstützung von 8 und 16-Bit-Pixelformaten
- Möglichst einfache / kompakte Implementation bei ausreichender Flexibilität

Zur Benutzung muss die Bibliothek in das Projekt eingebunden werden (*liblcd.a*) sowie der Header *lcd.h* inkludiert werden. Zudem existieren zwei weitere Header-Dateien, die der Konfiguration dienen: *lcd_hardware_config.h* enthält Präprozessorkonstanten und Makros zur Definition der I/O-Pins und anderen Hardware-spezifischen Aspekten, über *lcd_config.h* kann das Verhalten der Software (und damit zum Teil auch die resultierende Codegröße) gesteuert werden.

Die LCD-Library unterstützt für die Zeichenoperationen jeweils zwei Modi: 8 (Präprozessorkonstante *LCD_MODE_8BIT = 8*) und 16-Bit (*LCD_MODE_16BIT = 16*). Der Modus stellt dabei die Anzahl der Bits dar, die zur Farbkodierung genutzt werden. Er ist bei allen Zeichenoperationen als Parameter mit anzugeben, mit Ausnahme von *lcd_setPixel8()* und *lcd_setPixel16()*, wo vorher explizit vor jedem Aufruf *lcd_setBitMode()* aufgerufen werden muss (es sei denn der Modus hat sich nicht geändert). Durch das explizite Setzen des Bitmodus kann bei blockweisen Pixeloperationen SPI-Transfer gespart werden. Bei allen anderen Operationen wird der Modus implizit gesetzt.

Unabhängig vom Bitmodus verwendet die LCD-Library nur einen Typen für Farbwert, nämlich *lcd_color_t*, der als vorzeichenloser 16-Bit Integer Typ definiert ist und somit Farbwerte beider Modi aufnehmen kann. Allerdings ist darauf zu achten, dass Farbwerte der beiden Modi nicht untereinander vertauscht werden können (sie sind also inkompatibel zueinander), außerdem

gibt der Typ an sich keine Auskunft darüber, ob es sich um einen 8 oder 16-Bit Farbwert handelt. Die Library stellt aber einige Funktionen und Präprozessormakros zur Umrechnung der Farbwerte beider Modi ineinander sowie deren Modifikation zur Verfügung.

Die durch die LCD-Library definierten Funktionen lassen sich in folgende Kategorien einteilen:

- Funktionen zur Display-Steuerung (Initialisierung, Steuerung der Hintergrundbeleuchtung, Aktivieren/Deaktivieren des Displayinhaltes etc.)
- Grundlegende Zeichenfunktionen (Zeichnen von Pixeln, Linien, Rechtecken, Bildbereiche füllen etc.)
- Funktionen zum Zeichnen von Texten (Zeichnen von Charactern, Strings, Funktionen für Text-Metriken etc.)
- Funktionen zum Zeichnen von Bildern
- Diverse Farbwert-Funktionen (Transformationen, Konvertierung, Komponentenextraktion)

Genauere Beschreibungen der Funktionen sowie der Library generell sind in der Doxygen-Dokumentation der LCD-Library zu finden, siehe Anhang (Abschnitt 7).

4.5.1.1 Benutzungsbeispiel

Im Folgenden ist ein minimales Benutzungsbeispiel angegeben, das den Umgang mit der LCD-Library demonstrieren soll.

Listing 4.1: Codebeispiel für LCD-Library

```
1  #include <stdlib.h>
2  #include <stdbool.h>
3
4  // Nur lcd.h einbinden, keine anderen Header
5  #include "lcd/lcd.h"
6
7  // Ggf. Bild- und Font-Arrays einbinden
8  #include "fonts/font.h"
9  #include "images/image_24bit.h"
10
11
```

```
12 int main ()
13 {
14     // SPI Hardware/Pins konfigurieren,
15     // wird nicht durch lcd_configurePins() erledigt!
16     DDRB |= (1 << PB5);
17     DDRB &= ~(1 << PB6);
18     DDRB |= (1 << PB7);
19
20     lcd_configurePins(); // Konfiguriert die Display IO-Pins
21     lcd_init();          // Initialisiert den Display-Controller
22
23     // Den Bildschirm ggf. mit definiertem Inhalt füllen
24     lcd_fillScreen(LCD_COLOR8_BLACK, LCD_MODE_8BIT);
25
26     // Die Hintergrundbeleuchtung aktivieren
27     lcd_setBacklight(80);
28
29     // (Initialisierung abgeschlossen, LCD kann verwendet werden)
30
31
32     // ------------------- Zeichentests -------------------------
33
34     // Linie im 8-Bit Modus zeichnen:
35     lcd_drawLine(0, 0, LCD_RIGHT, LCD_BOTTOM, LCD_COLOR8_RED,
36                  LCD_MODE_8BIT);
37
38     // Pixel im 16-Bit Modus zeichnen:
39     // Vor lcd_setPixel() muss der Bit-Modus ggf. explizit per
40     // lcd_setBitMode() gesetzt werden!
41     lcd_setBitMode(LCD_MODE_16BIT);
42     lcd_setPixel16(0, 0, LCD_COLOR16(100, 240, 50));
43
44     // Text aus Programmspeicher im 8-Bit-Modus zeichnen:
45     // Modus wird automatisch wieder auf 8-Bit gesetzt.
46     lcd_drawText_P(PSTR("Test"),   50,  50, LCD_COLOR8_WHITE,
47                    NULL, font, 0, LCD_MODE_8BIT);
48
49     // Bild zeichnen (mit aktivierter Transparenz):
50     lcd_drawImage_P(image, 10, 10, true, NULL);
51
52     // ...
53
54     while (true);
55 }
```

4.5.2 ID3

Das ID3-Quelltextmodul implementiert das Auslesen von Textfeldern aus ID3-Tags innerhalb von MP3-Dateien, die sich in einem FAT-Dateisystem befinden. Dabei werden die ID3-Versionen 1.0, 1.1, 2.3.0 und 2.4.0 unterstützt.

Bei der Implementation wurde besonders auf die Konformität zur ID3-Spezifikation geachtet (vor allem bei den ID3v2-Versionen) [ea99] [ea00]. Es können weitestgehend alle in der Praxis gängigen ID3-Tags der genannten Versionen gelesen werden. Jedoch werden keine „synchronisierten“ Tags (d.h. Tags mit gesetztem *synchronized header flag*) sowie Tags mit „exotischen“ Optionen, wie z.B. verschlüsselten Frames, unterstützt.

Optional lässt sich für ID3v2 die Unterstützung von Unicode-kodierten Textfeldern aktivieren, welche dann (soweit möglich) auf ISO-8859-1 abgebildet werden. Die Wahl von ISO-8859-1 als interne Zeichenkodierung der Software anstelle von Unicode ist durch das Schriftart-Format der LCD-Bibliothek bedingt, das nur den Zeichenvorrat enthält, den ISO-8859-1 abdeckt.

Abbildung 4.4 gibt einen Überblick über die Strukturierung des ID3-Quelltextmoduls. Die Quelldateien *id3.c*, *id3.h* und *id3_config.h* abstrahieren von den einzelnen ID3-Tag Versionen und liefern letztlich die ausgelesenen Textfelder des Tags in einer Struct-Variable. Die Quelldatei *id3v1_genres.c* definiert einige Strings der ID3v1-Genrenamen im Programmspeicher. Die Quelldateien *id3v2.c*, *id3v2.h* und *id3v2_config.h* beinhalten Funktionalitäten zum Auslesen von ID3v2-Tags und deren Frames. Zuletzt stellen die Quelldateien *unicode.c* und *unicode.h* Funktionalitäten zum Konvertieren von Unicode-Daten nach ISO-8859-1 zur Verfügung (genutzt durch den ID3v2-Teil).

Teile des Softwaremoduls (ID3v2, Unicode) sind dabei plattformunabhängig implementiert. Mit plattformunabhängig ist dabei C-Quelltext gemeint, der dem C99-Sprachstandard entspricht und keine Abhängigkeiten außer der C-Standard-Library aufweist. Daher lässt sich der entsprechende Code (im Normalfall) ohne Änderungen auch auf anderen Architekturen als der AVR-Prozessorfamilie einsetzen.

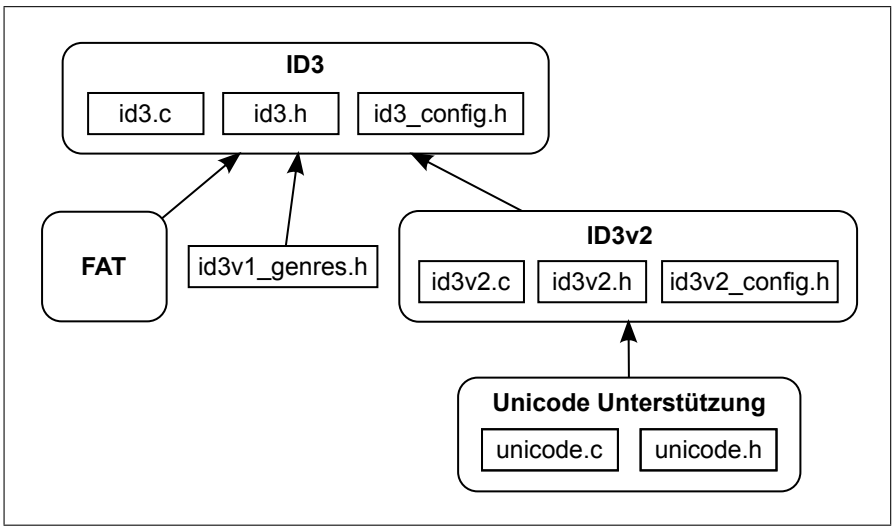


Abbildung 4.4: Struktur und Abhängigkeiten des ID3-Quelltextmoduls

4.5.2.1 ID3 Funktionalitäten

Die Quelldateien *id3.c*, *id3.h*, *id3_config.h* implementieren das Lesen von ID3-Tags aus FAT-Dateien. Für die FAT-Unterstützung wird hier auf die FAT-Quelltextbibliothek von Roland Riegel, die Teil seiner SD-Karten-Ansteuerung ist, aufgesetzt. Außerdem wird die Header-Datei *id3v1_genres.h* benötigt, die (bei entsprechender Konfiguration in *id3_config.h*) einen Array mit ID3v1 Genre-Namen im Programmspeicher definiert.

Der ID3-Quelltextteil unterstützt das Lesen der ID3-Tag Versionen 1.0, 1.1, 2.3.0 und 2.4.0. Gelesen werden einige Textfelder der Tags (welche je nach Version des Tags variieren können), die in einer struct-Variable vom Typ *id3_tag_t* gespeichert werden. Dieser Typ besitzt folgende Member-Variablen:

Member	Bedeutung
uint8_t tagType	ID3_INVALID, ID3_ID3V1, ID3_ID3V1_1, ID3_ID3V2_3_0 oder ID3_ID3V2_4_0
char *title	Titel des Musikstücks
char *artist	Interpret des Musikstücks
char *album	Album auf dem das Musikstück enthalten ist
char *trackNumber	Titelnummer des Musikstücks auf dem Album
char *date	Veröffentlichungsdatum, Aufnahmedatum oder sonstiges Datum
char *genre	Genrebeschreibung, bei ID3v1 eine von 80 fixen Genrebezeichnungen

Neben einigen Konstanten für die verschiedenen ID3-Tag-Versionen und Fehlercodes stellt der ID3-Tag-Quelltextteil folgende Funktionen bereit:

uint8_t id3_readTagFromFile(id3_tag_t *pTag, struct fat_file_struct* pFatFile, uint8_t *pBuffer, uint16_t bufferSize)

Ließt einen ID3v1 oder ID3v2 (2.3.0 oder 2.4.0) Tag aus einer FAT-Datei und setzt danach die Dateiposition auf den Anfang der MP3-Daten.

void id3_freeTagFields(id3_tag_t *pTag)

Gibt die String-Member eines ID3-Tag structs frei.

4.5.2.2 ID3v2 Funktionalitäten

Die Quelldateien *id3v2.c*, *id3v2.h* und *id3v2_config.h* beinhalten eine plattformunabhängige Implementation zum Lesen von ID3v2-Tag Textfeldern und Header-Daten aus einem Puffer. Dabei können der Tag-Header, einzelne Frame-Header sowie Text-Frames gelesen werden.

Neben diversen Konstanten für Fehlercodes und ähnliches werden folgende ID3v2-spezifische Datentypen definiert:

id3v2_header_t

Ein Struct-Datentyp für ID3v2-Tag-Header. Er besitzt folgende Member-Variablen:

Member	Bedeutung
uint8_t majorVersion	Hauptversionsnummer des Tags (3 oder 4)
uint8_t revisionNumber	Revisionsnummer (nur 0 unterstützt)
uint8_t flags	Byte, das die Header-Flags enthält
uint32_t tagSize	Größe des ID3v2 Tags, exklusive der Header-Größe (10 Byte)

id3v2_frameHeader_t

Ein Struct-Datentyp für den Header eines ID3v2-Frames. Ein ID3v2-Frame repräsentiert ein Datenfeld innerhalb eines ID3v2-Tags. Der Typ *id3v2_frameHeader_t* besitzt folgende Member-Variablen:

Member	Bedeutung
char id[4]	Frame-ID, ohne Nullterminierung (immer 4 Zeichen)
uint32_t frameSize	Größe des Frames, exklusive der Frame-Header-Größe (10 Byte)
uint16_t flags	Zwei Bytes, die die Header-Flags enthalten

Die folgenden drei Funktionen werden zum Lesen der ID3v2-Tags bereitgestellt:

uint8_t id3v2_readHeader(id3v2_header_t *pHeader, const uint8_t *pBuffer)

Ließt einen ID3v2-Header aus einem Puffer und gibt ggf. einen Fehlercode zurück.

uint8_t id3v2_readFrameHeader(id3v2_frameHeader_t *pFrameHeader, const id3v2_header_t *header, const uint8_t *pBuffer)

Ließt einen ID3v2-Frame-Header aus einem Puffer und gibt ggf. einen Fehlercode zurück.

char *id3v2_readTextFrame(const uint8_t *pBuffer, uint32_t size)

Ließt einen ID3v2-Text-Frame mit beliebiger Zeichenkodierung (ISO-8859-1 oder Unicode) aus und gibt diesen als neu im Speicher angelegten String zurück. Tritt ein Fehler auf, so wird *NULL* zurückgegeben.

4.5.2.3 Unicode nach ISO-8859-1 Funktionalitäten

Die Quelldateien *unicode.c* und *unicode.h* enthalten plattformunabhängige Funktionen zum Konvertieren von Unicode-Strings in ISO-8859-1-kodierte C-Character-Strings, soweit diese in ISO-8859-1 darstellbar sind. Unterstützt werden dabei die Unicode-Kodierungen UTF-16 und UTF-8 (big endian und little endian).

Folgende drei Funktionen werden hierzu zur Verfügung gestellt:

char* unicode_utf8ToIso88591(const uint8_t *pBuffer, uint16_t bufferSize)

Ließt aus einem Puffer UTF-8-kodierte String-Daten und versucht diese in einen ISO-8859-1 kodierten String umzuwandeln.

int8_t unicode_utf16GetByteOrder(const uint8_t *pBuffer)

Liefert die Bytereihenfolge einer UTF-16-kodierten Zeichenfolge innerhalb eines Puffers. Der Puffer muss mindestens die zwei Byte der Unicode Byte Order Mark (BOM) am Anfang enthalten.

char* unicode_utf16ToIso88591(const uint8_t *pBuffer, uint16_t bufferSize, int8_t byteOrder)

Ließt aus einem Puffer UTF-16-kodierte String-Daten und versucht diese in einen ISO-8859-1-kodierten String umzuwandeln.

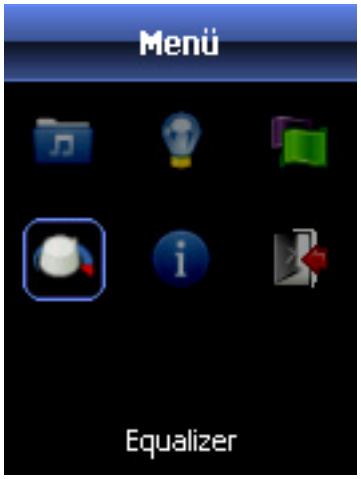
Bei den Funktionen werden immer die per Parameter *bufferSize* angegebenen Bytes aus dem Puffer gelesen und falls möglich in ISO-8859-1 umkodiert. Nicht in ISO-8859-1 kodierbare oder ungültige Unicode Zeichen (Codepoints) werden dabei im resultierenden Character-String ausgelassen. Der resultierende String wird jeweils im Speicher neu angelegt.

4.5.3 GUI-Elements

Die eigens für den MP3-Player entwickelte GUI-Elements-Library bietet verschiedene Funktion zur einfachen Erstellung einer komfortablen grafischen Oberfläche. Sie bietet eine Funktion zum interruptgesteuerten Auslesen der Tastatur des MP3-Players und benutzt die LCD-Library zur Darstellung der Elemente auf dem Siemens S65-LCD. Die wichtigsten bereitgestellten Funktionen der GUI-Library werden auf den folgenden Seiten mit Bildern dargestellt und ihre Funktion sowie ihre Benutzung werden erklärt. Zusätzlich sind Codebeispiele vorhanden um einen schnellen und unkomplizierten Einstieg in die Benutzung der Funktionen in eigenen Projekten zu ermöglichen. Dabei ist zu beachten, dass Funktionen, die lediglich etwas zeichnen, das Namenspräfix *draw* besitzen und solche, die auf eine Tastatureingabe warten, das Präfix *show*.

Abbildung 4.5: Das „Flaggschiff“ der Bibliothek in den verschiedenen Farbschemata

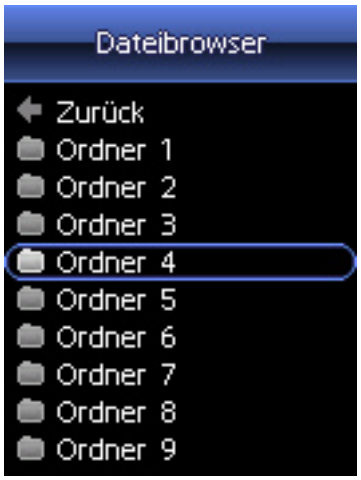
4.5.3.1 Auflistung der GUI-Elemente



gui_elements_showGraphicMenu_P

Realisiert ein grafisches Menü. Wird im Player für das Hauptmenü benutzt.

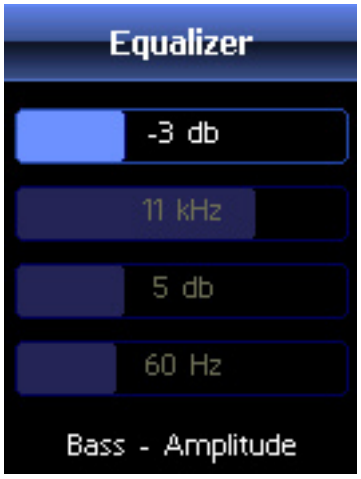
Seite 46



gui_elements_showCallbackTextMenu

Realisiert ein Textmenü mit zusätzlichen Icons für jedes Element. Wird im Player für den Dateibrowser benutzt. Die Benutzung der Callbackfunktionen dient dazu, dass nicht alle dynamischen Elemente beim Funktionsaufruf übergeben und somit im RAM gehalten werden müssen.

Seite 48



gui_elements_showSliders_P

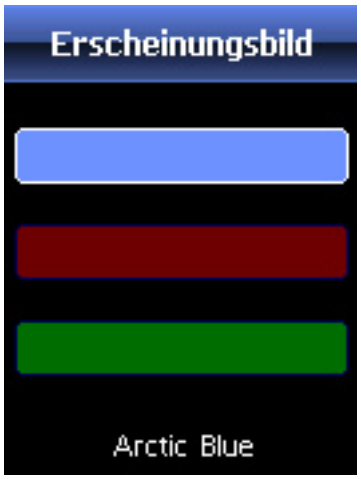
Zeigt eine Liste von einstellbaren Slidern. Wurde für den Bass-/Treble-Enhancer des MP3-Players entwickelt. Für die einzelnen Slider werden Structs übergeben.

Seite 50

gui_elements_showInfo

Zeigt eine Informations-Meldung an mit der Möglichkeit, einzelne Zeilen farblich hervorzuheben. Wurde ursprünglich implementiert, um Informationen über die Softwareversion des MP3-Players oder die SD-Karte auszugeben.

Seite 51



gui_elements_showThemeMenu

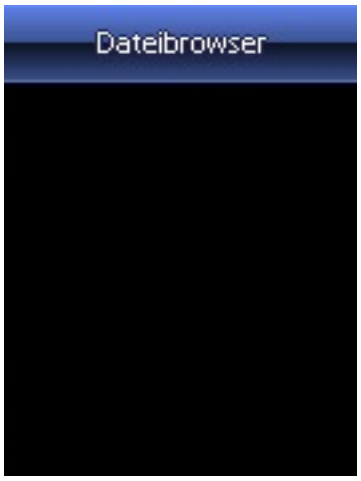
Bietet eine Auswahl über die beim Aufruf mitgegebenen Themes. Wurde entwickelt, um zwischen den drei Farbgebungen beim MP3-Player wählen zu können.

Seite 52

gui_elements_drawChecklist_P

Zeichnet eine Checkliste, bei der die einzelnen Punkte nach und nach durch Aufruf der Funktion gui_elements_drawChecklist_check() quittiert werden können. Sie wurde in den frühen Softwareversionen benutzt, um den Bootvorgang des MP3-Players verfolgen zu können.

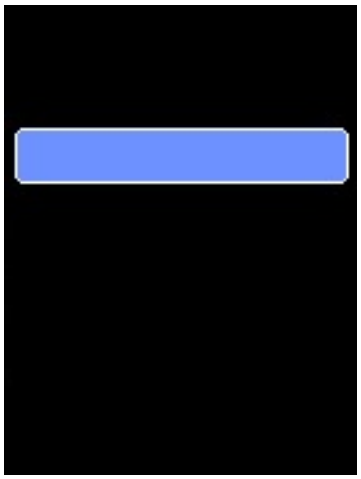
Seite 53



gui_elements_drawScreen_P

Löscht den Bildschirm und erstellt ein Panel am oberen Bildschirmrand mit dem mitgegebenen Titel. Wird von vielen anderen Oberflächen benutzt, um die Grundlage für die Darstellung zu schaffen.

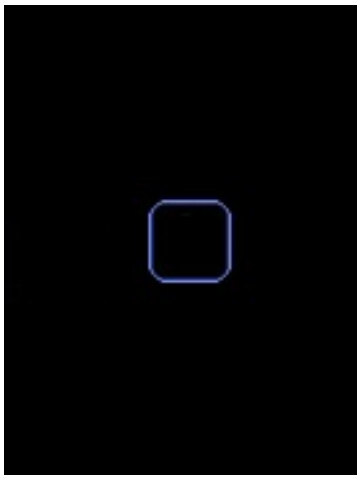
Seite 54



gui_elements_drawRoundedBox

Zeichnet eine abgerundete Box mit unterschiedlicher Hintergrund- und Randfarbe. Wird von der Funktion gui_elements_showThemeMenu aufgerufen.

Seite 54



gui_elements_drawRoundedFrame

Zeichnet eine noch intensiver abgerundete Box mit unterschiedlicher Hintergrund- und Randfarbe, sowie farblicher Abgrenzung zu den Kanten. Wird von der Funktion gui_elements_showGraphicMenu_P aufgerufen.

Seite 54

gui_elements_drawProgressbar

Zeichnet eine Fortschrittsleiste mit aufliegendem Text. Wird von der Funktion gui_elements_showSliders_P benötigt.

Seite 55

gui_elements_drawPanel_P

Zeichnet einen Panel mit aufliegendem Text an beliebiger Position. Wird von der Funktion gui_elements_drawScreen_P benötigt.

Seite 57

gui_elements_drawTextWithBorder

Schreibt einen umrandeten Text auf das Display. Wird von der Player-GUI benötigt.

Seite 57

4.5.3.2 Benutzung, Initialisierung, nötige Einstellungen

In diesem Abschnitt wird auf die Einstellungen in der Datei gui_elements␣ _config.h eingegangen und es werden die zur Initialisierung nötigen Funktionen erklärt. Nach der Initialisierung kann jede Funktion der nächsten Abschnitte frei verwendet werden. Gegen Ende dieses Abschnittes wird ein Codebeispiel für die Initialisierung in Zusammenhang mit der Siemens S65-LCD-Bibliothek gezeigt.

Einstellungen

Define	**Bedeutung**
GUI_ELEMENTS_KEY_DDR	Das DDR, an dem die Tastatur angeschlossen ist.
GUI_ELEMENTS_KEY_PORT	Der PORT, an dem die Tastatur angeschlossen ist.
GUI_ELEMENTS_KEY_PIN	Das Register, aus dem die PINs ausgelesen werden können.
GUI_ELEMENTS_ KEY_PIN_UP	An welchem Pin die Hoch-Taste angeschlossen ist.
GUI_ELEMENTS_ KEY_PIN_RIGHT	An welchem Pin die Rechts-Taste angeschlossen ist.
GUI_ELEMENTS_ KEY_PIN_DOWN	An welchem Pin die Runter-Taste angeschlossen ist.
GUI_ELEMENTS_ KEY_PIN_LEFT	An welchem Pin die Links-Taste angeschlossen ist.
GUI_ELEMENTS_ KEY_PIN_CENTER	An welchem Pin die mittlere Taste angeschlossen ist.
GUI_ELEMENTS_ KEY_COUNT	Nach wie vielen Intervallen der Tastendruck erkannt werden soll (entprellen).
GUI_ELEMENTS_ KEY_COUNT2	Nach wie vielen zusätzlichen Intervallen „Dauerfeuer“ gestartet werden soll. (Startverzögerung)

Define	Bedeutung
GUI_ELEMENTS_ KEY_COUNT3	Nach wie vielen Intervallen die Taste jeweils im Rahmen des Dauerfeuers ausgelöst werden soll. (Periodendauer)
GUI_ELEMENTS_SAVE_CODE	Wenn true, so wird die Optik verbessernder Code entfernt.
GUI_ELEMENTS_MAX_ PROGRESSBAR_CAPTION_ LENGTH	Maximale Textlänge für die Beschriftung von Fortschrittsbalken (ohne string escape character).
GUI_ELEMENTS_DARKEN_ BRIGHTNESS	Der Helligkeitswert für abgedunkelte Elemente (Bereich 0-255).
GUI_ELEMENTS_SHOW_ CALLBACK_TEXT_MENU_ COUNT	Die maximale Anzahl Einträge pro Seite der Funktion gui_elements_- showCallbackTextMenu.

uint8_t gui_elements_init()

Diese Funktion muss aufgerufen werden, bevor die ersten Gui-Elemente gezeichnet werden. Sie setzt die nötigen Schriftarten, lädt das Farbschema[1] und richtet die Tastatur ein[2].

Parameter	Bedeutung
gui_elements_theme_t theme	Das zu Anfang zu ladende Farbschema.
lcd_pgmPointer_t pHeadlineFont	Ein Pointer auf die Schriftart für Überschriften.
lcd_pgmPointer_t pFont	Ein Pointer auf die Schriftart für normale Texte.

[1] Mithilfe der Funktion gui_elements_loadTheme
[2] Mithilfe der Funktion gui_elements_initKeys (Kapitel 4.5.3.3)

uint8_t gui_elements_thread()

Diese Funktion sollte regelmäßig durch einen Timer-Interrupt aufgerufen werden, da sie die Pins des ATmegas auf gedrückte Taster überprüft und entsprechende Bits setzt, die dann beim nächsten Aufruf der Funktion gui_elements␣ _getKey() oder gui_elements_getKeyWait() verarbeitet werden.

Benutzungsbeispiel

Der folgende Code zeigt ein einfaches Codebeispiel zur Einrichtung der GUI. Die benutzten Funktionen der GUI-Elements-Bibliothek werden in den folgenden Abschnitten genauer erklärt.

Listing 4.2: Codebeispiel zur Einrichtung der gui

```
1  //Einbinden der LCD-Library
2  #include "lcd/lcd.h"
3
4  //Einbinden der Schriftarten
5  #include "fonts/font_headline.h"
6  #include "fonts/font_text.h"
7
8  //Einbinden der GUI-Library
9  #include "gui_elements/gui_elements.h"
10
11 //Callback-Funktionen
12 void keyPressed()
13 {
14         //...
15 }
16
17 void idleThread()
18 {
19         //...
20 }
21
22 int main()
23 {
24         //Theme definieren
25         gui_elements_theme_t theme;
26         theme->title = PSTR("Arctic␣Blue");
27         theme->colorTheme8 =
28                 LCD_COLOR8_RGB(96, 128, 255);
29         theme->colorPanel8 = theme->colorTheme8;
30         theme->colorPanelText8 = 0xFF;
31         theme->colorBackground8 = 0;
32         theme->colorBorder8 =
33                 LCD_COLOR8_RGB(70, 70, 160);
```

```
34          theme->colorText8 = 0xFF;
35          theme->colorSelected8 =
36                  LCD_COLOR8_RGB(96, 128, 255);
37
38          //GUI + Tastatur initialisieren
39          gui_elements_init(theme,
40                  font_headline, font_text);
41
42          //Callback-Funktionen einrichten
43          gui_elements_setCallbackFunction-
44                  KeyPressed(keyPressed);
45          gui_elements_setCallbackFunction-
46                  NoKeyPressed(idleThread);
47
48          //Display einrichten
49          lcd_configurePins();
50          lcd_init();
51          lcd_setBacklight(100);
52
53          //Benutzung der GUI-Elemente
54          //...
55
56          return 1;
57  }
```

4.5.3.3 Tastatureingabe

uint8_t gui_elements_initKeys()

Diese Funktion setzt die entsprechenden Data Direction Register und Pins des ATmegas auf Eingang und aktiviert die Pull-Up-Widerstände. Sie wird automatisch von der Funktion gui_init() aufgerufen. Die Register und Pins werden in der Datei gui_elements_config.h definiert.

uint8_t gui_elements_getKey()

Diese Funktion liefert die gedrückte Taste zurück oder GUI_ELEMENTS_KEY␣ _NONE, falls keine Taste gedrückt wurde. Die eigentliche Überprüfung der Pins des ATmegas wird allerdings durch Aufruf der Funktion gui_elements␣ _thread() ausgeführt. Diese Funktion setzt dann in einer Variable entsprechende Bits für die gedrückten Tasten, bis die Funktion gui_elements_getKey() oder gui_elements_getKeyWait() aufgerufen wird und die Bits verarbeitet.

Rückgabewert	**Bedeutung**
GUI_ELEMENTS_KEY_NONE	Es wurde keine Taste gedrückt.
GUI_ELEMENTS_KEY_UP	Die linke Taste wurde gedrückt.
GUI_ELEMENTS_KEY_RIGHT	Die rechte Taste wurde gedrückt.
GUI_ELEMENTS_KEY_DOWN	Die obere Taste wurde gedrückt.
GUI_ELEMENTS_KEY_LEFT	Die untere Taste wurde gedrückt.
GUI_ELEMENTS_KEY_CENTER	Die mittlere Taste wurde gedrückt.

uint8_t gui_elements_getKeyWait()

Diese Funktion verhält sich analog zur Funktion gui_elements_getKey() mit dem Unterschied, dass die Funktion erst beim Drücken einer Taste verlassen wird.

uint8_t gui_elements_setCallbackFunctionKeyPressed()

Mithilfe dieser Funktion kann eine Callback-Funktion übergeben werden, die aufgerufen wird, sobald die Funktion gui_elements_getKey() oder gui␣ _elements_getKeyWait() einen Tastendruck bemerkt. Diese Funktion ist sehr nützlich, wenn man eine automatische Abdunklung der Hintergrundbeleuchtung vorsieht, die bei Tastendruck wieder aufleuchten soll.

Parameter	**Bedeutung**
gui_elements_callbackFunction_t	Ein Pointer auf eine void-Funktion ohne Parameter.

uint8_t gui_elements_setCallbackFunctionNoKeyPressed()

Mithilfe dieser Funktion kann eine Callback-Funktion übergeben werden, die aufgerufen wird, wenn die Funktion gui_elements_getKey() oder gui_elements_getKeyWait() keinen Tastendruck bemerkt. Dadurch können wiederkehrende Aufgaben erledigt werden ohne die Reaktionszeit auf Eingaben stark zu beeinflussen.

Parameter	Bedeutung
gui_elements_callbackFunction_t	Ein Pointer auf eine void-Funktion ohne Parameter.

4.5.3.4 Oberflächen mit Tastatureingabe

uint8_t gui_elements_showGraphicMenu_P()

Diese Funktion realisiert ein grafisches Menü, das eine beliebige Anzahl von Menüpunkten in Form von Symbolen anzeigen kann. Der aktuell ausgewählte Menüpunkt leuchtet dabei heller als die übrigen und wird zusätzlich durch einen auffälligen Rahmen gekennzeichnet. Die Navigation erfolgt mit den Richtungstasten, das Ausführen eines Menüpunktes erfolgt durch die mittlere Taste.

Parameter	Bedeutung
PGM_P menu[]	Ein Pointer auf ein PROGMEM-Array mit den Menüpunkten. Der Aufbau eines solchen Arrays wird im Beispielcode gezeigt.
uint8_t selected	Gibt den Index des zu Anfang selektieren Menüpunktes an.

Rückgabewert	Bedeutung
Beliebig	Den Index des ausgewählten Menüpunktes.

Listing 4.3: Codebeispiel zu gui_elements_showGraphicMenu_P()

```
#include "images/img_ordnerwechseln.h"
<...>
#include "images/img_zurueck.h"

// Hauptmenü im Programmspeicher
char pgm_menu_title[] PROGMEM = "Menü";
char pgm_menu_0[] PROGMEM = "Ordner wechseln";
<...>
char pgm_menu_5[] PROGMEM = "Zurück";

PGM_VOID_P pgm_menu[] PROGMEM = {
        pgm_menu_title, //Titel
        PGM_UINT8(6), //Anzahl der Menüeinträge
        PGM_UINT8(3), //Anzahl der Spalten
        pgm_menu_0,     //Titel des 1. Eintrags
        img_ordnerwechseln, //Bild des 1. Eintrags
        <...>
        pgm_menu_5,
        img_zurueck
};

//Aufruf des Menüs
switch (gui_elements_showGraphicMenu_P
                (pgm_menu, menu_selected))
{
        case 0: //Ordner wechseln
                break;
        <...>
        case 5: //Zurück
                break;
}
```

uint8_t gui_elements_showCallbackTextMenu()

Diese Funktion zeichnet ein Textmenü, das seine Elemente über eine Callback-Funktion ermittelt. Es benutzt eine zweite Callback-Funktion, um die Gesamtanzahl der Elemente zu ermitteln.

Parameter	**Bedeutung**
PGM_P title	Ein Pointer auf den Titel des Menüs.
gui_elements_callbackTextMenuCallback_getCount_t fpGetCount	Ein Pointer auf eine Funktion, die die Anzahl der Elemente liefert.
gui_elements_callbackTextMenuCallback_getItem_t fpGetItem	Ein Pointer auf eine Funktion, die ein Element und seinen Typ anhand des Indexes liefert und 1 bei Erfolg, 0 bei Fehler zurückgibt.
PGM_P captionLoad	Ein Pointer auf ein Text, der während des Ladens eines Eintrags angezeigt werden soll. Wird NULL mitgegeben, so wird keine Ladenachricht angezeigt.
lcd_pgmPointer_t *images	Ein Array von Bildern für die verschiedenen Typen. Die Typnummer entspricht der Position des Typbildes im Array.

Parameter	**Bedeutung**
uint8_t trimText	Wenn 1, so wird zu langer Text automatisch gekürzt und um „...“ erweitert.
uint8_t maximumCaptionLength	Gibt die maximale Länge (ohne string escape char) eines Menüeintrags an, damit entsprechend Speicher reserviert werden kann. Dieser Wert wird auch an fpGetItem weitergegeben.

Rückgabewert	**Bedeutung**
Beliebig	Den Index des ausgewählten Menüpunktes.

Listing 4.4: Codebeispiel zu gui_elements_showCallbackTextMenu()

```
//Die Funktion, die die Anzahl der Elemente liefert
uint16_t getCount()
{
        return 42;
}

//Die Funktion, die ein Element liefert
uint8_t getItem(uint16_t index,
                uint8_t maximumCaptionLength,
                char *itemCaption, uint8_t *type)
{
        if (index < 42)
        {
                //Titel ist Itemnummer
                char buffer[3];
                itoa(index, buffer, 10);
                strncpy(itemCaption,
                        buffer,
                        maximumCaptionLength);

                *type = 0; //Typ ist konstant 0

                //Das Element wurde gefunden
                return 1;
        } else
        {
                //Das Element wurde nicht gefunden
                return 0;
        }
}

//Array von Pointern auf Typenicons einrichten
uint8_t *images[2] = {image_type_0, image_type_1};

//Aufruf der Funktion
index = gui_elements_showCallbackTextMenu(PSTR("Menü"),
            getCount, getItem, images);
```

void gui_elements_showSliders_P

Diese Funktion zeigt eine Liste von Slidern. Die Richtungstasten hoch und runter wählen den zu ändernden Slider aus, die Richtungstasten links und rechts ändern dessen Wert. Der aktuell ausgewählte Slider leuchtet auf. Die Slider werden mithilfe eines Structs definiert, der im Beispielcode genauer erklärt wird.

Parameter	Bedeutung
PGM_P title	Ein Pointer auf den Titel der Sliderliste.
uint8_t nSliders	Gibt die Anzahl der darzustellenden Slider an.
gui_slider_t pSliders[]	Ein Array mit Slider-Structs.

Listing 4.5: Codebeispiel zu gui_showSliders_P()

```
1  //Die Slider definieren
2  gui_slider_t sliders[4];
3
4  // Der Titel des aktuellen Sliders
5  sliders[0].title = PSTR("Bass␣-␣Amplitude");
6
7  //Der minimale einstellbare Wert
8  sliders[0].minValue = PLAYER_BASS1_MIN;
9
10 //Der maximale einstellbare Wert
11 sliders[0].maxValue = PLAYER_BASS1_MAX;
12
13 //Ein Pointer auf den zu ändernden Wert
14 sliders[0].pValue = &settings_bass1;
15
16
17
18 //Ein Pointer auf eine Funktion, die mit dem
19 //neuen Wert "live" aufgerufen wird
20 sliders[0].valueCallback = player_setBass1;
21
22 //Eine Funktion, die minValue, maxValue und den
23 //eingestellten Wert entgegennimmt, um daraus
24 //den auf dem Slider anzuzeigenden Wert erstellt.
25 //Wird NULL eingesetzt, so wird die Prozentzahl
26 //angezeigt.
27 sliders[0].captionCallback = equalizer_callback;
28 <...>
29
30 //Die Sliderfunktion aufrufen.
31 gui_elements_showSliders_P(PSTR("Equalizer"), 4, sliders);
32
33 //Die Callback-Funktion könnte wie folgt aussehen:
34 void equalizer_callback(uint8_t minValue,
```

```
35                          uint8_t maxValue,
36                          uint8_t value,
37                          char *caption)
38 {
39         //Wertebereich von -8 bis 7 dB
40         char buffer[6];
41         itoa((float)(value-minValue) /
42                 (maxValue-minValue) * 15 - 8,
43                buffer, 10);
44         strcat_P(buffer, PSTR("␣dB"));
45         strcpy(caption, buffer);
46 }
```

void gui_elements_loadTheme

Diese Funktion lädt mit sofortiger Wirksamkeit das mitgegebene Theme.

Parameter	Bedeutung
gui_elements_theme_t theme	Das zu ladende Theme.

void gui_elements_showInfo()

Diese Funktion zeigt einen Informationsbildschirm. Der anzuzeigende Titel und Text kann mitgegeben werden. Beim Text wird das Zeichen \n erkannt und automatisch in die nächste Zeile gesprungen. Das Zeichen \r wurde dazu zweckentfremdet, um eine Zeile farblich hervor heben zu können.

Parameter	Bedeutung
PGM_P title	Ein Pointer auf den Titel der Informationsseite.
char *text	Der anzuzeigende Text.
uint8_t wait	Wenn 1, so wird auf einen Tastendruck gewartet, bevor die Funktion verlassen wird.
uint8_t centered	Gibt an, ob der Text der einzelnen Zeilen zentriert angezeigt werden soll.
uint8_t trimText	Wenn 1, so wird zu langer Text automatisch gekürzt und um „...“ erweitert.

uint8_t gui_elements_showThemeMenu()

Die GUI bietet eine Unterstützung für mehrere Themes. Diese Funktion zeigt einen Auswahlbildschirm für die zur Wahl stehenden vordefinierten Themes. Jedes Theme hat eine repräsentative Farbe und einen Titel. Für jedes zur Auswahl stehende Theme wird ein Rechteck mit der repräsentierenden Farbe gezeichnet. Der Benutzer kann also anhand der Farbe und einem aussagekräftigen Titel zwischen den Themes wählen. Für Themes steht ein Struct zur Verfügung, der im Beispielcode erklärt wird. Abbildung 4.5 zeigt drei Beispielthemes.

Parameter	**Bedeutung**
PGM_P title	Ein Pointer auf den Titel der Themeauswahl.
uint8_t nThemes	Gibt die Anzahl der zur Auswahl stehenden Themes an.
gui_elements_theme_t pThemes[]	Ein Array mit Theme-Structs.
uint8_t selectedTheme	Den Index des zurzeit ausgewählten / vorselektierten Themes.

Rückgabewert	**Bedeutung**
Beliebig	Den Index des ausgewählten Themes.

Listing 4.6: Codebeispiel zu gui_elements_showThemeMenu()

```
1  //Themes definieren
2  gui_theme_t themes[3];
3
4  themes[0].title = PSTR("Arctic␣Blue");
5  themes[0].colorTheme = LCD_COLOR8_RGB(96, 128, 255);
6  themes[0].colorPanel = theme->colorTheme;
7  themes[0].colorPanelText = 0xFF;
8  themes[0].colorBackground = LCD_COLOR8_RGB(0, 0, 40);
9  themes[0].colorBorder = LCD_COLOR8_RGB(48, 64, 128);
10 themes[0].colorText = 0xFF;
11 themes[0].colorSelected = LCD_COLOR8_RGB(96, 128, 255);
12 <...>
13
14 //Funktion aufrufen
15 settings_theme = gui_elements_showThemeMenu(PSTR("Themes"),
16                 3, themes,
17                 settings_theme);
```

4.5.3.5 Oberflächen ohne Tastatureingabe

void gui_elements_drawChecklist_P()

Diese Funktion zeigt eine Checkliste an, deren Elemente per Aufruf der Funktion void gui_elements_drawChecklist_check() nacheinander quittiert werden können. Die Checkliste wird in einem PROGMEM-Array gespeichert.

Parameter	Bedeutung
PGM_VOID_P checklist[]	Ein Pointer auf die Checkliste.

Listing 4.7: Codebeispiel zu gui_elements_drawChecklist_P()

```
1  // Boot-Checkliste im Programmspeicher
2  char title[] PROGMEM = "Startvorgang";
3  char item_1[] PROGMEM = "Display␣einrichten";
4  char item_2[] PROGMEM = "Einstellungen␣laden";
5  char item_3[] PROGMEM = "SD-Karte␣einrichten";
6  char item_4[] PROGMEM = "MP3-Decoder␣einrichten";
7  char item_5[] PROGMEM = "Bereit!";
8
9  PGM_VOID_P list[] PROGMEM = {
10         title, // Titel
11         PGM_UINT8(5), // Anzahl der Einträge
12         item_1, // 1. Element
13         item_2,
14         item_3,
15         item_4,
16         item_5
17 };
18
19 //Aufruf der Funktion
20 gui_elements_drawChecklist_P(pgm_checklistboot);
21
22 <...Display einrichten...>
23 gui_elements_drawChecklist_check();
24
25 <...Einstellungen laden...>
26 gui_elements_drawChecklist_check();
27
28 <...>
```

void gui_elements_clearScreen()

Diese Funktion löscht den Bildschirm. Sie ruft lediglich einen Befehl der LCD-Library auf, der den Bildschirm löscht und dient zur Abkapselung der LCD-Library vom Hauptprojekt.

void gui_elements_drawScreen_P()

Diese Funktion stellt den vielen Funktionen zugrunde liegenden Standard-Bildschirm dar, welcher aus einem schwarzen Hintergrund mit einem Panel mit dem Titel am oberen Rand besteht.

Parameter	Bedeutung
PGM_P title	Ein Pointer auf den Titel der Seite.

4.5.3.6 Zeichenelemente

void gui_elements_drawRoundedBox()

Diese Funktion zeichnet eine abgerundete Box mit variablen Dimensionen, sowie variabler Rahmen- und Hintergrundfarbe.

Parameter	Bedeutung
uint8_t x1, x2, y1, y2	Die Koordinaten der Ecken der Box.
lcd_color_t *colorBackground8	Ein Pointer auf die Hintergrundfarbe der Box. Wird der Nullpointer mitgegeben, so ist die Box transparent.
lcd_color_t colorBorder8	Die Rahmenfarbe der Box.

void gui_elements_drawRoundedFrame()

Diese Funktion bietet eine Alternative zu gui_elements_drawRoundedBox(). Die Abrundung der Ecken ist noch ausgeprägter und bietet eine leichte farbliche Abgrenzung zu den Kanten des Frames. Der abgerundete Frame findet in gui_elements_showGraphicMenu_P() Anwendung und dient zur zusätzlichen Markierung des ausgewählten Menüpunktes.

Parameter	Bedeutung
uint8_t x1, x2, y1, y2	Die Koordinaten der Ecken des Frames.
lcd_color_t colorBorder8	Die Rahmenfarbe des Frames.
lcd_color_t colorBackground8	Die Hintergrundfarbe des Frames.

void gui_elements_drawProgressbar()

Diese Funktion zeichnet eine Fortschrittsleiste unter Zuhilfenahme der Funktion gui_elements_drawRoundedBox(). Der Fortschritt wird anhand der mitgegebenen positiven Werte minValue, maxValue und value berechnet. Ist bShowCaption wahr, so kann mithilfe einer Callback-Funktion ein eigener variabler Text für die eingestellten Werte auf der Fortschrittsleiste angezeigt werden. Wird dort der Nullpointer mitgegeben, wird automatisch die eingestellte Prozentzahl angezeigt.

Parameter	Bedeutung
uint8_t x1, x2, y1, y2	Die Koordinaten der Ecken der Fortschrittsleiste.
lcd_color_t colorText8	Die Textfarbe, falls ein Titel auf der Fortschrittsleiste angezeigt werden soll.
lcd_color_t colorFill8	Die Füllfarbe des Fortschrittsbalkens.
lcd_color_t colorBackground8	Die Hintergrundfarbe der Box.
lcd_color_t colorBorder8	Die Rahmenfarbe der Box.
uint8_t minValue, maxValue	Der einstellbare Minimal- und Maximalwert.
uint8_t value	Der voreingestellte Wert.
gui_elements_progressbar-CaptionCallback_t captionCallback	Eine Callback-Funktion, die anhand von minValue, maxValue und dem aktuell eingestellten Wert einen Titel ermittelt und zurückliefert. Die Parameter einer solchen Funktion sind im Codebeispiel zu sehen. Wird der Nullpointer mitgegeben, so wird die Prozentzahl angezeigt.
bool bShowCaption	Wenn wahr, so wird ein Titel auf der Fortschrittsleiste angezeigt.

Listing 4.8: Codebeispiel zu gui_drawProgressbar()

```
1  //Die Callbackfunktion für den Titel
2  //der Fortschrittsleiste
3  void callbackFunction(uint8_t minValue,
4                  uint8_t maxValue,
5                  uint8_t value,
6                  char *caption)
7  {
8          if (value == maxValue)
9                  //Wenn der Maximalwert
10                 //eingestellt wurde, ist das
11                 //Licht immer angeschaltet.
12                 strcpy_P(caption, PSTR("Immer␣an"));
13         else
14         {
15                 //Wert in Sekunden
16                 char buffer[6];
17                 itoa(value, buffer, 10);
18                 strcat_P(buffer, PSTR("␣s"));
19                 strcpy(caption, buffer);
20         }
21 }
22
23 //Aufruf von gui_elements_drawProgressbar
24 uint8_t minValue = 0;
25 uint8_t maxValue = 200;
26 uint8_t value = 30;
27 gui_elements_drawProgressbar(0, 0, 100, 20,
28                 WHITE, BLUE, WHITE,
29                 minValue, maxValue, value,
30                 callbackFunction, true);
```

void gui_elements_drawPanel_P()

Diese Funktion zeichnet ein Panel mit aufliegendem, umrandeten Text.

Parameter	Bedeutung
uint8_t x1, y1, y2	Koordinaten des Panels, die Höhe ist fest.
PGM_P caption_P	Ein Pointer auf den Titel des Panels.
lcd_color_t color8	Die Grundfarbe des Panels, anhand derer der Farbverlauf berechnet wird.
lcd_color_t colorCaption8	Die Textfarbe des Titels.
lcd_pgmPointer pFont	Ein Pointer auf die Schriftart des Titels.

void gui_elements_getPanelHeight()

Diese Funktion liefert die Höhe des eingespeicherten Panels der Funktion gui_elements_drawPanel_P().

void gui_elements_drawTextWithBorder()

Diese Funktion zeichnet einen umrandeten Text an eine bestimmte Stelle.

Parameter	Bedeutung
char *text	Der anzuzeigende Text.
uint8_t x, y	Koordinaten des Textes.
lcd_color_t colorText8	Die Farbe des Textes.
lcd_color_t colorBorder8	Die Farbe der Textumrandung.
lcd_pgmPointer pFont	Ein Pointer auf die Schriftart.

void gui_elements_drawTextWithBorder_P()

Diese Funktion zeichnet einen umrandeten Text an eine bestimmte Stelle.
Sie unterscheidet sich lediglich im ersten Parameter von gui_elements_ drawTextWithBorder() und erwartet einen Pointer auf einen im Programmspeicher liegenden String.

4.5.4 SD-Karten-Ansteuerung

Zur Ansteuerung der SD-Karte und dem Lesen aus FAT-Dateisystemen wird in diesem Projekt eine Quelltextbibliothek von Roland Riegel verwendet [Rie10]. Diese Bibliothek besteht aus folgenden Modulen:

- Ansteuerung von MMC/SD/SDHC-Karten
- Unterstützung für Partitionstabellen
- FAT-Dateisystem-Unterstützung

Dadurch unterstützt der MP3-Player die Speicherkarten-Standards *SD* und *SDHC* sowie die Dateisysteme *FAT16* und *FAT32*. Es findet dabei nur lesender Zugriff statt, schreibende Funktionalitäten der Bibliothek sind deaktiviert.

Damit die SD-Bibliothek den Ansprüchen des Projekts genügt, mussten einige Änderungen am Quelltext vorgenommen werden:

sd_raw_config.h:

- Ergänzung des Präprozessor-Makros *set_spi_clock()*, mit dem vor jeder SPI-Kommunikation mit der SD-Karte der SPI-Takt des Controllers neu gesetzt werden kann (falls dieser für andere SPI-Slaves verändert wurde).

sd_raw.c:

- Die Pin-Initialisierung (Slave Select) wurde in die Funktion *sd_raw_configure_pins()* ausgelagert, so dass diese separat aufgerufen werden kann. Die Initialisierung der SPI-Hardware erfolgt dann nur noch einmal im Hauptprogramm und nicht mehr separat für die SD-Ansteuerung.
- Aufrufe von Makro set_spi_clock() wurden hinzugefügt.

fat_config.h:

- Präprozessor-Konstante *FAT_LFN_MAX_LENGTH* auf den Wert der Konstante *FILENAME_MAXLEN* aus *global.h* gesetzt.

fat.c / fat.h:

- Beim Typ *struct fat_dir_entry_struct* wurde die Array-Größe des Members *long_name* auf *FAT_LFN_MAX_LENGTH* gesetzt, statt fester 32 Zeichen.
- Funktion *fat_get_file_position()* ergänzt, die die aktuelle Position in einer FAT-Datei liefert (welche vorher durch den opaken Datentypen abgekapselt war).

- Funktion *fat_get_file_size()* ergänzt, die die Größe einer FAT-Datei liefert (welche vorher durch den opaken Datentypen abgekapselt war).
- Funktion *fat_read_dir_raw()* ergänzt, eine modifizierte Variante von *fat_read_dir()*, die die Einträge in einem Verzeichnis anhand eines cluster- und offset-Parameters liest. Dadurch wird kein langer Dateiname von *struct fat_dir_struct* als Parameter benötigt, wodurch Hauptspeicher gespart werden kann. Wird für FAT-File-Browser benötigt.

Ergänzend zu dieser Bibliothek wurde ein FAT-File-Browser implementiert, der leichteres iterieren über Dateien eines FAT-Verzeichnisses ermöglicht (siehe Abschnitt 4.5.5).

4.5.5 FAT-File-Browser

Ergänzend zu der FAT-Quelltextbibliothek von Roland Riegel (die Teil der SD-Karten-Ansteuerung ist) wurde ein Datei-Browser-Quelltextmodul implementiert, welches ein einfaches Iterieren über FAT-Verzeichniseinträge innerhalb eines Verzeichnisses eines FAT-Dateisystems in sortierter Reihenfolge ermöglicht. Dieses im Folgenden als *FAT-File-Browser* bezeichnete Modul setzt auf der FAT-Dateisystem-Unterstützung von Roland Riegel auf.

Nach der Initialisierung kann mit dem FAT-File-Browser entweder der erste oder letzte FAT-Eintrag in einem Verzeichnis geladen werden. Anschließend kann das Folgeelement geladen werden, entweder in aufsteigender oder absteigender Reihenfolge. Die Sortierung der Elemente erfolgt dabei nach lexikografischer Ordnung, standardmäßig werden dabei die Dateinamen verglichen. Durch das Angeben einer (optionalen) Callback-Funktion ist es aber auch möglich, beliebige Merkmale eines FAT-Eintrages für die alphabetische Sortierung zu verwenden. Diese Callback-Funktion erhält dabei als Parameter einen Zeiger auf einen FAT-Eintrags-Struct und liefert einen Character-String, der anschließend für die Sortierung verwendet wird. Somit wäre es z.B. möglich, die Datei-Einträge nach bestimmten ID3-Tag Feldern zu sortieren (was in diesem Projekt jedoch nicht implementiert wurde). Eine zweite optionale Callback-Funktion ermöglicht zusätzlich das Filtern bestimmter FAT-Einträge (in unserem Fall werden z.B. nur MP3-Dateien und Ordner vom File-Browser berücksichtigt).

Neben dem Laden eines Elements ermöglicht der FAT-File-Browser auch das Wechseln des aktuellen Verzeichnisses. Außerdem kann der geladenen FAT-Eintrag jederzeit freigegeben werden, um den belegten Hauptspeicher freizugeben.

4.5.5.1 Schnittstelle

An die zu entwickelnde FAT-File-Browser Komponente wurden vor der Implementation zwei Anforderungen gestellt: Die Code-Größe sollte möglichst klein sein, trotzdem sollte der File-Browser die nötige Funktionalität und Flexibilität liefern. Um eine möglichst kompakte Code-Größe zu erreichen, wurde daher gezielt auf eine strikte Kapselung in C mittels *opaker Datentypen* verzichtet, da dies einen deutlich erhöhten Programmaufwand (und daraus resultierende Code-Größe) mit sich gebracht hätte.

Stattdessen wurde eine einfachere Schnittstelle gewählt, die dem Aufrufer vollen Zugriff auf die Datentypen gewährt. Dadurch wird gleichzeitig jedoch

eine höhere Vorsicht des Aufrufer bei der Benutzung des File-Browsers abverlangt und die Fehlergefahr steigt erheblich. Aus diesem Grund sollte sich der Aufrufer bei der Benutzung des FAT-File-Browsers strikt an die hier beschriebene Schnittstellendefinition halten und vor allem zusätzliche Zugriffe auf Member-Variablen des File-Browser-Structs möglichst vermeiden.

Für eine gekapselte Implementation hätte man den Typ *fatFileBrowser_t* erst in der Source-Datei definieren und im Header nur deklarieren müssen, was ihn zu einem opaken Typ gemacht hätte und Zugriffe auf die Struct-Member direkt durch den aufrufenden Programmteil verhindert hätte. Für Zugriffe auf die Struct-Member wären dann einige *getter-Funktionen* nötig gewesen.

Konstanten:

Das FAT-File-Browser-Quelltextmodul definiert folgende Konstanten in seiner Schnittstelle:

Suchrichtungen bzw. Suchmodi:

Konstante	**Bedeutung**
FAT_FILE_BROWSER_PREVIOUS (-1)	Suchrichtung für den vorherigen Eintrag in lexikografisch sortierter Ordnung
FAT_FILE_BROWSER_NEXT (1)	Nächster Eintrag
FAT_FILE_BROWSER_FIRST (2)	Erster Eintrag im Verzeichnis
FAT_FILE_BROWSER_LAST (3)	Letzter Eintrag im Verzeichnis

Eintragstypen:

Konstante	**Bedeutung**
FAT_FILE_BROWSER_ENTRY_DIRECTORY (0)	Eintragstyp für Verzeichnisse
FAT_FILE_BROWSER_ENTRY_FILE (1)	Eintragstyp für Dateien

Datentypen:

Folgende Typen werden definiert:

bool (*fatFileBrowser_boolCallback_t) (const struct fat_dir_entry_struct *pFatEntry)

Typ für eine Callback-Funktion, die einen Zeiger auf einen FAT-Eintrag-Struct übergeben bekommt und einen bool-Wert zurückliefert. Wird für die Dateifilter-Callback-Funktion verwendet.

char* (*fatFileBrowser_attributeCallback_t) (const struct fat_dir_entry_struct *pFatEntry)

Typ für eine Callback-Funktion, die einen Zeiger auf einen FAT-Eintrag-Struct übergeben bekommt und einen Character-String liefert, der als Vergleichsattribut für die alphabetische Sortierung verwendet wird.

fatFileBrowser_t

Ein Struct-Typ, der einen FAT-File-Browser repräsentiert bzw. dessen Daten speichert. Er besitzt folgende Member-Variablen:

Member	**Bedeutung**	**erlaubter Zugriff**
struct fat_fs_struct *pFatFileSystem	Zeiger auf das FAT-Dateisystem	*keiner (intern)*
struct fat_dir_entry_struct *pFatEntry	Zeiger auf den aktuell geladenen FAT-Eintrag	*lesend*
cluster_t currentDirectoryCluster	Cluster-Nummer des aktuellen Verzeichnisses	*keiner (intern)*

Member	**Bedeutung**	**erlaubter Zugriff**
int16_t index[2]	Array mit den Indizes der aktuell geladenen Datei- bzw. Verzeichniseinträge. Als Index kann FAT_FILE_BROWSER␣ _ENTRY_DIRECTORY oder FAT_FILE_BROWSER␣ _ENTRY_FILE verwendet werden. Die Indizes beginnen bei 0 und sind nur gültig, falls pFatEntry ungleich *NULL* ist	*lesend*
int16_t nEntries[2]	Anzahl der Datei- bzw. Verzeichniseinträge im aktuellen Verzeichnis. Als Index kann FAT_FILE_BROWSER␣ _ENTRY_DIRECTORY oder FAT_FILE_BROWSER␣ _ENTRY_FILE verwendet werden	*lesend*
fatFileBrowser_bool␣ Callback_t fileFilterCallback	Callback-Funktion zum Filtern von Einträgen. Wird bei der Initialisierung gesetzt	*keiner (intern)*
fatFileBrowser_attribute␣ Callback_t getFileAttributeCallback	Callback-Funktion, mit der ein Vergleichsattribut für die Sortierung geliefert werden kann. Wird bei der Initialisierung gesetzt	*keiner (intern)*

Funktionen

Zur Benutzung der FAT-File-Browser Komponente werden folgende fünf Funktionen bereitgestellt:

bool fatFileBrowser_init()

Die Funktion fatFileBrowser_init() initialisiert einen FAT-File-Browser und setzt dessen aktuelles Arbeitsverzeichnis auf das Stammverzeichnis.

Parameter	**Bedeutung**
fatFileBrowser_t *pFileBrowser	Zeiger auf die Struct-Variable eines FAT-File-Browsers
struct fat_fs_struct *pFatFileSystem	Zeiger auf die Struct-Variable des FAT-Dateisystem
fatFileBrowser_boolCallback_t fileFilterCallback	Zeiger auf eine Callback- Funktion, die zum Herausfiltern einzelner FAT-Einträge verwendet werden kann. Ist der Parameter *NULL*, so werden alle Datei- und Ordner-Einträge berücksichtigt
fatFileBrowser_attribute␣ Callback_t getFileAttributeCallback	Zeiger auf eine Callback- Funktion, die ein *Vergleichsattribut* für die lexikografisch geordnete Sortierung liefert. Ist der Parameter *NULL*, so wird der Name des FAT-Eintrages als Vergleichsattribut verwendet

Rückgabewert	**Bedeutung**
bool	*true*, falls die Initialisierung erfolgreich war, sonst *false*

bool fatFileBrowser_changeDirectory()

Die Funktion fatFileBrowser_changeDirectory() ändert das aktuelle Arbeitsverzeichnis eines FAT-File-Browsers, indem entweder in einen Unterordner oder den übergeordneten Ordner („..“) gewechselt wird.

Parameter	**Bedeutung**
fatFileBrowser_t *pFileBrowser	Zeiger auf die Struct-Variable eines FAT-File-Browsers
const char *directory	Charakter-String, der den Namen des Unterverzeichnis enthält, in das gewechselt werden soll. Zum Wechseln in den übergeordneten Ordner ist „..“ anzugeben.

Rückgabewert	**Bedeutung**
bool	*true*, falls das Verzeichnis gewechselt wurde, sonst *false*.

bool fatFileBrowser_loadEntry()

Die Funktion fatFileBrowser_loadEntry() lädt einen neuen FAT-Eintrag in einen FAT-File-Browser. Dabei kann entweder der erste, letzte, vorherige oder nächste Datei- oder Verzeichnis-Eintrag aus dem aktuellen Arbeitsverzeichnis geladen werden.

Parameter	**Bedeutung**
fatFileBrowser_t *pFileBrowser	Zeiger auf die Struct-Variable eines FAT-File-Browsers

Parameter	Bedeutung
int8_t direction	Die Suchrichtung bzw. der Modus, entweder FAT_FILE_BROWSER_PREVIOUS, FAT_FILE_BROWSER_NEXT, FAT_FILE_BROWSER_FIRST oder FAT_FILE_BROWSER_LAST
uint8_t entryType	Der Typ des zu ladenden Eintrages, entweder FAT_FILE_BROWSER_ENTRY_DIRECTORY oder FAT_FILE_BROWSER_ENTRY_FILE. Soll der nächste/vorherige Eintrag von einem anderen Typen als der aktuelle Eintrag geladen werden, so ist ggf. vorher entsprechend der erste/letzte Eintrag dieses Typen zu laden.

Rückgabewert	Bedeutung
bool	*true*, falls ein neues Element geladen wurde, sonst *false*. Schlägt das Laden des nächsten oder vorherigen Eintrags fehl, so bleibt der alte Eintrag geladen.

void fatFileBrowser_freeEntry()

Mit der Funktion fatFileBrowser_freeEntry() kann jederzeit der aktuell geladene Eintrag eines FAT-File-Browser wieder freigegeben werden, um Hauptspeicher zu sparen. Dies ist vor allem dann sinnvoll, wenn die FAT-Dateisystemunterstützung mit langen Dateinamen arbeitet (die entsprechende Member-Variable des FAT-File-Browser structs belegt dann permanent bis zu 255 Byte im RAM). Der Eintrag wird anschließend bei Bedarf automatisch neu geladen (der Speicher wird dynamisch alloziert). Die Member-Variable pFatEntry des FAT-File-Browser hat nach dem Aufruf den Wert *NULL*.

Parameter	Bedeutung
fatFileBrowser_t *pFileBrowser	Zeiger auf die Struct-Variable eines FAT-File-Browsers

void fatFileBrowser_free()

Die Funktion fatFileBrowser_free() gibt den reservierten Speicher eines FAT-File-Browsers frei. Intern ruft fatFileBrowser_free() lediglich fatFileBrowser_freeEntry() auf, die Funktion dient als Gegenstück zu fatFileBrowser_init().

Parameter	Bedeutung
fatFileBrowser_t *pFileBrowser	Zeiger auf die Struct-Variable eines FAT-File-Browsers, dessen Speicher freigegeben werden soll.

4.5.5.2 Benutzungsbeispiel

Im Folgenden ist ein Beispiel für die Benutzung des FAT-File-Browsers angegeben. Neben der Initialisierung mit fatFileBrowser_init() und ggf. dem Freigeben von alloziertem Speicher nach der Benutzung über fatFileBrowser_free() ist darauf zu achten, dass das erste Laden von Einträgen mit einem der beiden Parameter FAT_FILE_BROWSER_FIRST oder FAT_FILE_BROWSER_LAST geschehen muss.

Listing 4.9: Codebeispiel für FAT-File-Browser

```
1  // Header der Roland Riegel SD-Library inkludieren
2  #include "sd-reader/sd_raw.h"
3  #include "sd-reader/partition.h"
4  #include "sd-reader/fat.h"
5
6  // Header des FAT-File-Browsers inkludieren
7  #include "fat_file_browser.h"
8
9  // [...]
10
11 // Eine leere Callback-Funktion, die zum Filtern von
12 // Einträgen verwendet werden könnte
13 bool fileFilterCallback(const struct fat_dir_entry_struct
14                         *pFatEntry)
15 {
16     // Soll ein Eintrag rausgefiltert werden, könnte hier
17     // false zurückgegeben werden
18     return true;
19 }
20
21 int main()
22 {
```

```
23      fatFileBrowser_t fileBrowser;
24      struct fat_fs_struct *pFatFs;
25
26      // [...]
27
28      // FAT-File-Browser initialisieren, ohne Attribut-Callback
29      if (!fatFileBrowser_init(&fileBrowser, pFatFs,
30          fileFilterCallback, NULL))
31      {
32          puts_P(PSTR("fatFileBrowser_init() fehlgeschlagen."));
33          while (1);
34      }
35
36      // Anzahl der Dateien im Verzeichnis ausgeben
37      printf_P(PSTR("Dateien im Verzeichnis: %d\n\n"),
38              pFileBrowser->nEntries[FAT_FILE_BROWSER_ENTRY_FILE]);
39
40      // Alle Dateien im aktuellen Verzeichnis alphabetisch
41      // sortiert ausgeben
42      for (int i = 0; i < pFileBrowser->nEntries[entryType]; i++) {
43          if (i == 0)
44              // Der erste Eintrag muss immer zunächst mit
45              // FAT_FILE_BROWSER_FIRST geladen werden
46              fatFileBrowser_loadEntry(pFileBrowser,
47                                       FAT_FILE_BROWSER_FIRST,
48                                       FAT_FILE_BROWSER_FILE);
49          else
50              fatFileBrowser_loadEntry(pFileBrowser,
51                                       FAT_FILE_BROWSER_NEXT,
52                                       FAT_FILE_BROWSER_FILE);
53
54          // Index und Eintrags-Namen ausgeben
55          printf_P(PSTR("(%d/%d)\t'%s'\n"),
56                   pFileBrowser->index[FAT_FILE_BROWSER_FILE] + 1,
57                   pFileBrowser->nEntries[FAT_FILE_BROWSER_FILE],
58                   pFileBrowser->pFatEntry->long_name);
59      }
60
61      // In das Verzeichnis "subdir" wechseln
62      fatFileBrowser_changeDirectory(&fileBrowser, "subdir");
63
64      // [...]
65
66      // Reservierten Hauptspeicher freigeben
67      fatFileBrowser_free(&fileBrowser);
68
69      // [...]
70  }
```

Kapitel 5

Chronologie

Im Folgenden sollen die einzelnen Entwicklungsphasen des Projektes und deren Chronologie beschrieben werden.

Begonnen wurde mit der Konzipierung der benötigten Hardware. Hier galt es zunächst, geeignete Hardwarekomponenten zu finden, vor allem ein adäquates Display sowie einen MP3-Decoder-IC. Nachdem die Wahl des Displays auf das Siemens S65 Display fiel, wurden erste Tests zu dessen Ansteuerung mit der GLCD Assembler Library von Christian Kranz[1] durchgeführt. Als Decoder wurde der VS1011e gewählt.

Anschließend musste eine verwendbare Bibliothek zur Ansteuerung der SD-Karte gefunden werden. Nach ersten Tests mit der MMC/SD-Bibliothek von Ulrich Radig[2] entschieden sich die Autoren dazu, eine andere Bibliothek zu verwenden: Die von Roland Riegel[3] entwickelte MMC/SD-Bibliothek. Diese bot den Vorteil, dass bereits Unterstützungen für das FAT-Dateisystem und Partitionstabellen enthalten waren und sich die Softwarekonfiguration einfacher gestaltete.

Parallel zu Hardware-Experimenten mit Display und SD-Karte wurde an der Ansteuerung des VS1011e-Decoders experimentiert und mit der Implementation der Software zur Decoder-Steuerung begonnen.

Kurz danach wurde die Testplatine entwickelt, die die gesamte für das Projekt benötigte Hardware umfasst. Diese war zunächst noch mit einem ATme-

[1] Siehe [Kra05]
[2] Siehe [Rad10]
[3] Siehe [Rie10]

ga32 bestückt. Es folgten weitere Softwareentwicklung an der GUI, dem eigentlich Player-Code und an einer neuen Bibliothek zur Ansteuerung des Displays, da sich die ursprünglich verwendete Assembler-LCD-Library als unzureichend erwies.

Anfang März wurde begonnen, den Code auf den ATmega644 Kontroller zu portieren. Dies war nötig, da der Programmcode ständig wuchs und Grafiken für die GUI einiges an Speicherplatz verbrauchten. Einige Zeit später wurde mit der Entwicklung eines FAT-File-Browser Softwaremoduls begonnen, welches zur einfachen Navigation durch Einträge im FAT-Dateisystem benötigt wurde. Als nächster großer Schritt stand dann an, die GUI des Players weiterhin zu verbessern und einen vollständigen Dateibrowser (inklusive Benutzeroberfläche) zu implementieren, der auf der FAT-File-Browser Komponente aufsetzt.

Letztlich folgte die Testphase, die Beseitigung von Fehlern und das Vervollständigen der Dokumentation.

Kapitel 6

Ausblick

Die Entwicklung eines MP3-Players auf Basis eines ATmega stellt bei einfachen MP3-Playern keine große Schwierigkeit dar. Das Übertragen der Daten von einer SD-Karte zum MP3-Decoder-Chip schafft der ATmega rechtzeitig. Soll er allerdings parallel zum Abspielen einer MP3-Datei von hoher Qualität noch zusätzliche Operationen auf der SD-Karte ausführen oder große Grafiken auf das Display zeichnen, gelangt der SPI-Bus schnell an seine Grenzen. Ideal wäre es, einen Prozessor mit mehreren SPI-Bus-Systemen einzubauen und diese parallel zu benutzen. Ebenfalls hilfreich wäre zusätzlicher RAM, damit die Dateiliste im RAM sortiert werden kann und MP3-Daten länger zwischengepuffert werden können. Ein weiteres Problem bei der Entwicklung war die Tatsache, dass zu dem benutzten Display kein öffentliches Datenblatt existiert. Einige Leute haben allerdings durch Reverse Engineering einen Teil der Display-Befehle ermitteln können. Die tatsächliche Leistungsfähigkeit des Displays kann mit dem vorhandenen Inventar an ermittelten Befehlen leider nicht voll ausgereizt werden (da das Siemens S65 Handy trotz ebenfalls nur serieller Anbindung des Displays über SPI Bus einen deutlich schnelleren Bildaufbau schafft und auch Videowiedergabe bewältigt).

Trotz dieser Hindernisse ist es uns gelungen, eine weitgehend flüssige Oberfläche zu erstellen, mit der es sogar möglich ist, während dem Hören der Musik durch die Menüs zu navigieren und Einstellungen zu übernehmen.

Doch die Möglichkeiten sind noch nicht ausgeschöpft. Denkbar wäre noch die Umrüstung auf ein Touch-Display oder die Erweiterung des MP3-Players um Netzwerkfunktionalität, sodass auch das Hören von Internetradio oder das Nutzen von sozialen Musikdiensten wie beispielsweise last.fm[1] ermöglicht werden würde. Dies sollte mit dem Netzwerkchip ENC28J60 in Kombination des uIP-Stacks von Adam Dunkels realisierbar sein. Zusätzlich wäre noch eine Steuerung des Internetradios per Infrarot-Fernbedienung oder eine Weckfunktion denkbar. Bei manchen Projekten ist auch bereits eine USB 2.0-Schnittstelle zur direkten Übertragung der Daten auf den MP3-Player vorhanden.

[1]Soziale Musikplattform, die durch Übermittlung eigener Hörgewohnheiten sowie basierend auf dem Hörverhalten anderer Benutzer personalisierte Musikempfehlungen generiert, http://www.last.fm/

Kapitel 7

Anhang

Auf dem beigelegten Datenträger befinden sich:

- Quellcode des MP3-Players und der zugehörigen Tools
- Lizenzbeschreibung
- Doxygen-Dokumentation der LCD-Library
- EAGLE-Schaltplan
- EAGLE-Layout
- Latex-Quelltext der Ausarbeitung
- Ausarbeitung als PDF

Literaturverzeichnis

[Din06] Dennis Dingeldein. Ansteuern des VS1011 MP3 Decoders mit dem AVR. http://www.dingeldein-online.de/basteln/vs1011.html, 2006. Abgerufen am: 17.03.2010.

[Don J] Karsten Donat. AVR Bootloader FastBoot von Peter Dannegger. http://www.mikrocontroller.net/articles/AVR_Bootloader_FastBoot_von_Peter_Dannegger, o. J. Abgerufen am: 29.05.2010.

[ea99] Martin Nilsson et al. ID3v2.3.0 Spezifikation. http://www.id3.org/d3v2.3.0, 1999. Abgerufen am: 24.04.2010.

[ea00] Martin Nilsson et al. ID3v2.4.0 Spezifikation. http://www.id3.org/id3v2.4.0-structure, 2000. Abgerufen am: 24.04.2010.

[EA09a] Elektronik-Atelier. MP3orator. http://www.avrcard.com/products/mp3orator.htm, 2009. Abgerufen am: 17.03.2010.

[ea09b] Martin Nilsson et al. ID3 Website. http://www.id3.org/, 2009. Abgerufen am: 24.04.2010.

[Fra09a] Fraunhofer IIS. Die MP3 Geschichte. http://www.iis.fraunhofer.de/bf/amm/products/mp3/mp3history/mp3history01.jsp, 2009. Abgerufen am: 24.04.2010.

[Fra09b] Fraunhofer IIS. Was ist mp3? http://www.iis.fraunhofer.de/bf/amm/products/mp3/mp3history/wasistmp3.jsp, 2009. Abgerufen am: 24.04.2010.

[Han06] Jesper Hansen. yampp - Yet Another MP3 Player. http://www.myplace.nu/mp3/yampp7.htm, 2006. Abgerufen am: 17.03.2010.

[Jak07] Marcel Jakobs. Handydisplays am ATMega für einen Ein-Chip-Computer. http://userpages.uni-koblenz.de/~physik/informatik/ECC/handydisplay.pdf, 2007. Abgerufen am: 06.05.2010.

[Kra05] Christian Kranz. Using the Siemens S65-Display. http://www.superkranz.de/christian/S65_Display/DisplayIndex.html, 2005. Abgerufen am: 17.03.2010.

[Lot08] Nick Lott. AVR Butterfly MP3. http://www.brokentoaster.com/butterflymp3, 2008. Abgerufen am: 17.03.2010.

[Max08] Simeon Maxein. Benutzen einer SD-Speicherkarte mit dem ATmega-Microcontroller. http://userpages.uni-koblenz.de/~physik/informatik/ECC/sd.pdf, 2008. Abgerufen am: 06.05.2010.

[mik10] mikrocontroller.net. Vergleich von MP3 Decoder ICs. http://www.mikrocontroller.net/articles/MP3#Vergleich, 2010. Abgerufen am: 27.04.2010.

[Rad10] Ulrich Radig. MMC-SD. http://www.ulrichradig.de/home/index.php/avr/mmc-sd, 2010. Abgerufen am: 06.05.2010.

[Rie10] Roland Riegel. sd-reader: MMC/SD/SDHC card library. http://www.roland-riegel.de/sd-reader/index.html, 2010. Abgerufen am: 06.05.2010.

[Sol08] VLSI Solution. Layout guidelines for VS10xx family. http://www.vlsi.fi/uploads/media/vs10xxan_layout.pdf, 2008. Abgerufen am: 06.05.2010.

[VLS09] VLSI Solution. VS1011e - MP3 Audio Decoder. http://www.vlsi.fi/fileadmin/datasheets/vlsi/vs1011.pdf, 2009. Abgerufen am: 24.04.2010.

[Wik10] Wikipedia. Artikel zu ID3. http://en.wikipedia.org/wiki/ID3, 2010. Abgerufen am: 24.04.2010.