
Xmedia

1. Inhaltsverzeichnis

2. Projektentstehung

2.1. Die Idee

Die Projektidee, zumindest für den Kern des Projektes, wurde durch den Teamleiter Bernhard Wörndl-Aichriedler fast ein Jahr vor dem Projektstart ins Auge gefasst:

„Bei meinem Praktikum als Entwickler in einem größeren Betrieb, kamen meinen damaligen Kollegen und ich immer wieder 32bit Prozessoren zu sprechen, welche für mich damals noch in ungreifbarer Ferne schienen. Kurz vor dem Ende meines Praktikums wagte ich jedoch dennoch den Schritt, bestellte mir zwei Referenzboards bestückt mit dem, vor wenigen Wochen zuvor erschienen AP7000 von Atmel. Der Grund, warum ich 2 bestellte, lag darin, dass ich erwartete eines nach kurzer Zeit zu zerstören. Doch es kam alles anders. Die Leistung und vor allem die Komplexität des Systems übertrafen meine Erwartungen bei weitem, wodurch das Board damals für mich an Interesse verlor und ich es für ein Jahr beiseite legte um es im vierten Jahrgang wieder neu zu entdecken. Die eigentliche Idee kam dann von meinem Vater, der ein Gerät suchte, dass er in sein bestehendes Heimkinosystem integrieren konnte, welches es ihm ermöglicht, Musik aus dem Netzwerk abzuspielen. Da wir zu Hause jedoch nicht über kabelgebundene Netzwerkinfrastruktur verfügten, musste dass Gerät W-Lan fähig sein. Nachdem ich mit Alexander und Michael die idealen Projektpartner fand, um mich bei meinem Vorhaben zu unterstützen. Gegen Ende des vierten Jahrganges begann die Grobplanung des Projektes. Da wir jedoch mehr wollten beschlossen wir neben dem selbst Media Player, noch eine Fernsteuerung zu entwickeln. Ab diesem Zeitpunkt stand der Name fest.- xMedia
Dies sollte das nächste 3/4 Jahr unseres Lebens bestimmen...“

2.2. Beeinflussungen & Ziele

Ziel der Projektarbeit ist es eine Multimedia Station zu entwickeln, die es ermöglicht Musikdateien und gegebenenfalls auch Streams wiederzugeben. Im Mittelpunkt des Projektes steht eindeutig die Benutzerinteraktion, bei welcher neue Maßstäbe gesetzt werden sollten, die sich deutlich vom „old school“ Taster entfernen.

Die Projektziele wurden zunächst eher etwas lose formuliert, da keiner von uns Erfahrung mit solchen Systemen hatte. Sogar der wichtige Punkt der Implementation von Wireless Lan wurde nicht offizielle als Projektziel definiert. Wir setzten die Erwartungen mit unserer Zieldefinition bewusst nicht zu hoch an, da wir einfach nicht wussten ob unsere Ziele bzw. mehr unsere Träumen technisch realisierbar wären. Jedoch genau dies gab uns im Laufe des Projektes die Freiheit Ideen und aktuelle Technologien einfließen zu lassen.

Da das Projekt zunächst nicht für den kommerziellen Einsatz gedacht war, sahen wir den Markt im Bereich der Unterhaltungselektronik eigentlich nicht als Konkurrenz, sondern vielmehr als Ideenquelle. So warfen wir etwas genauere Blicke auf die Playstation Portable von Sony, auf die Nintendo Wii und natürlich auch auf den Apple IPod und das neue IPhone.

Der IPod, für viele Menschen ein Lifestylesymbol und fast sogar schon ein Modeaccessoire. Die einzigartige Steuerung über das "IPod Wheel" kombiniert mit einfachster Menüführung vermittelt dem Benutzer eine optimale User Experience und ermöglicht eine optimale Steuerung des Gerätes. Genau dieses Konzept wollten wir aufgreifen und in unsere Hardware implementieren.

Abbildung 1: Apple I-Pod

Die Playstation Portable ist ein Muss für jeden Spieler. Sie besticht vor allem durch ihre W-Lan Fähigkeiten und ihrem riesigen, brillanten Display. Auch diese Fähigkeiten kamen auf die Wunschliste unseres Projekts.

Abbildung 2: Sony PSP

Nicht nur aus der Sicht eines Gelegenheitsspielers, sondern auch aus technischer Sicht setzt die Nintendo Wii neue Maßstäbe. Die größte technische Leistung ist jedoch nicht die Konsole selbst (der es deutlich an Leistung mangelt), sondern der Controller – die Wiimote. Auch einige Eigenschaften der Wiimote kamen auf unsere Wunschliste.

Abbildung 3: Nintendo Wii

Im November 2007 erblickte das IPhone das Licht der Welt. Wie viele Entwicklungen von Apple setzte auch das IPhone einen neunen Trend mit dem Namen „Multitouch“. So wurde dieses Konzept auch auf unsere Wunschliste gesetzt.

Abbildung 4: Apple IPhone

2.3. Lasten

2.3.1. Zielbestimmungen

Es soll ein Mediaplayer entwickelt werden, der das Abspielen von Videos und Musik über Netzwerk ermöglicht. Weiters soll ein hochauflösender Display und ein qualitativ-hochwertiger AudioCodec implementiert werden. Die Bedienung soll entweder über kapazitive Tasten oder über eine eigens entwickelte Fernsteuerung erfolgen.

Die Bedienung der Fernsteuerung erfolgt einfach und komfortable entweder über:

- den Touch-Screen
- die kapazitiven Tasten
- durch Kippen der Fernsteuerung in die jeweilige Richtung

Das Menü soll übersichtlich und benutzerfreundlich entwickelt werden. Die Navigation durch das Menü sollte intuitiv erfolgen.

2.3.2. Produkteinsatz

Da xMedia mehrere bestehende Systeme in einem integriert und darüber hinaus die Funktionen sogar erweitert, ist man in der Lage mehrere herkömmliche Geräte zu ersetzen. xMedia wurde für Privatanwender, die großen Wert auf Funktion und Design legen, entwickelt.

2.3.3. Produktfunktionen

2.3.3.1. Mainstation

Darstellung auf Display
Zur Darstellung der Videos und Musik soll ein hochauflösendes Display integriert werden.

Darstellung auf externen Monitor
Die Darstellung der Videos bzw. Musik soll mittels VGA-Ausgang auf einen externen Monitor übertragbar sein.

Wiedergabe von Musik über Lan und W-lan
Als Quelle der abspielbaren Musik und Videos soll Lan sowie ein W-lan Modul implementiert werden, um auf externe Ressourcen zugreifen zu können.

SD-Karten Slot
Als interner Speicherplatz für das Betriebssystem und für Musik Videos sollen 2 SD-Karten Slots implementiert werden.
Bedienung über kapazitive Tasten
Die Bedienung der Mainstation soll, angelehnt an den Ipod, über kapazitive Tasten erfolgen.

2.3.3.2. xRemote

Stromversorgung

Als Stromversorgung dient ein Lithium Polymer-Akku.

Bedienung

- Touchscreen
  Die Bedienung der Fernbedienung soll durch einen Touchscreen ermöglicht werden.
- kap. Tasten
  Als Alternative zum Touchscreen sollen kapazitive Tasten in die Fernbedienung implementiert werden, dadurch soll man in der Lage sein durch das Menü der Mainstation zu navigieren.
- Flip-Control
  Als Hightlight soll die Navigation durch das Menü durch reines Kippen in die entsprechende Himmelsrichtung der Fernbedienung erfolgen.

2.3.4. Qualitätsanforderungen

Das Gerät soll in verschiedensten Umgebungen stabil laufen und einsetzbar sein. Die Oberfläche soll intuitiv und benutzerfreundlich gestaltet werden, um ein schnelles Arbeiten zu ermöglichen, ohne Fehler in der Handhabung zu produzieren. Großer Wert wird auf einen fehlerfreien Einsatz der Software sowie der Hardware gelegt.

2.4. Pflichten

2.4.1. Zielbestimmungen

2.4.1.1. Musskriterien

Mainstation

- Wiedergabe von Videos
- Wiedergabe von Musik
- Wiedergabe über Lan und W-Lan
- Darstellung auf hochauflösendem Display
- Erweiterung der Anzeige auf einen externen Display
- diverse Aus- und Eingänge(Line-in,Line-out ,optischer Ausgang)
- Bedienung über kapazitive Tasten
- 2 SD Karten Slots
- Verbindungsmöglichkeit mit der Fernbedienung

Fernbedienung

- Verbindung zur Mainstation über Bluetooth
- innovative Bedienkonzepte
  - Bedienung über Touchscreen
  - Bedienung über kapazitive Tasten
  - Bedienung durch Kippen der Fernsteuerung
- Spannungsversorgung durch Lithium Polymer Akku
- Infrarot

2.4.1.2. Wunschkriterien

Erweiterung des Bedienkomforts durch Steuerung der Mainstation durch reine Fingergesten.

2.4.2. Produkteinsatz

2.4.2.1. Anwendungsbereiche

Das Projekt wurde für den Heimgebrauch entwickelt und stellt weiters eine willkommene Alternative jeglichen Heimkinosystem dar. Durch das futuristische Design passt sich das Gerät an jegliche Einrichtungen gekonnt an.

2.4.2.2. Zielgruppen

Das Gerät kann von Personen bedient werden, die nach einer kurzen Einarbeitungszeit in die Materie, das Gerät selbstständig bedienen können.
PCBasiskenntnisse werden vorausgesetzt.
Soweit keine weiteren Sprachen integriert sind, muss der Benutzer die Verkehrssprache Englisch zumindest verstehen.

2.4.2.3. Betriebsbedingungen

- Betriebsdauer: täglich, 24 Stunden durchgehend
- Wartungsfrei
- Neustart des Systems kann von jeder Person durchgeführt werden

2.5. Zeitplanung

Da das Projekt vom Arbeitsaufwand das Stundenpensum eines „normalen“ Maturaprojektes bei weitem überstiegen hätte, hat Bernhard bereits im Juni 2008 mit einer Analyse über die Realisierbarkeit der Punkte unserer Wunschliste begonnen.

Juni 2008	...inoffizieller Projektstart (Machbarkeitsanalyse)
August 2008	...Fertigstellung des ersten Prototypen
September 2008	...offizieller Projektstart
Dezember 2008	...Vollständiger Test der Hardware
	...Funktionsfähige Systemtreiber
Januar 2009	...Fertigstellung der finalen Hardware
März 2009	...Fertigstellen des Gehäuses
April 2009	...Fertigstellen der ersten Version der Software
Juni 2009	...Projektabschluss

2.6. Aufgabenverteilung

Da der Projektleiter Bernhard bereits fundierte Programmierkenntnisse, sowie einige Erfahrung bei Hardwareentwicklung hatte, ist seine Hauptaufgabe der Hardwaresektor sowie die Low-Level Programmierung auf Firmware bzw. Treiberebene.

Die Entwicklung des Gehäuses wird aufgrund dementsprechender Erfahrungen von Alexander erledigt. Da Bernhard durch die Entwicklung der Hardware sehr weit ausgelastet ist, wurden Alexander bereits beim Projektstart diverse Verwaltungstätigkeiten anvertraut.

Michaels Aufgabenbereich fand sich in der Entwicklung bzw. Weiterentwicklung der Hardware, der Firmware für die Fernsteuerung, sowie die Fertigung des Gehäuses.

2.6.1. Projektorganisation

Projektleiter:
- Bernhard Wörndl-Aichriedler

Projektmitglieder:
- Alexander Lindenthaler
- Michael Dullnig

Projektbetreuer:
- Prof. Dipl. Ing. Robert Vogl

2.6.2. Aufgabenbeschreibung:

Bernhard Wörndl-Aichriedler
- Entwicklungsarbeit
  - Hardware Entwicklung (Multimedia-Station)
  - Firmware Entwicklung (Multimedia-Station)
  - Linux Entwicklung – Low Level (Multimedia-Station)
- Vertiefende Grundlagenarbeit
  - Advanced techniques of Web 2.0

Alexander Lindenthaler
- Entwicklungsarbeit
  - Linux Entwicklung – Anwendungsebene (Multimedia-Station)
  - Gehäuse-Design, Entwicklung und Fertigung
  - Management
- Vertiefende Grundlagenarbeit
  - kapazitive Sensortechnik

Michael Dullnig
- Entwicklungsarbeit
  - Hardware Entwicklung (Fernsteuerung)
  - Firmware Entwicklung (Fernsteuerung)
  - Linux Entwicklung – Menüführung
- Vertiefende Grundlagenarbeit
  - integrierte Bussysteme

3. Auswahl der Hardwarekomponenten

Nun ging es an die Umsetzung des Projektes und vor allem an die Machbarkeitsanalyse unserer Ideen.

Zunächst teilten wir unser Projekt grob in zwei Gruppen ein, den xMedia Player und die xRemote. Der xMedia Player ist die Multimediastation selbst, während die xRemote die Fernsteuerung des Projektes ist. Im Laufe des Projektes kam noch ein dritter Teil, hinzu mit dem Namen xDimension, welcher von unserem Projektleiter als kleine Überraschung gedacht war.

3.1. Der xMedia Player

Nun zur technischen Umsetzung des Hauptgerätes, dem xMedia Player.

Abbildung 5: AP7000

Der erste Schritt, die Wahl eines richtigen Prozessors fiel uns nicht schwer, da Bernhard bereits zwei Refernenzboards, mit dem Atmel AP7000 bestückten NGW100, besaß, wählten wir diesen Prozessor. Dieser Prozessor gilt als ideal geeignet für multimediale Applikationen mit wenig Stromverbrauch und spielt in der Leistungsklasse eines ARM9 oder sogar darüber. Er besitzt einen 32bit RISC Kern und viele Hardwaremodule, wie zwei SD/SDHC/SDIO Interfaces, ein LCD Interface, eine AC97 Audio Interface als auch einen Bitstream Audio DAC und zwei Netzwerkschnittstellen Interfaces. Es stellte sich nur die Frage nach der Fertigung bzw. nach der Entwicklung eines Prozessorboards. Hier kamen wir jedoch zum Entschluss, dass es für Schüler ohne finanzielle Unterstützung unmöglich sei, den im 256-ball BGA Gehäuse ausgelieferten Prozessor zu verbauen, da hier mit Sicherheit mit 4 bis 5 Prototypen zu rechnen wäre, welche bei zwingend nötigen 8 Layer Leiterplatten, geschätzte 5000€ kosten würden – für Schüler unfinanzierbar.
Aus diesem Grund beschlossen wir das Refernenzboard, auf welchem alle wichtigen Anschlüsse nach außen geführt waren, in unser Projekt zu integrieren.
Hier hatten wir zunächst eine AP7000 CPU mit 32MB SDRam, 8MB parallelen Flash Speicher sowie 8MB seriellen Flash Speicher. Außerdem hatte das NGW100 bereits zwei Netzwerkschnittstellen, einen USB Anschluss, einen RS232 Anschluss und einen SD-Karten Einschub integriert.

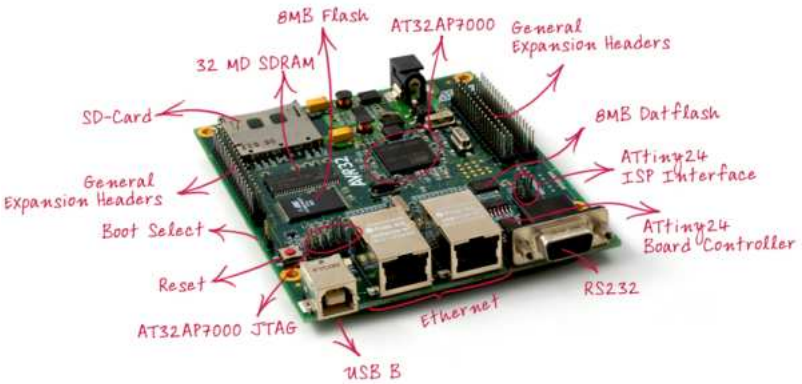


Abbildung 6: Atmel NGW100 Referenzdesign

Der zweite Schritt galt nun der Auswahl der richtigen Audio Hardware. Hier gab es grundsätzlich drei Möglichkeiten, die Benutzung des internen Bitstream DAC, einen einfachen Audio Codec oder eines höherwertigen AC97 Codec. Der interne Bitstream DAC schied nach einer kurzen Suche im Internet schnell aus, da er als sehr schwer handhabbar und instabil galt. Da wir vor allem auf die technische Realisierbarkeit achten mussten, konnten wir nur Hardware verwenden, die bereits von anderen getestet wurde. Darum hatten wir noch die Entscheidung zwischen dem STK1000 (einem Demoboard von Atmel) verbauten Atmel AT73C213 und dem Cirrus Logic CS4202, wo bereits einige sehr positiv ausgefallenen Erfahrungsberichte auf www.avrfreaks.net zu finden waren.
Wir entschieden uns schließlich für den CS4202, da er als hochwertige „Soundkarte“ galt und uns mehrere Ausgänge (unverstärkt, mit Kopfhörerverstärker, optisch digital) sowie auch mehrere Eingänge bot. Dieser Chip wird auch in PC Soundkarten verbaut. Der AT73C213 hingegen war dem CS4202 aufgrund von nur 3 Audioausgängen deutlich unterlegen.

Abbildung 7: CS4202

Mit der Wahl des Displays erfüllten wir uns den ersten Wunsch auf unserer Wunschliste. Wir wählten den Sharp LQ043 – besser bekannt unter dem Namen PSP Display. Er bietet eine Auflösung von 480x272 Pixel bei einer Pixeltiefe von bis zu 24bit was 2^24 verschiedenen Farben entspricht.
Beim VGA Ausgang setzten wir auf den erprobten und bereits im Atmel STK1000 eingesetzten ADV7125.

Abbildung 8: Sharp LQ043

Für die komplette Steuerung des User Interfaces des Players wurde ein ATmega1281 eingesetzt, welcher im lötbaren TQFP64 Gehäuse ausgeliefert wird und mit 128KByte Flash, der größte Atmel 8-bit Mikroprozessor ist, der mit 3.3V betrieben werden kann.

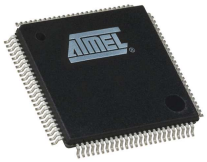


Abbildung 9: ATmega1281

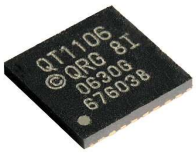


Abbildung 10:QT1106

Als nächsten Punkt auf unserer Liste standen das „I-Pod Wheel“ und die kapazitiven Sensoren. Hier hatten wir eine breites Spektrum an Möglichkeiten, die PSoCs von Cypress, mTouch von Microchip und qTouch von Quantum (d.h. eigentlich von Atmel). Da wir bereits bei den Mikroprozessoren auf Atmel setzten und da Quantum einen komplette Treiber, sowie detaillierte Beschreibungen zum Design der Sensoren anboten, haben wir uns für den QT1106 von Quantum/Atmel entschieden.

Beim Bluetooth setzten wir auf bewährte Technologie. Der BlueSMiRF von Sparkfun wurde von Bernhard bereits bei anderen Projekten eingesetzt. Es ist ein sehr einfaches Bluetooth Modul, das über eine serielle Verbindung angesprochen werden kann.

Abbildung 11: BlueSMiRF

Wireless Lan ... eines der Problemthemen unseres Projektes. Der Auswahlprozess eines geeigneten Wireless Moduls dauerte beinahe zwei Monate, dass Problem vor dem wir

standen war die Kompatibilität zu unserem Prozessor und die Unterstützung des aktuellen Linux Kernel.
Wir wollten auf jeden Fall einen 802.11g Wireless mit 54Mbit Datendurchsatz. Der Linux Kernel bietet vollen Support für die Marvell 88W8385 und die 88W8686 Chips, darum wollten wir ein Modul mit diesen Chipsätzen. Jedoch gab es auch hier unterschiedliche Module, vor allem was die Ansteuerung betrifft. Einerseits ein paralleles PCI ähnliches Interface sowie das vollständig proprietäre SDIO Protokoll und eine mit GSPI, eine nicht sehr quelloffene SPI Art. Nach langer Überlegung kamen wir zum Entschluss, dass das parallele Interface für uns, aufgrund akutem Pinmangels, nicht verwendbar war. Auch das GSPI verlor an Attraktivität. Wir hatten weder Informationen ob der Linux Treiber dieses Protokoll überhaupt unterstützte, noch ob es überhaupt verwendet wird.
Wir entschieden uns schließlich für SDIO, das Protokoll, dass auf dem SD Standard aufbaut und für Außenstehende komplett unzugänglich ist, da darüber hinaus keine Datenblätter oder Spezifikationen zu finden (außer man tritt der SD Association bei, was natürlich mit einem finanziellen Aufwand verbunden ist)sind. Wir waren also zu 100% auf die Funktionstauglichkeit des Linux Treibers angewiesen. Das Finden des geeigneten Moduls gestallte sich als äußerst schwierig. Am Ende unserer Suche gab es die Auswahl zwischen zwei Modulen, einem Modul von embeddedworks.com, einem US-Amerikanischen Hersteller und einem Modul von Zcomax einer international tätigen Firma.
Das Bestellen der Module bei Embeddedworks ist relativ einfach, nur der Preis mit 100$ pro Modul zuzüglich rund 150$ Versand und dem Zoll ist jedoch überzogen. So entschieden wir uns für das XG-182M von Zcomax. Das Beschaffen des Moduls stellte sich als äußerst schwierig da, die minimale Bestellmenge bei einigen 1000 Einheiten liegt und nur an Firmenkunden verkauft werden. Nach einigen Telefonaten mit der Liegenschaft in England und einem außergewöhnlich netten Verkäufer, bekamen wir zwei dieser Module zu einem wirklich sehr niedrigen Preis, jedoch ohne Dokumentation und Treiber.

Abbildung 12: XG-182M

3.2. Die xRemote

Die xRemote- Eine kompakte und moderne Fernsteuerung mit einem etwas anderen Konzept.

Als Hauptprozessor kommt der im xMedia Player als UI- & Boardcontroller verbaute ATmega1281 zum Einsatz, da er unsere Hardwareanforderungen perfekt trifft und wir viele Firmware Komponenten vom Boardcontroller auch bei der xRemote verwenden konnten.

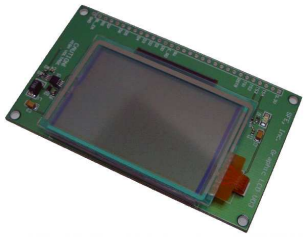


Abbildung 13: CFAX

Als Anzeige wird ein monochromer grafischer LCD von CrystalFontz verwendet, der 128x64 Pixel groß ist und bereits mit einem resistiven Touchscreen bestückt ist. Diesen wählten wir nur aus Mangel an Alternativen in dieser Größenklasse.

Um die Funktionalität der Wiimote nachzubilden, begaben wir uns auf die Suche nach einem geeigneten Beschleunigungssensor, der möglichst einfach zu handhaben war, wenig periphere Bauteile benötigte und möglichst wenig Platz auf der Leiterplatte benötigte. Unsere Wahl fiel auf LIS3LV02DQ von ST. Da er neben einem SPI Interface auch über ein I²C (oder TWI – Two Wire Interface) Interface verfügt, was für die Einbindung in unsere Hardware ideal war.
Er erforderte bis auf die Standard Stützkondensatoren keine peripheren Bauteile und wird in einem QFPN-28 Package geliefert.

Wieder auf Grund der Wiederverwendbarkeit der Bauteile und Treiber kam auch in der xRemote der QT1106 als Controller für die kapazitiven Sensoren zum Einsatz. Das gleiche galt auch bei der Wahl des Bluetooth Moduls. Hier wurde wieder das Sparkfun BlueSMiRF Modul verwendet.

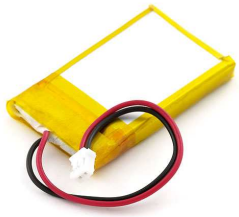


Abbildung 14: LiPo

Mobilität ist einer der wichtigsten Aspekte bei der Entwicklung einer Fernsteuerung, diese kann nur durch einen Leistungsstarken Akku erreicht werden. Wir entschieden uns für Single-Cell Lithium Polymere Akku Packs – die mit einer Nennspannung von 3.7V und einer Kapazität von 860mAh bzw. 1100mAh genau unseren Anforderungen entsprachen. Da wir auf unsere Systemspannung von 3.3V mit einem LiPo nur eine Spannungsdifferenz von maximal 0.4V erreichen, mussten wir einen speziellen Low-Dropout Voltage Regulator verwenden. Wir entschieden uns für den Micrel MIC5205, der bis zu einem Dropout von 150mV arbeitet. Zum Laden des Akkus wurde ein perfekt geeigneter Maxim MAX1551 eingesetzt, der es ermöglicht, den Akku einerseits über USB und andererseits über eine alternative Spannungsquelle zu laden.

USB, einer der Knackpunkte der xRemote. Hier setzten wir beim ersten Prototypen zunächst auf eine softwaremäßige USB Implementation. Diese erschien zunächst zwar funktionstüchtig, jedoch nach dem ersten Prototype, stellte sich schnell heraus, dass die Prozessorlast der USB Routinen viel zu hoch war. Daher entschieden wir uns im zweiten Schritt für den Silabs CP2102 einer USB to UART Bridge.

Abbildung 15: µSD

Da Grafiken im Flash des Prozessors sehr viel Platz verbrauchen, beschlossen wir die Fernsteuerung mit einem microSD Karten Slot auszustatten, welcher über SPI angesprochen wird.

3.3. xDimension

Als Innovation wollte sich der Projektleiter Bernhard einen langen Traum verwirklichen. Der Film „Minority Report “ inspirierte ihn dermaßen, dass er ein Gerät nur mittels Fingerbewegungen steuern wollte, nicht etwa mit Druck auf Sensoren. Mit anderen Worten, wenn der Benutzer seine Finger nach rechts zieht, selektiert er im Menü den nächsten Punkt, wenn er sein Finger danach nach unten zieht, wählt er den entsprechenden Punkt aus. Genau diese Gesture Control wurde laut unseren Informationen noch nie für Consumer Elektronik eingesetzt, geschweige denn zur Steuerung eines Multimediaplayers.

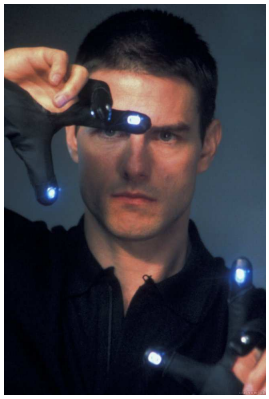


Abbildung 23: Minority Report

Da nicht feststand, wie fordernd der xMedia Player und die xRemote werden würden, wurde xDimension zunächst nicht offiziell in das Projekt aufgenommen.

Als später einige große Hürden des Projektes überwunden waren, konnten wir einige Stunden bzw. Tage in den Projektteil mit dem Namen xDimension investieren.

Ziel des Projektteils xDimension war es, Finger, die mit speziellen Markierungen versehen sind, verfolgen zu können, bzw. Gesten der Punkte/Finger zu erkennen und an den xMedia Player weiterzuleiten.

Zur Realisierung im Detail gab es viele Möglichkeiten. Das Konzept war jedoch recht schnell klar. Ein starkes Infrarotpanel strahlt auf die Finger des Benutzers, die mit Infrarotreflektoren bestückt sind – die Lichtstrahlen werden zurückgeworfen und von einer Kamera aufgenommen. Nun müssen die Daten noch verarbeitet werden und die Gesten erkannt werden – der schwierigste Teil des Projektes.

Es gab grundsätzlich zwei Ansätze dieses Projekt anzugehen, eine Möglichkeit, ist es eine Kamera per Firewire oder USB an einen kleinen PC anzuschließen, und dort mit der entsprechenden Software die Kameradaten zu verarbeiten. Als zweiter Ansatz kam das zweite NGW100 – Linux Board in Betracht, der AP7000 ist mit einem Imagesensorinterface ausgestattet, das es ermöglicht, Daten von einer angeschlossenen Kamera zu verarbeiten, um sie später zu analysieren.

Diese Methoden waren jedoch bei weitem zu kompliziert, zu unhandlich - schlicht und einfach unrealisierbar. Darauffolgend viel unser Augenmerk wiederum auf den Controller der Nintendo Wii – der Wiimote. Im großen weiten Internet fanden wir eine Beschreibung des Innenlebens einer Wiimote - ... ein Bluetooth Module, ein Beschleunigungssensor, ein EEPROM, ein Infrarot Imagesensor, ... – ein Infrarot Imagesensor? Genau dieser Infrarotsensor kam genau richtig für unser Projekt. Er ermöglicht Infrarotpunkte (sogenannte Blobs) aufzuspüren und zu verfolgen - nicht nur einen Punkt, sondern die stärksten vier Infrarotblobs. Wohl der größte Vorteil des Sensors ist die Ansteuerung, die über I^2C so einfach und unkompliziert wie nur irgend möglich ist. Ein Nachteil dieses Sensors liegt darin, dass er für Normalpersonen nicht erhältlich ist, und es keinerlei offizielle Dokumentation gibt.

Da der Sensor nicht als Einzelstück erhältlich ist, war der erste Schritt, der Tod eines der innovativsten Eingabegeräte unserer Zeit, einer nagelneuen Wiimote. Da wir nicht die Zeit und das Geld hatten ein komplett neues Mikrocontrollersystem zu entwickeln, entschlossen wir uns das Ganze als Zusatzmodul für die Leistungsstarke xRemote zu realisieren. Nach wenigen Wochen Entwicklungszeit hatten wir ein fertiges Infrarotpanel und ein Trägermodul für den Infrarotsensor.

Über das Elektronikmagazin Elektor kamen wir schließlich zum Treiber für den Imagesensor und nach kürzester Zeit funktionierte der erste Prototyp von xDimension und erfasste die Position von bis zu 4 Fingern gleichzeitig.

Nach weiteren Tagen bzw. Wochen Arbeit funktionierte die erste Testversion der Gestikerkennung und der xMedia Player konnte in seinen Grundfunktionen mittels Fingerbewegungen gesteuert werden.

Der aktuelle Stand der Dinge, betreffend xDimension ist relativ gut – die erste Testversion funktioniert – zwar mit leichten Ungenauigkeiten bei Gestikerkennung – aber sie funktioniert.

4. Der Hardwareaufbau

4.1. xMedia Player

Die Entwicklung der Hardware gestaltete sich zwar als sehr fordernd und zeitlich aufwändig, jedoch gab es beim Erstellen des Stromlaufplans nur einige kleinere Knackpunkte.

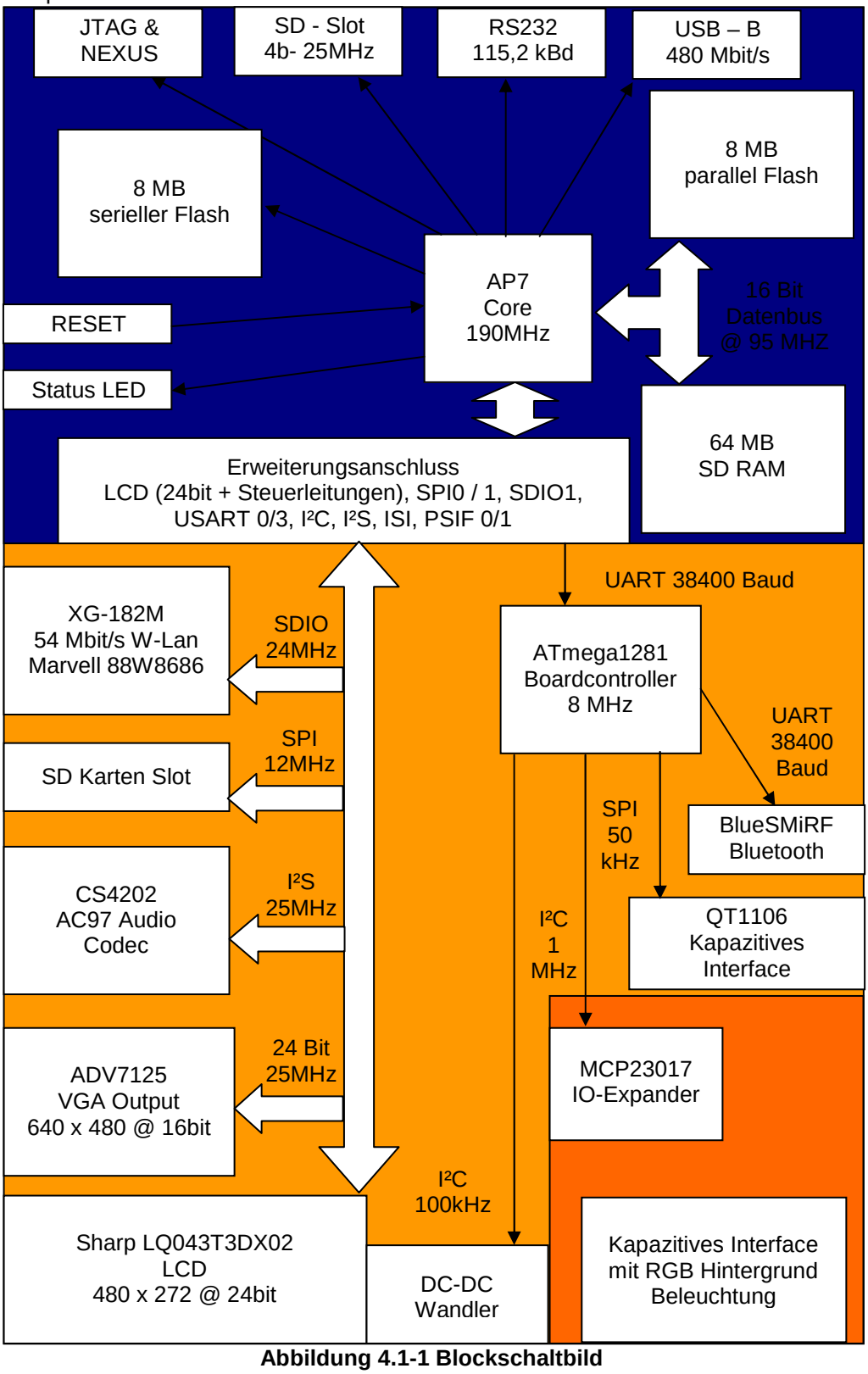


Abbildung 4.1-1 Blockschaltbild

4.1.1. Das NGW100

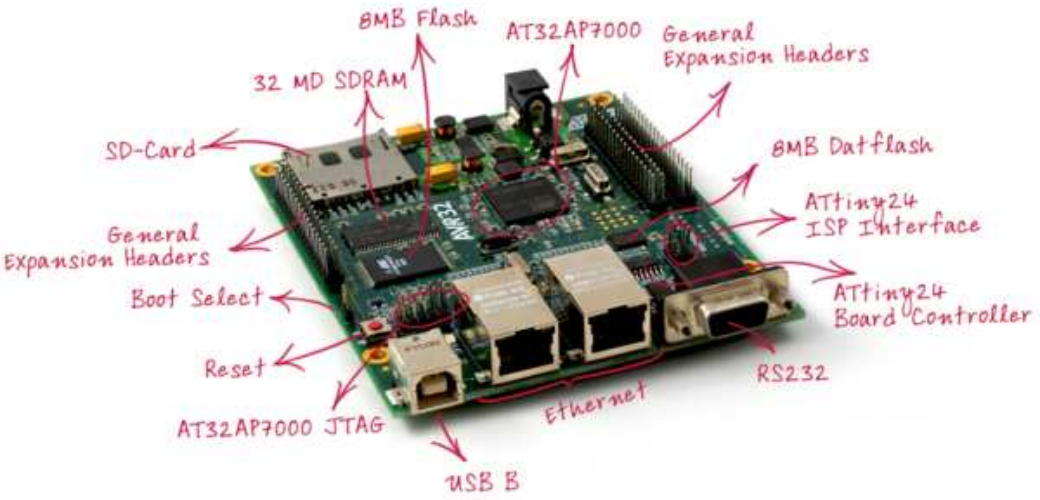


Abbildung 4.1.1-1 NGW100 Übersicht

Als Basis für unser Projekt diente wie bereits erwähnt, das NGW100, ein Referenzdesign des Prozessorherstellers Atmel. Das NGW100 schien für uns aus mehreren Gründe die ideale Basis, einerseits ist es mit rund 80€ sehr kostengünstig zu erstehen, es ist ohne weitere Umwege auch in Österreich zu erstehen und es ist mit seinen nur 100 x 120 mm in der ideal Größe, um in unserem Projekt benutzt zu werden.

Das NGW100 bot uns in der Grundausstattung bereits folgende Hardwarekomponenten:

- Die AP700 CPU – Der Prozessor des Projektes, der standardgemäß auf 140 bzw. auf 150 MHz (abhängig von der Firmware Version) läuft.
- 32 MB SD RAM – Nach dem Kauf ist das NGW100 mit einem 32MB großen SDRAM Chip der Firma Atmel bestückt, da es im weiteren Projektverlauf jedoch zu eine Speicherknappheit kam wurde dieser im Projektverlauf gegen ein 64MB Modul ausgetauscht. Der RAM ist mit einem nur 16 Bit breiten Speicherbus an der Prozessor angebunden, was das Nadelöhr des Systems darstellt.
- 8MB Parallel Flash – Zum Speichern des Betriebssystems des NGW100 wird ein schneller parallel angebundener Flash Speicher verwendet.
- 8MB Serieller Dataflash – Um zusätzliche Daten, wie Einstellungen zu speichern, wird ein über SPI Bus angebundener Dataflash, ebenfalls von der Firma Atmel, verwendet. Dieser Dataflash ist relativ langsam und daher für unser Projekt eher ungeeignet.
- Einen SD Karten Slot – Der SD Kartenslot ist einer der zentralen Punkte des Projektes, er ist über den 4 Bit breiten SD/SDIO Bus an den Hauptprozessor angebunden. Dieser SD Kartenslot wird, da der Speicher auf dem internen Flash sehr knapp bemessen ist, für das Betriebssystem verwendet.
- Zwei 10/100 MBit/s Netzwerk Anschlüsse – Da das NGW100 als Referenzdesign für einen Gateway, der zwei Netzwerke mit einander verbindet, gedacht war, ist das NGW100 mit zwei 100 MBit/s Netzwerkanschlüssen ausgestattet. Da die Last auf den Prozessor bei zwei aktiven Netzwerkanschlüssen zu hoch wäre, wurde die zweite Schnittstelle im Projektverlauf deaktiviert.
- Einen USB – B Anschluss – Dieser dient dazu um das NGW100 mit einem PC zu verbinden, es kann so zum Beispiel als Massenspeichergerät dienen.
- Eine RS232 Anschluss – Dieser serielle Anschluss ist die Kommunikationsschnittstelle mit dem NGW100 während der Entwicklungsphase, sie ermöglicht es direkt auf das System zuzugreifen und über die Konsole mit ihm zu interagieren.
- Einen JTAG Anschluss – Der JTAG Anschluss dient zum Programmieren und Debuggen des Hauptprozessors, er wird jedoch für das Endprodukt nicht benötigt.
- 3 Erweiterungsanschlüsse mit insgesamt 63 IO Leitungen.

4.1.2. Die Erweiterungsanschlüsse

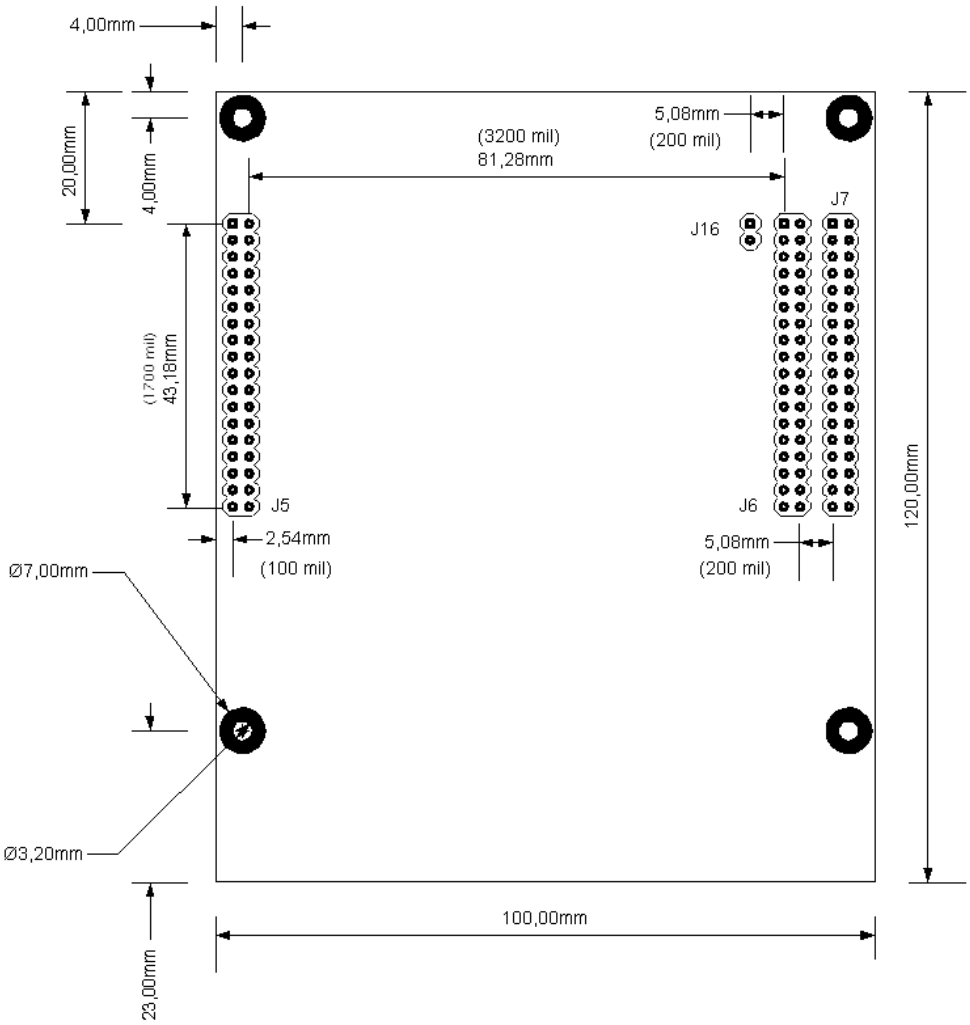


Abbildung 4.1.2-1 Erweiterungsanschlüsse

Das NGW100 ermöglicht es über insgesamt drei Stiftleisten mit je 38 Pins eine eigene Peripherie anzuschließen. Auf diese Stiftleisten hat der Prozessorhersteller die meisten Leitungen für die vom NGW100 nicht benutzte Peripheriekomponenten herausgeführt. Dies ermöglicht es uns ein Daughterboard zu erstellen, das nun neue Peripherie über diese Stiftleisten an das NGW100 anbindet, wie zum Beispiel den LCD oder den Audio Codec.

Neben den drei 38 poligen Stiftleisten im 2,54er Raster mit den Bezeichnungen J5, J6 und J7, ist auf dem NGW100 außerdem noch ein zweipoliger Anschluss mit der Bezeichnung J16 herausgeführt. Auf diesem liegt die gleichgerichtet Versorgungsspannung des an das NGW100 angeschlossenen Steckernetzgerätes an.

Für nähere Informationen über die interne Beschaltung des NGW100 liegt der Stromlaufplan der Revision B des NGW100 bei.

4.1.2.1. J5

Der Expansion Header mit dem Namen „J5“ ermöglicht den Anschluss diverser Busse, wie zum Beispiel des SPI0/1, der USART0, des ISI und des SSC, sowie Teile des MCI1 (SDIO) Busses.

Das Pinout des kompletten Anschlusses sieht wie folgt aus:

alt. 2	alt. 1	port	pin	pin	port	alt. 1	alt. 2
	3.3V		1	2		GND	
SSC1 - RX_FRAME	SPI0 - MISO	PA00	3	4	PA01	SPI0 - MOSI	SSC1 - TX_ FRAME
SSC1 - TX_CLOCK	SPI0 - SCK	PA02	5	6	PA03	SPI0 - NPCS0	SSC1 - RX_CLOCK
SSC1 - TX_DATA	SPI0 - NPCS1	PA04	7	8	PA05	SPI0 - NPCS2	SSC1 - RX_DATA
USART0 - RTS	TWI - SDA	PA06	9	10	PA07	TWI - SCL	USART0 - CTS
USART0 - RXD	PSIF - CLOCK0	PA08	11	12	PA09	PSIF - DATA0	USART0 - TXD
PWM - PWM2	SSC0 - RX_FRAME	PA21	13	14	PA22	SSC0 - RX_CLOCK	PWM - PWM3
TIMER1 - A0	SSC0 - TX_CLOCK	PA23	15	16	PA24	SSC0 - TX_FRAME	TIMER1 - A1
TIMER1 - B0	SSC0 - TX_DATA	PA25	17	18	PA26	SSC0 - RX_DATA	TIMER1 - B1
TIMER1 - CLK0	SPI1 - NPCS3	PA27	19	20	PA28	PWM - PWM0	TIMER1 - A2
TIMER1 - B2	PWM -PWM1	PA29	21	22	PA30	SM - GCLK0	TIMER1 - CLK1
TIMER1 - CLK2	SM - GCLK1	PA31	23	24	PB00	ISI - D0	SPI1 - MISO
SPI1 - MOSI	ISI - D1	PB01	25	26	PB02	ISI - D2	SPI1 - NPCS0
SPI1 - NPCS1	ISI - D3	PB03	27	28	PB04	ISI - D4	SPI1 - NPCS2
SPI1 - SCK	ISI - D5	PB05	29	30	PB06	ISI - D6	MMCI - CMD1
USART1 - CTS	SPI0 - NPCS3	PA20	31	32		N.C.	
	3.3V		33	34		GND	
	3.3V		35	36		GND	

In unserem Projekt wurde jedoch nur das SPI0 Interface, für den Anschluss des seriellen Flashs, das SPI1 Interface für den Anschluss des zweiten SD Kartenslots und das SDIO Interface für den Anschluss des WIFI Moduls verwendet. Der „Manual Wire“ Kommentar bei der „SDIO-CLK“ Leitung bedeutet, dass diese Leitung mittels einer Drahtbrücke am NGW100 mit dem Anschluss verbunden werden musste, da die für WIFI unbedingt erforderliche Taktleitung nicht auf den Erweiterungsanschluss herausgeführt wurde. Das Tastatur / Maus Interface PSIF0 wird nur im Revision 1 Prototyp verwendet, wurde jedoch in den Revisionen 3 und 4 aus Layoutgründen entfernt.

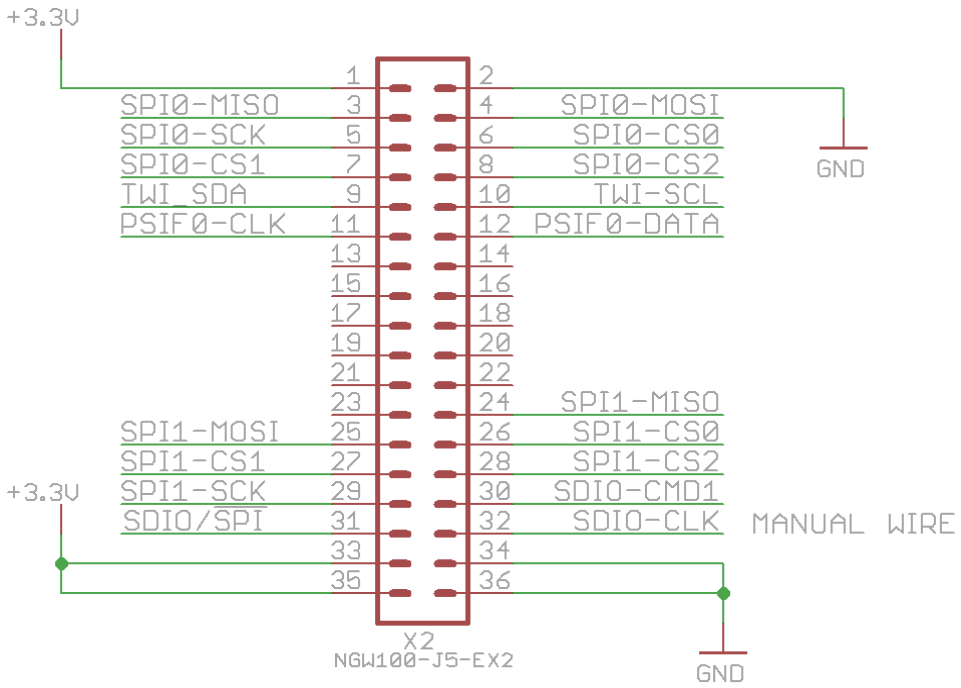


Abbildung 4.1.2.1-1 Erweiterungsanschluss J5

4.1.2.2. J6

Auf dem Erweiterungsanschluss mit dem Namen „J6“ sind unter anderem Teile des ISI (Image Sensor Interface), die USART 3, das komplette I²S Interface, die zweite Hälfte des MCI1 (SDIO) Interfaces und das PSIF1 herausgeführt. Außerdem hat dieser Port die Ausgänge für den prozessorinternen Bitstream DAC und den WAKE Pin, der es ermöglicht den Prozessor aus dem Ruhezustand wieder aufzuwecken.

alt. 2	alt. 1	port	pin	pin	port	alt. 1	alt. 2
	3.3V		1	2		GND	
	3.3V		3	4		GND	
	N.C.		5	6		N.C.	
MMCI - D4	ISI - D7	PB07	7	8	PB08	ISI - HSYNC	MMCI - D5
MMCI - D6	ISI - VSYNC	PB09	9	10	PB10	ISI - PCLK	MMCI - D7
ISI - D8	PSIF - CLOCK1	PB11	11	12	PB12	PSIF - DATA1	ISI - D9
ISI - D10	SSC2 - TX_DATA	PB13	13	14	PB14	SSC2 - RX_DATA	ISI - D11
USART3 - CTS	SSC2 - TX_CLOCK	PB15	15	16	PB16	SSC2 - TX_FRAME	USART3 - RTS
USART3 - TXD	SSC2 - RX_FRAME	PB17	17	18	PB18	SSC2 - RX_CLOCK	USART3 - RXD
USART3 - CLK	SM - GCLK2	PB19	19	20	PB20	DAC - DATA1	AUDIOC - SDO
AUDIOC - SYNC	DAC - DATA0	PB21	21	22	PB22	DAC - DATAN1	AUDIOC - SCLK
AUDIOC - SDI	DAC - DATAN0	PB23	23	24	PB24	DMAC - DMARQ0	NMI - NMI_N
IRQ - EXTINT0	DMAC - DMARQ1	PB25	25	26	PB26	USART2 - RXD	IRQ - EXTINT1
IRQ - EXTINT2	USART2 - TXD	PB27	27	28		WAKE_N	
	N.C.		29	30		N.C.	
	N.C.		31	32		N.C.	
	3.3V		33	34		GND	
	3.3V		35	36		GND	

In unserem Projekt werden nun des Datenleitungen des MCI Interfaces (mit dem Titel D4 bis D7) zum Anschluss des W-LAN Moduls verwendet. Außerdem werden die RX und TX Leitungen der USART3 für die Kommunikation mit dem Boardcontroller verwendet. Zum Anschluss des Audio Codecs wird das SPI ähnliche I²S Interface des Prozessors verwendet. Um die Aktivität der SD Karte zu ermitteln, wurden die Leitungen „SD_CD“ (Card Detect) und „SD_WP“ (Write Protect) als normale IO Pins verwendet. Damit der Boardcontroller den Hauptprozessor aus dem Tiefschlaf (Hibernate) aufwecken kann wurde außerdem der WAKE Pin an den Boardcontroller angebunden.

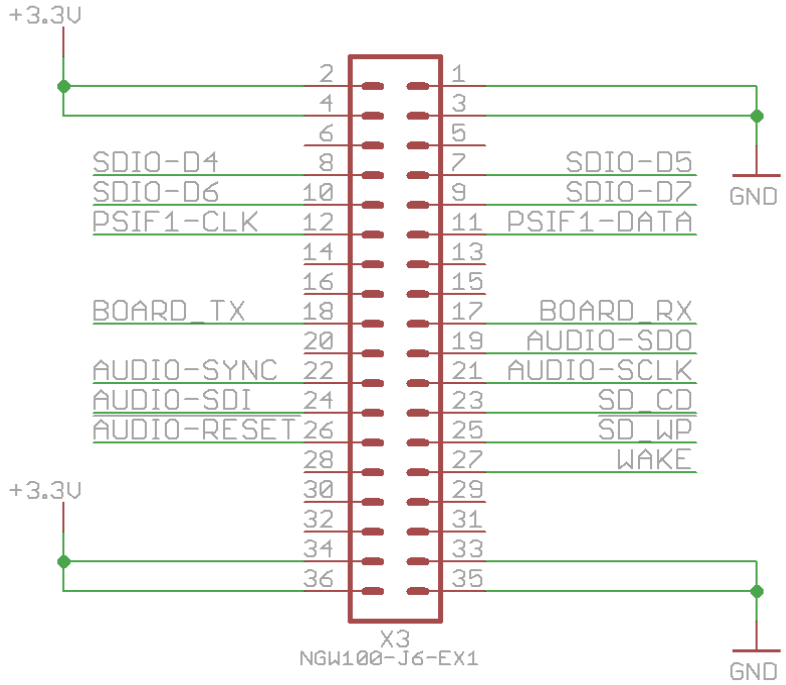


Abbildung 4.1.2.2-1 Erweiterungsanschluss J6

4.1.2.3. J7

Der „J7“ Anschluss hat eigentlich nur eine Aufgabe, den Anschluss von Grafikhardware, wie einem LCD oder einem VGA Ausgang. Aus diesem Grund hat der Prozessorhersteller hier auf den oberen Teile des EBI (External Bus Interface), der gleichzeitig Teil des alternativen Pinouts des LCDC (LCD Controllers) ist herausgeführt.

alt. 2	alt. 1	port	pin	pin	port	alt. 1	alt. 2
EBI - D19	LCD - D0	PE03	1	2	PE04	LCD - D1	EBI - D20
EBI - D21	LCD - D2	PE05	3	4	PE06	LCD - D3	EBI - D22
EBI - D23	LCD - D4	PE07	5	6	PC31	LCD - D5	
	LCD - D6	PD00	7	8	PD01	LCD - D7	
EBI - D24	LCD - D8	PE08	9	10	PE09	LCD - D9	EBI - D25
EBI - D26	LCD - D10	PE10	11	12	PE11	LCD - D11	EBI - D27
EBI - D28	LCD - D12	PE12	13	14	PD07	LCD - D13	
	LCD - D14	PD08	15	16	PD09	LCD - D15	
EBI - D29	LCD - D16	PE13	17	18	PE14	LCD - D17	EBI - D30
EBI - D31	LCD - D18	PE15	19	20	PE16	LCD - D19	EBI - A23
EBI - A24	LCD - D20	PE17	21	22	PE18	LCD - D21	EBI - A25
	LCD - D22	PD16	23	24	PD17	LCD - D23	
EBI - D17	LCD - DVAL	PE01	25	26	PE02	LCD - MODE	EBI - D18
	LCD - HSYNC	PC20	27	28	PC21	LCD - PCLK	
	LCD - VSYNC	PC22	29	30		N.C.	
	N.C.		31	32		N.C.	
	3.3V		33	34		GND	
	3.3V		35	36		GND	

In unserem Projekt wird nun eigentlich der ganze Port für die Video Hardware verwendet, der 24Bit Datenbus und die Timingsignale HSYNC, VSYNC und die Pixel Clock (PCLK) wurden sowohl beim VGA Ausgang als auch beim LCD verwendet. Während die DVAL Leitung nur beim VGA Ausgang verwendet wurde. Der 47Ω Widerstand in der Pixelclock Leitung dient zur Verbesserung der Signalqualität auf der im VGA Modus mit nahezu 25 MHz getakteten Leitung.

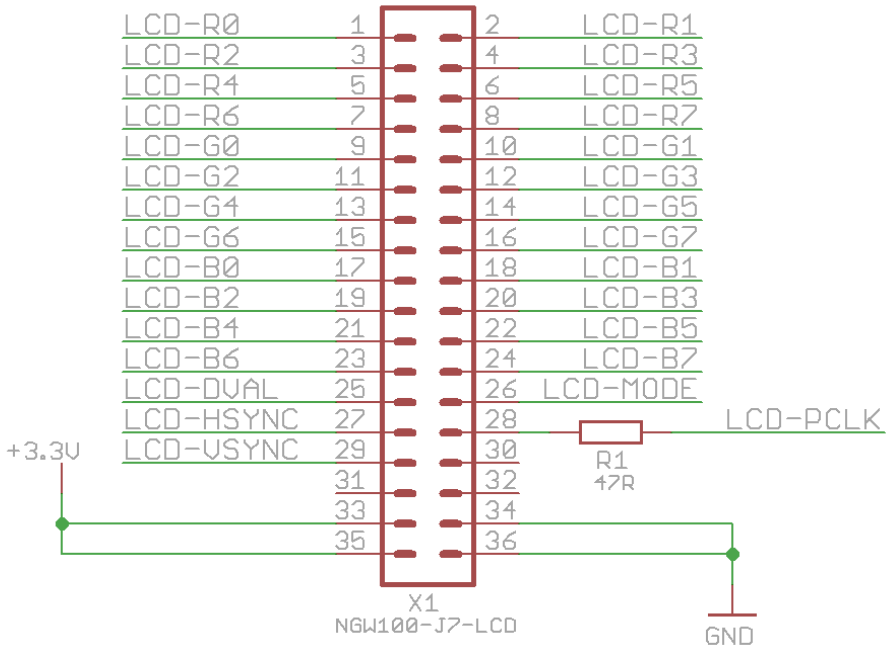


Abbildung 4.1.2.3-1 Erweiterungsanschluss J7

4.1.2.4. J16

Auf dem Anschluss J16 des NGW100 ist die gleichgerichtete (da das NGW100 über einen Brückengleichrichter verpolungssicher gemacht wurde) Versorgungsspannung (9V bis 15V, abzüglich der Verluste für den Brückengleichrichter) herausgeführt. Diese Spannung dient uns, unter Verwendung von zwei Linearregulatoren als Analogspannung für den Display und die Audio Hardware. Da der Anschluss über ein Flachbandkabel (d.h. verpolbar) an die xMedia Hauptplatine angeschlossen wird, wird eine Diode verwendet, um die Hardware beim verpolen nicht zu zerstören.

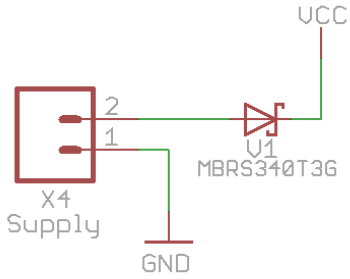


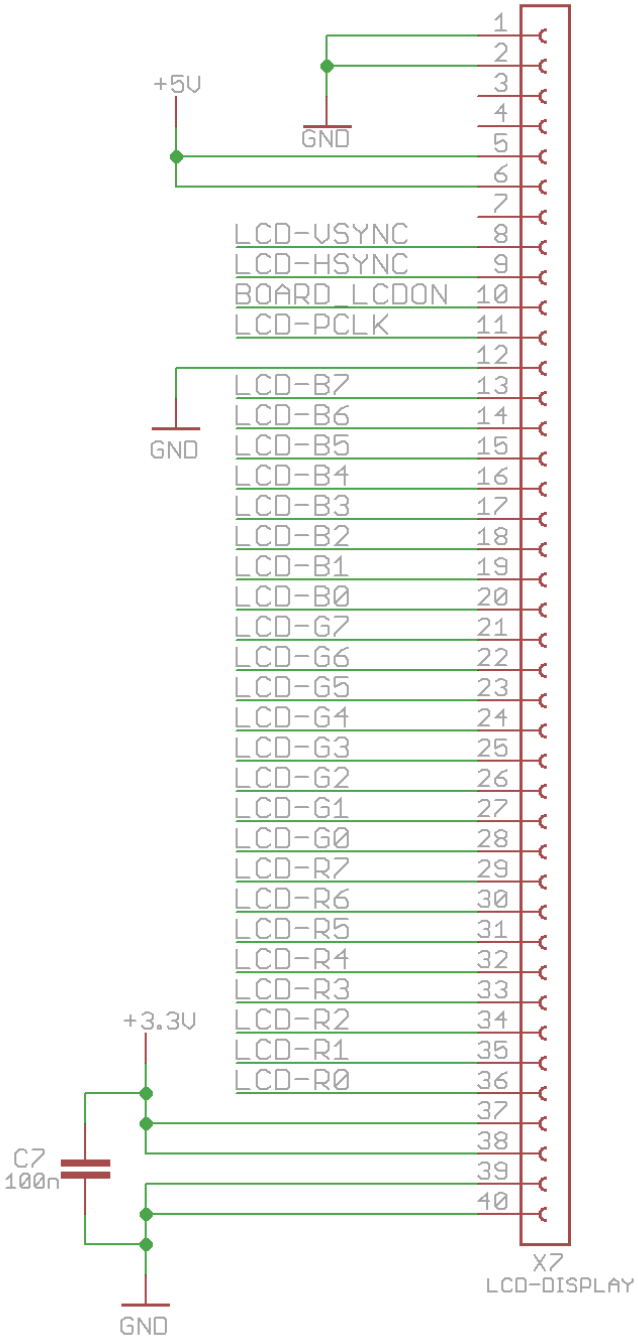
Abbildung 4.1.2.4-1 Spannung J16

4.1.3. Der LC-Display

Wie bereits erwähnt verwenden wir zu Anzeige des Menüs den qualitativ hochwertigen Sharp LQ043T3DX02 LCD, der mit seiner Auflösung von 480x272 Pixel bei einer Größe von 4,3 Zoll für unser Projekt ideal ist.

Das Pinout des LCD:

Pin	Name	Funktion
1	GND	GND(0V)
2	GND	GND(0V)
3	VCC	+2.5V power source
4	VCC	+2.5V power source
5	R0	RED Data Signal (LSB)
6	R1	RED Data Signal
7	R2	RED Data Signal
8	R3	RED Data Signal
9	R4	RED Data Signal
10	R5	RED Data Signal
11	R6	RED Data Signal
12	R7	RED Data Signal (MSB)
13	G0	GREEN Data Signal (LSB)
14	G1	GREEN Data Signal
15	G2	GREEN Data Signal
16	G3	GREEN Data Signal
17	G4	GREEN Data Signal
18	G5	GREEN Data Signal
19	G6	GREEN Data Signal
20	G7	GREEN Data Signal (MSB)
21	B0	BLUE Data Signal (LSB)
22	B1	BLUE Data Signal
23	B2	BLUE Data Signal
24	B3	BLUE Data Signal
25	B4	BLUE Data Signal
26	B5	BLUE Data Signal
27	B6	BLUE Data Signal
28	B7	BLUE Data Signal (MSB)
29	GND	GND(0V)
30	CK	Clock signal to sample each date
31	DISP	Display ON/OFF Signal
32	Hsync	Horizontal synchronizing signal
33	Vsync	Vertical synchronizing signal
34	NC	NC
35	AVDD	+5V Analog power source
36	AVDD	+5V Analog power source
37	NC	NC
38	TEST1	TEST1
39	TEST2	TEST2
40	TEST3	TEST3



Die digitale Anbindung des LCDs schien denkbar einfach, es musste lediglich der 24bit breite Datenbau und die Timingsignale HYSNC, VSYNC und PCLK an den Prozessor angebunden werden.

Um den Display über den Boardcontroller (der gleichzeitig das Powermanagement regelt) abschalten zu können, wurde das „DISP“ Signal, das zum aktivieren bzw. deaktivieren des Displays dient, mit diesem verbunden.

Die Analoge Anbindung an das System schien zunächst deutlich schwieriger, da der Display laut dem ersten Blick in das Datenblatt insgesamt drei verschiedene Spannung benötigt - 2.5V für den digitalen Teil des Displays, 5V für den analogen Teil und um die 20V bei rund 20mA für die LED Hintergrundbeleuchtung.

Bei genauer Betrachtung des Datenblattes wurde jedoch klar, dass 3.3V für die Digitalspannung noch in den „absolute maximum ratings“ des Displays liegen, und der Display somit ohne weitere Regulatoren mit der Systemspannung von 3.3V versorgt werden konnte. Durch die Versorgungsspannung von 3.3V war es nun auch zulässig, den Datenbus und die Steuerleitungen (welche mit 3.3V arbeiten) direkt, d.h. ohne Pegelumsetzer oder Spannungsteiler mit dem LCD zu verbinden.

4.1.3.1. Die Analogversorgung

Die Analoge +5V Versorgungsspannung wurde hier sehr einfach realisiert, da die Last auf dieser Power Rail nur maximal 18mA beträgt, musst hier der Effizienzfaktor nur wenig beachtet werden und des konnte mit dem MC78M05CDTG ein einfach Linearregulator verwendet werden.

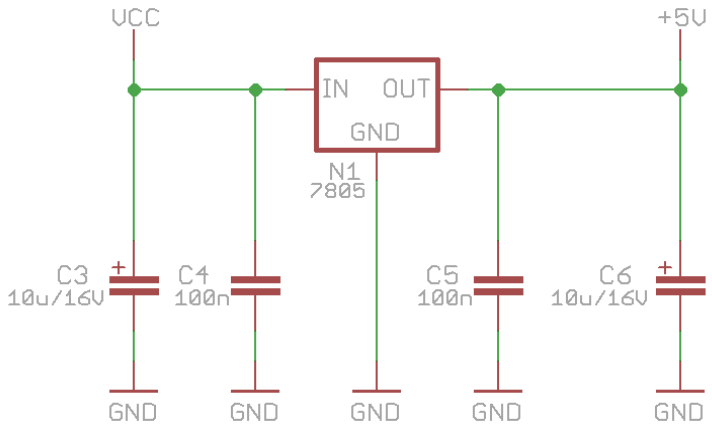


Abbildung 4.1.3.1-1 Linearregulator

Diese Bild zeigt das 7805er Derivat mit der Standardbeschaltung, mit zwei gepolten Kondensatoren (in diesem Fall Tantal Elektrolytkondensatoren) und zwei ungepolten Kunststoffkondensatoren.

4.1.3.2. Der Boost Konverter

Bereits bei der ersten Revision des Stromlaufplans galt der DC-DC Wandler zum Erzeugen der Hintergrundspannung für das LC-Display als einer dieser Knackpunkte. Der Boostkonverter sollte Spannung von den 3.3V (bzw. 7-12V) auf die benötigten ~24V bei 15mA bringen. Boostkonverter dieser Art haben mehrere Nachteile, sie sind Schaltwandler und schalten mit meist relativ hohen Frequenzen den Strom durch eine Spule an und aus, das wiederum sehr hohe elektromagnetische Störungen und auch hörbares Summen der Spule verursachen kann. Wir entschieden uns bei der ersten Hardwarerevision, nach langem Überlegen, für den LM2735 von National, da dieser gute Leistung und einfach Handhabung versprach, jedoch war dem leider nicht so. Gleich der erste Versuch brachte die Ernüchterung, der Boostkonverter ging in Rauch auf, und schmorte einen großen Krater in die Platine unseres ersten Prototype. Der Grund dafür war schnell geklärt. Der Ausgangsstromkreis war aufgrund einer schlechten Verbindung am Display unterbrochen was zum Tod des Chips führte, außerdem summte der Chip doch deutlich hörbar, bevor er sich in die ewigen Jagdgründe verabschiedete. Die nächsten Wochen suchten wir verzweifelt nach einem Ersatz, jedoch zeigte jeder Chip dasselbe Verhalten. Wir waren am Ende und Bernhard begann auf Basis eines seiner anderen Projekte selber einen Boostkonverter zu entwerfen, der unseren Anforderungen gerecht wurde. Der Boostkonverter auf Basis eines ATtiny85 Prozessors wurde eine Meisterleistung, mit einer Schaltfrequenz im Megahertzbereich und einer Spannungs- sowie einer Strombegrenzung war er sowohl zu 100% stabil, als auch für das menschliche Ohr unhörbar. Als kleines Highlight verfügt der Konverter noch über ein I²C

Interface, das es ermöglicht Spannung, Strom und Schaltfrequenz zur Laufzeit zu ändern.

Der Aufbau diese Boost Konverters ist denkbar einfach, der Timer des kleine Prozessor (der ATtiny 85) wird mittels internem RC Oszillator und interner PLL auf eine Frequenz von 64 MHz getaktet, was es ermöglicht, eine Pulbreitenmodulation mit einer Frequenz von 0.64 MHz bei 1% Einstellgenauigkeit, bzw. von 1,28MHz bei einer Einstellgenauigkeit von 2% zu erzeugen. Um wirklich einen Regler zu bauen, wurde sowohl der Ausgangsstrom, der über einen Shunt Widerstand gemessen wird, als auch die Ausgangsspannung, die über einen Spannungsteiler auf die für den ADC verwendbaren 0V – 3.3V gebracht wird, gemessen. Eine aufwändige Routine verarbeitet nun die ADC Daten und setz sie auf eine Tastfrequenzänderung um, das Reglerverhalten ist schwer kategorisierbar, ist jedoch am besten als erweiterter P Regler einordenbar.

Zur Hardware:

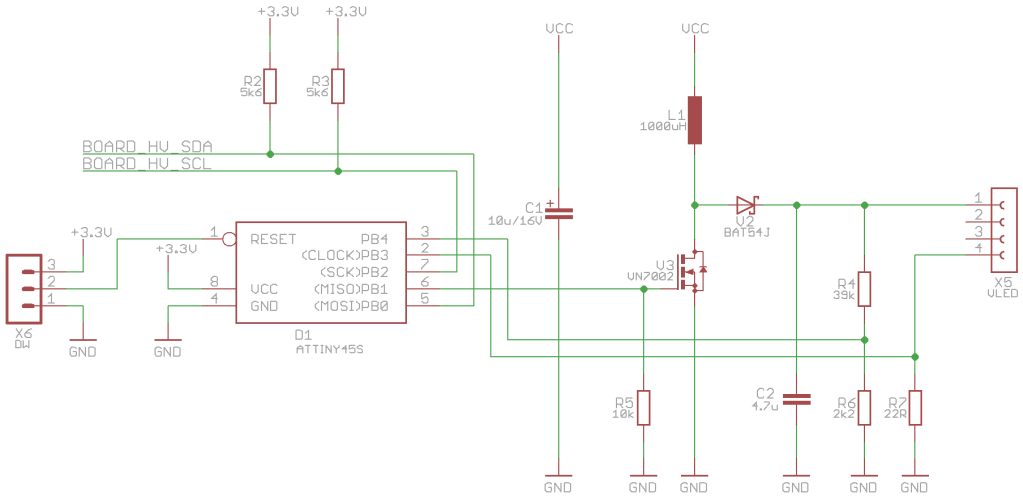


Abbildung 4.1.3.2-1 xDC Hardware

Auf der Ausgangsseite sehen wir zunächst (unter Außerachtlassung des Widerstände) des Aufbau eine Boost Konverters, bestehend aus einer 1000uH Spule, einem 2N7002 Schalttransistor, einer BAT54J Diode und einem 4.7uF Keramik Kondensators.

Zunächst einige allgemeine Überlegungen zum Aufbau des Boost Konverters:

- Der Boost Konverter sollte nahezu Lastfrei noch relativ Stabil arbeiten, d.h. er sollte einen geringen minimalen Ausgangsstrom besitzen
- Er sollte für das menschliche Ohr nicht hörbar sein, d.h. mit Schaltfrequenzen im Hundertkilohertzbereich arbeiten.
- Der Konverter sollte eine geringe Brummspannung haben und die Stabilität der Beleuchtung zu gewährleisten.
- Er sollte vom Formfaktor so klein wie möglich sein

Aus diesen Vorgaben entwickelten wir den Boost Konverter mit den oben genannten Werten.

$U_i := 6V$ Eingangspannung des Konverters (minimale Rechnung)

$U_o := 18V$ Ausgangsspannung des Konverters (Durchschnittswert)

$I_o := 20mA$ Ausgangsstrom für adequate Beleuchtung

$f := 640kHz$ Maximale mit dem Prozessor erreichbare Frequenz bei 1% Genauigkeit des Tastverhältnisses

$T := \frac{1}{640kHz}$

$L := 1000\mu H$ Gewählter Wert für die Spule

$C := 4.7\mu F$ Gewählter Wert für den Kondensator

Verlustfreie Rechnung

$p := \frac{U_o - U_i}{U_o}$	$p = 0.667$	Ungefähres Tastverhältnis, um die gewünschte Ausgangsspannung zu erreichen
$I_{omin} := \frac{U_i \cdot p \cdot (1-p) \cdot T}{2 \cdot L}$	$I_{omin} = 1.042mA$	Minimaler Ausgangsstrom um nicht in den lückenden Betrieb zu kommen.
$U_{brss} := \frac{I_o}{C} \cdot \left(1 - \frac{U_i}{U_o}\right) \cdot T$	$U_{brss} = 4.433mV$	Brummspannung des Konverters
$I_i := \frac{I_o}{p}$	$I_i = 30mA$	Ungefährer Eingangsstrom des Konverters

Wie aus der Rechnung hervorgeht erzielen die gewählten Kondensator und Spulenwerte, für unseren Konverter ideale Kenngrößen - einen minimalen Ausgangsstrom von nur ~1mA und eine Brummspannung von nur ~5mV.

Bei der Wahl des externen Schaltransistor viel unsere Wahl auf den von der Bauform sehr kleinen 2N7002, welcher einer der wenigen bei 3.3V schaltenden analogen Feld Effekt Transistoren ist, der uns einen maximalen Drain Source Strom von 115mA im Dauerbetrieb bzw. 800mA im gepulsten Betrieb bietet.

Um den Ausgangsstrom mit dem Prozessor messen zu können wurde ein Shunt Widerstand in den Stromkreis integriert, der mit 22Ω nur geringe Verluste am Ausgang verursachte, jedoch eine für den Mikroprozessor ohne Operationsverstärker messbare Spannung im Bereich um 0.4V bei einer Nennlast von 20mA verursacht.

Der Spannungsteiler für die Ausgangsspannung wurde mit 39kΩ zu 2.2kΩ so gewählt, dass bei einer Ausgangsspannung von ~20V ungefähr 1V am 10Bit ADC des Prozessors erreicht werden und die Spannung somit gut Messbar ist und gleichzeitig noch weit entfernt von der maximalen Eingangsspannung von 3.3V ist.

4.1.4. Der VGA Ausgang

Der VGA Ausgang wurde auf Basis mehrer Anleitungen aus dem Internet erstellt. Das VGA Signal wird über einen High Speed DAC von Analog Devices, der für diese Zwecke entwickelt wurde, generiert.

Dieser DAC ermöglicht es Analog auf Digital Wandlungen mit einer Geschwindigkeit von bis zu 125MHz durchzuführen.

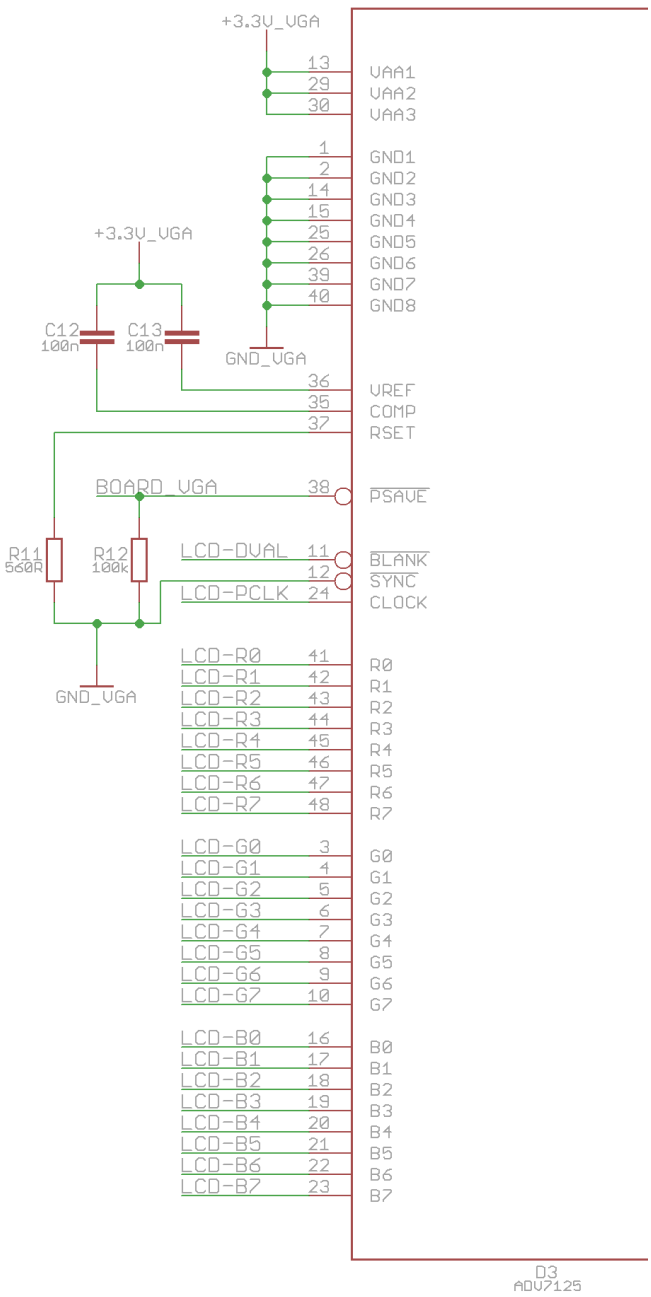
Er wurde gleich wieder Display direkt an den 24Bit breiten parallelen Bus zum LCD Controller des Prozessors angebunden und mit dem Pixel Clock Signal versorgt.

Der 24Bit Bus besteht wenn man ihn genauer betrachtet, aus drei mal 8Bit, welche jeweils eine Farbe repräsentieren.

Der IC sampelt die Daten nun, sobald ihn eine Taktflanke des PCLK Signals erreicht und legt den entsprechenden Analogwert binnen weniger Nanosekunden an die Ausgänge des Chips.

Die DVAL Leitung die an die BLANK Leitung des IC's angeschlossen ist, erzwingt den Ausgang auf das „blank" Level zu schalten (eine Referenzspannung die angibt, dass die Übertragung einer Zeile beendet ist – näheres dazu ist in den VGA Spezifikationen zu finden)

Der PSAVE Eingang ermöglicht es den Chip in einen Power Save Modus zu schalten, in dem sämtliche Ausgänge abgeschaltet werden – dieser wird wieder vom Boardcontroller gesteuert.



Der 560Ω große Widerstand am RSET Eingang des ADV7125 dient zur Regulierung des Ausgangsstroms des VGA Signals, laut Datenblatt des IC sollte dieser für ein Standard VGA Signal mit 560Ω gewählt werden.

Die Kondensatoren am VREF und COMP Eingang wurden ebenfalls der Standardbeschaltung aus dem Datenblatt entnommen.

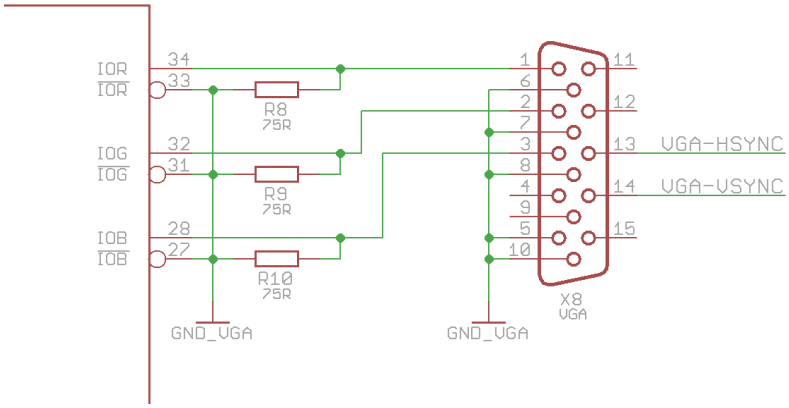


Abbildung 4.1.4-1 VGA - Ausgang

Bei der Beschaltung des Analogteils des ADV7125 vertrauten wir ebenfalls auf die Vorgaben im Datenblatt. Es wurde mit 75Ω Widerständen die parallel zu Masse geschaltet werden gearbeitet, diese sind in den Spezifikationen des VGA Signals vorgeschrieben, und rufen gemeinsam mit den 75Ω Widerständen im Monitor eine dem Ausgangsstrom des ADV7125 korrespondierende Spannung hervor.

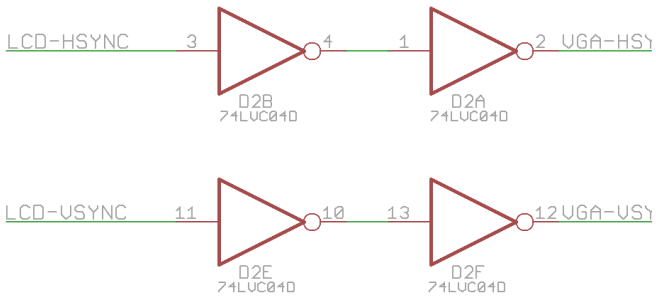


Abbildung 4.1.4-2 HSYC, VSYNC Buffer

Damit der extern angeschlossene Monitor keine Möglichkeit hat, Störeinflüsse direkt an den Monitor weiterzuleiten, werden die HSYNC und VSYNC Signale mittels doppelter Inversion gepuffert. Die mag zwar nicht die ideale Lösung sein, doch da dieses Variante bereits beim STK1000, einem Evaluation Board für den von uns verwendeten Prozessor eingesetzt wird, haben wir uns entschlossen den sicheren Weg zu wählen und auch diese Methode zu verwenden.

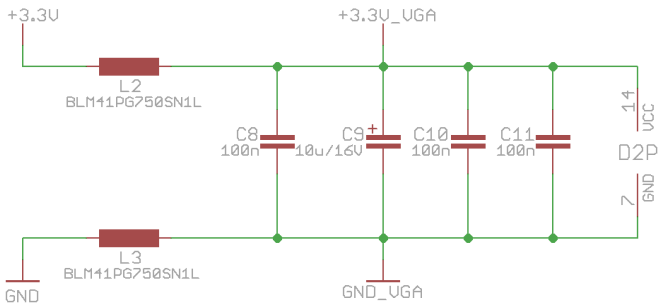


Abbildung 4.1.4-3 Supply Decoupling

Etwas Vorsicht war bei dem Entwurf der Spannungsversorgung des VGA Anschlusses geboten. Da für die Versorgung des DAC die Digitalspannung von 3.3V verwendet wurde, war Vorsicht geboten, da dieses sehr störungsanfällig und unsauber ist. Außerdem musste darauf geachtet werden, dass keine Störungen über den VGA Ausgang in das System eingebracht werden, da es hier keine Massentrennung gibt.

Wir realisierten dies durch die Entkopplung über spezielle Spulen in Kombination mit einigen Kondensatoren zur Stabilisierung. Die Beschaltung wurde Großteils wiederum aus dem Stromlaufplan des STK1000 entnommen.

4.1.5. Der AC97 Codec

Für die Wiedergabe und Aufnahme von Audiosignalen kam mit dem CS4202 von Cirrus Logic ein wirklich hochwertiger Chip zum Einsatz, der normalerweise seinen Einsatz in Desktopsystemen findet.

Der Chip bietet uns folgende Features:
- Einen Stereo Kopfhörerausgang (mit Ausgangsverstärker)
- Einen Line Out
- Einen SPDIF Ausgang (für optische Signale)
- Zwei Mono Mikrofoneingänge
- Zwei Stereo Eingänge
- Anbindung über I²S
- Digital Analog mit 20Bit Auflösung
- Analog Digital mit 18Bit Auflösung

Es war bereits bekannt, dass der CS4202 gut mit dem AP7000 zusammenarbeitete, da bereits einige Entwickler der AVR32 Gemeinde diesen Chip verwendeten und durchaus sehr positive Rückmeldungen gaben.

Es ist immer eine sehr schwierige Sache Audio Hardware so zu entwerfen, dass sie die optimale Qualität bringt und hier kam uns der Hersteller des IC's sehr zuvor. Cirrus Logic bot neben mehreren Beispielschaltungen auch ein komplettes Design mit Stücklisten auf seiner Webseite an. Dies machte es uns möglich die optimalen Bauteiletypen zu verwenden und das optimale Layout im Analogbereich zu finden.

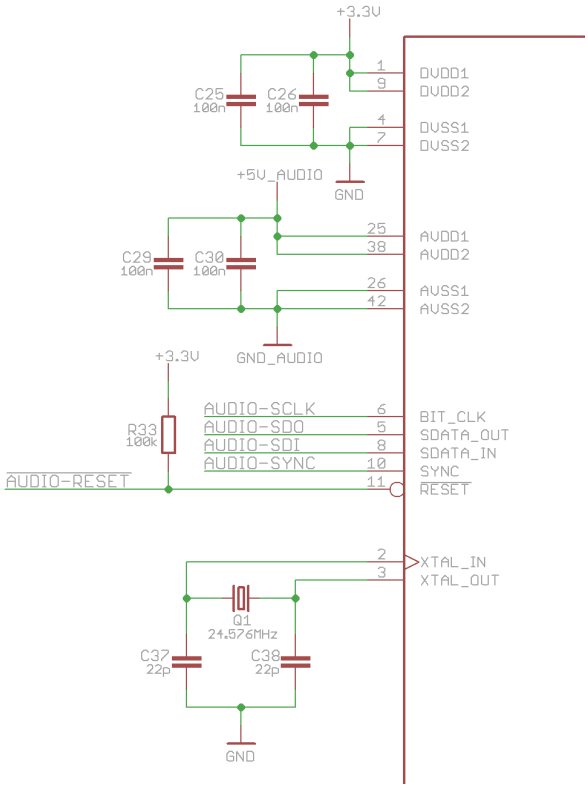


Abbildung 4.1.5-1 CS4202 Digitale Anbindung

Wie häufig, gestaltete sich die digitale Anbindung des CS4202 eigentlich sehr einfach und unproblematisch, er wurde einmal mit den 3.3V für die Digitale Subsektion des

Chips versorgt und das zweite Mal mit den, durch einen Linearregulator erzeugten, 5V für den Analogteil des Codecs.

Zur Kommunikation mit dem Prozessor wurde der CS4202 über den I²S and den Prozessor angebunden, dieser SPI ähnliche serielle Bus ermöglicht es Audiodaten mit einem nur Bus, der nur aus 5 Leitungen besteht, zu übertragen. Die Besonderheiten am I²S ist, dass hier nicht der Master, d.h. die CPU sondern der CS4202 den Takt vorgibt.

Um Arbeiten zu können benötigt der CS4202 außerdem einen Quarz, der für den Sampletakt des Chips und den Takt des I²S zuständig ist.

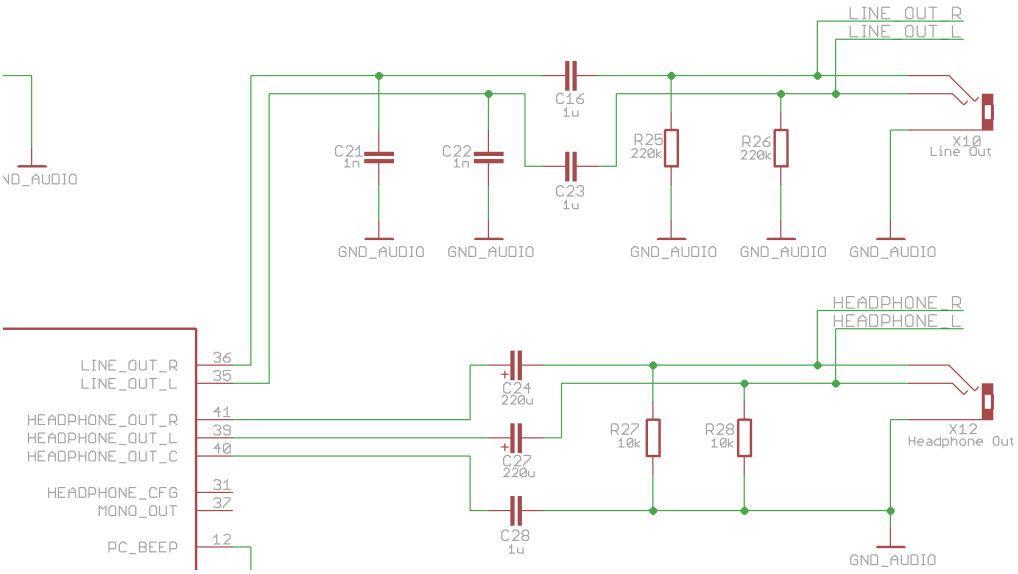


Abbildung 4.1.5-2 CS4202 Analoge Ausgänge

Die Beschaltung der analogen Ausgänge war im Grunde genommen nicht schwierig, da wir mit den Referenzdesigns von Cirrus bereits eine gute Vorgabe hatten und diese einfach eins zu eins übernahmen.

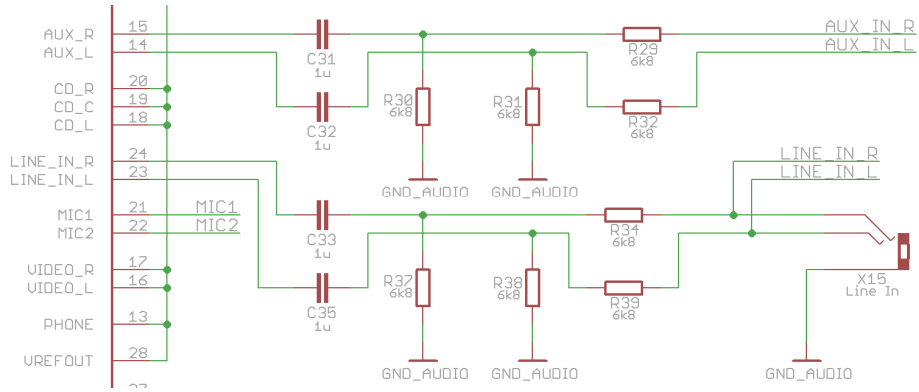


Abbildung 4.1.5-3 CS4202 Analoge Eingänge - Line

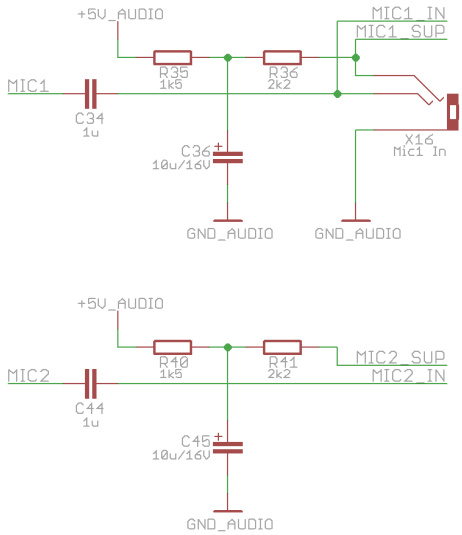


Abbildung 4.1.5-4 CS4202 Analoge Eingänge - Mikrofon

Auch im Bezug auf die Analogen Eingänge änderten wir am Referenzdesign eigentlich nur wenig, um auch hier auf der sicheren Seite zu sein.

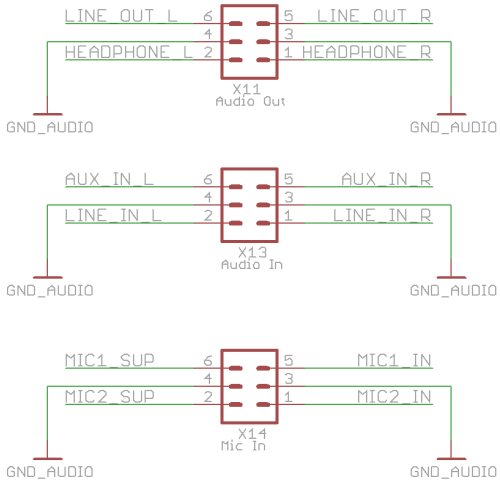


Abbildung 4.1.5-5 CS4202 Erweiterungsanschlüsse

Da unser Platine nicht genug Platz bietet, um alle Anschlüsse die der CS4202 bietet über Klinkenbuchsen zugänglich zu machen und wir nicht wollten, dass diese ungenutzt bleiben, entscheiden wir uns alle Signale auf Stiftleisten herauszuführen.

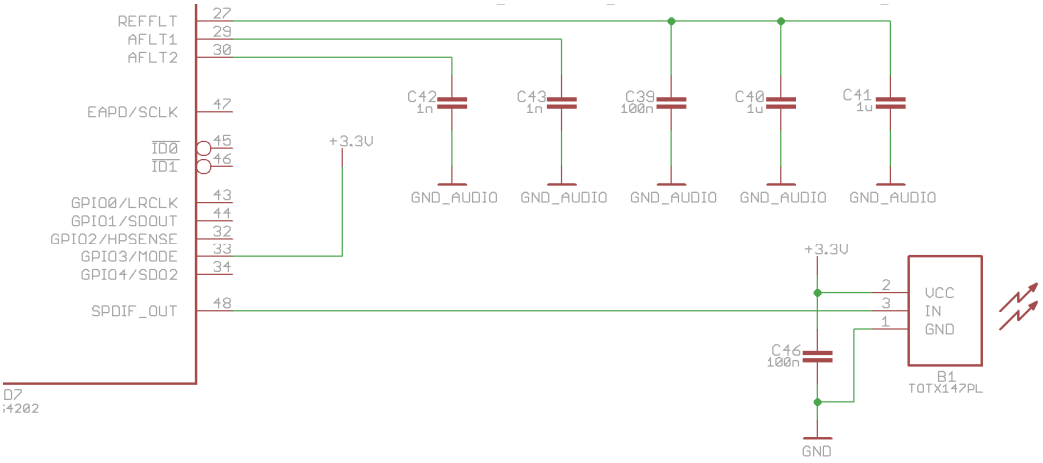


Abbildung 4.1.5-6 Optischer Ausgang & Analogbeschaltung

Neben der Beschaltung der Ein- und Ausgänge gibt das Referenzdesign auch die Kondensatorwerte für den REFFLT, AFLT1 und den AFLT2 Pin vor. Der Kondensator, bzw. die Kondensatoren an REFFLT dienen zur Stabilisierung der internen Referenzspannung, während die Kondensatoren an AFLT1 und AFLT2 als Anti-Aliasing Filter für den ADC dienen.

Der optische SPDIF Ausgang wurde mit einem TOTX147PL Transmittermodul realisiert, welches wir aufgrund des im vergleichsweise geringen Preises wählten.

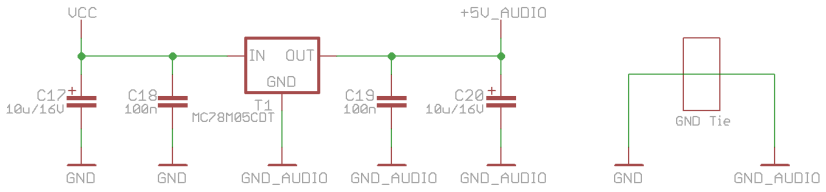


Abbildung 4.1.5-7 Spannungsversorgung

Wie eigentlich bei jedem elektronischem Gerät im Audio Bereich spielt die Spannungsversorgung eine zentrale Rolle, da das Menschliche Ohr eines der empfindlichsten Messgeräte für Störungen ist.

Um die Störungen der Analogspannung so gering wie möglich zu halten wurde hier ein zwar ineffizienter, jedoch sehr Spannungsstabiler Linearregulator verwendet. Er wurde wiederum in der Standardbeschaltung mit jeweils zwei Kondensatoren am Ausgang und am Eingang verbaut, die die Spannung glätten und den Regler stabilisieren sollen.

Bei Audioanwendungen ist die Masseführung ein wichtiges Thema, hier setzten wir auf einen zentralen Massepunkt um Querströme und somit entstehende Störungen zu vermeiden, der zentrale Massepunkt (GND Tie) wurde direkt unter dem Codec platziert.

4.1.6. Das WIFI Modul

Wohl einer der aufwändigsten Punkte bei der Auswahl der Hardware war das W-Lan Modul, doch genau dieses war bei der Anbindung an das System eines der einfachsten, da es bereits alle nötigen Schaltungen (wie die Core Voltage Generation und die Ausgangsstufen mit der Antennenbeschaltung integriert hatte). Das XG-182M von Zcomax musste lediglich mit den sechs, für den SDIO Bus üblichen Leitungen mit dem Prozessor verbunden werden.

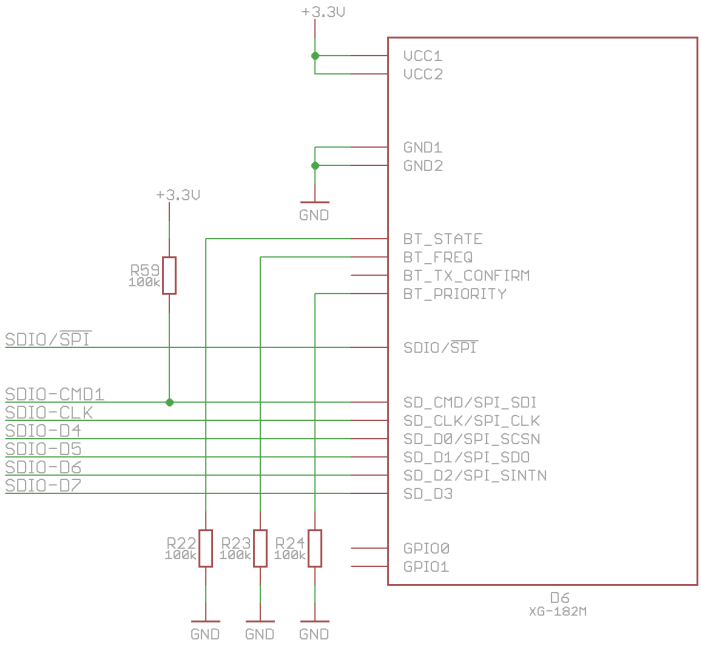


Abbildung 4.1.6-1 W-Lan Modul

Das Modul erwartete nur eine Spannungsversorgung, die mit 3.3V genau unserer Systemspannung entspricht (alle anderen Spannungen werden intern generiert).

Da wir das Modul vom Hersteller ohne weitere Dokumentation erhalten haben, blieb uns nur das mit 11 Seiten nur wenig Information bietende Datenblatt von der Homepage von Zcomax. Hier gab es vor allem Unklarheit über die Verwendung der Pins für den Bluetooth Coexistence Support. Diese Funktion sollte es ermöglichen sowohl Bluetooth als auch W-Lan quasi-parallel zu betreiben, da unser Bluetooth Modul diese Funktion jedoch nicht unterstütze wurden alle als Eingang deklariere Pins auf einen definierten Low Pegel gezogen während die Ausgangspins offen gelassen wurden.

Wichtig bei der Beschaltung ist, wie sich nach Revision 1 der Hardware herausstellte, der Pullup Widerstand auf der CMD Leitung, da sonst da sonst das W-Lan Modul beim start in einem undefinierten Zustand ist und somit nicht mehr ansprechbar.

4.1.7. Der SD Kartenslot

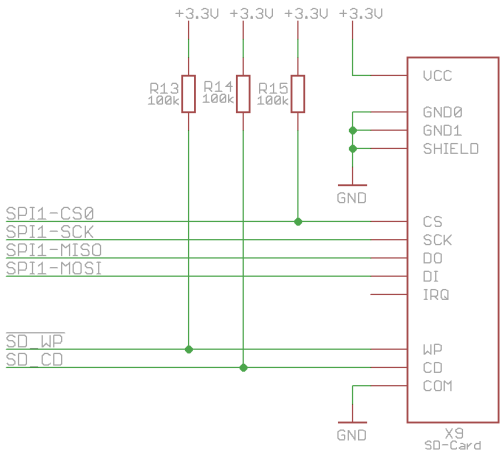


Abbildung 4.1.7-1 SD Kartenslot

Ein weiterer gänzlich unproblematischer Teil der Projekthardware, war der zweite (eigentlich 3.) SD Karten Slot. Da der Prozessor insgesamt nur zwei MCI Interfaces bereitstellte, d.h. eigentlich nur zwei SD Karten oder SDIO Geräte an den Prozessor

angeschlossen werden können, mussten wir uns eine andere Methode überlegen, den SD – Kartenslot an das System anzubinden. Hier kam uns unsere Erfahrung mit der xRemote (von der bereits ein erster Testaufbau funktionierte) zu gute, denn dadurch wussten wir, dass alle SD und MMC Karten auch über ein, von den Spezifikationen her, relativ offen liegendes SPI Interface angesprochen werden können. Das SPI Interface ist zwar langsamer wie das SD Interface, da es jeweils nur 1Bit gleichzeitig übertragen werden kann (während es im SD Modus 4Bit sind), jedoch für unsere Zwecke leicht ausreichend.

4.1.8. Der serielle Flash

Da wir zunächst davon ausgingen, das Betriebssystem auf den internen Flash des Gerätes zu speichern und uns bewusst war, dass die 16MB interner Speicher nicht reichen würden, sahen wir Footprints für weitere 32MB Flash vor. Da wir uns jedoch dazu entschlossen, das Betriebsystem von der SD-Karte zu starten (und nur den Kern in den Flash zu kopieren), wurde der verwendete serielle NOR Flash der Firma Spansion nie bestückt geschweige denn getestet.

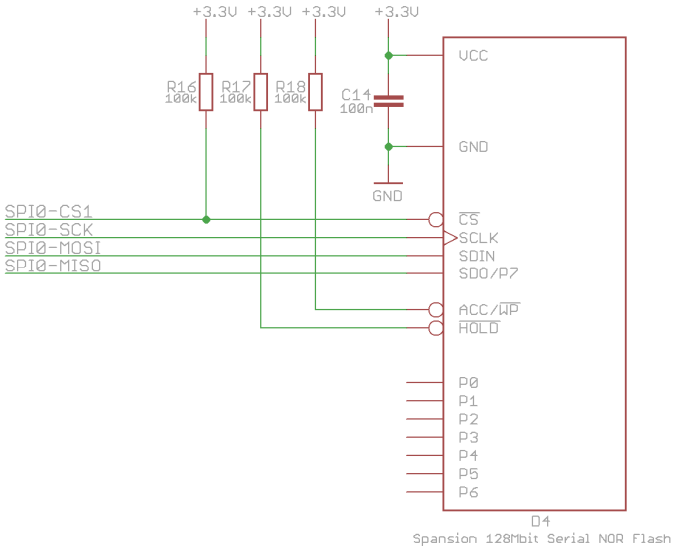


Abbildung 4.1.8-1 NOR Flash

Insgesamt wurden zwei Footprints vorgesehen, die an das SPI0 Interface angebunden waren, und deren Anbindung sich jeweils nur durch die Verwendung einer anderen Chip Select Leitung unterschied.

4.1.9. Der Boardcontroller

Für Zeit kritische Operationen, zur Entlastung des Hauptprozessors und zur Vereinfachung der Treiberentwicklung implementieren wir einen leistungsstarken Board Power Management und User Interface Management Controller. Unsere Wahl viel aus Gründen der Performance und der Einfachheit bzw. auch aus Kostengründen wiederum auf den in der xRemote verwendeten ATmega1281.

Der Takt wurde wie üblich über einen Quarz generiert, dieser wurde mit den bei 3.3V maximalen 8MHz getaktet (obwohl der Prozessor durch Übertakten 12MHz bei 3.3V ohne Probleme und Einschränkungen erreicht – wie bei der xRemote).

Alle Analog- und Digitalspannungen des Prozessors wurden mit den im System vorhanden 3.3V versorgt und der Reset wurde zum Debuggen herausgeführt und mit einem 100k Pullup auf einen definierten Zustand geführt (obwohl das nicht zwingend nötig ist, da dieser Pin intern mit einem Pullup versehen ist)

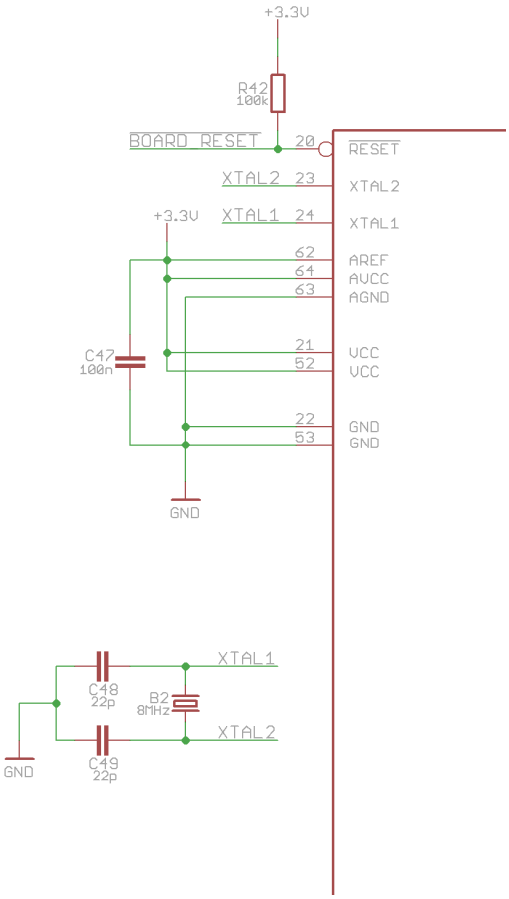


Abbildung 4.1.9-1 Boardcontroller

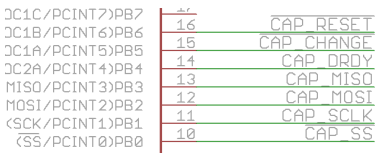


Abbildung 4.1.9-2 SPI

Der Gesamte Port B des Boardcontrollers wird's zur Anbindung des QT1106 – des Controllers der kapazitiven Sensors – verwendet. Grundsätzlich erfolgt die Übertragung über den SPI Bus, jedoch erfordert der QT1106 aus Timinggründen auch noch eine DRDY Leitung die das Timing für die SPI Abfragen vorgibt. Mittels der CHANGE Leitung, gibt der Controller ein Signal aus, sobald sich der Wert eines der kapazitiven Sensoren geändert hat.
Diese Leitung wird nun auf einen PCINT des Prozessors angeschlossen um effizient und ohne Polling auf Änderungen der Sensorwerte reagieren zu können.

Zur Anbindung des Boardcontrollers an den Hauptprozessor wird die USART1 des ATmega1281 verwendet, welche mit einer Baudrate von 38400 Baud Daten mit dem Boardcontroller austauscht. Zur Steuerung der Hintergrundbeleuchtung des kapazitiven Panels (welche über IO-Expander gepulst angesprochen wird) wird das I²C Interface des Prozessors verwendet, welches mit den verwendeten 2k2 Widerständen mit ausreichend niedrigen Pullups bestückt ist, um Frequenzen von bis zu 1 MHz Bustakt erreichen zu können. (Je kleiner der Pullup, desto schneller kann mit dem Bus gearbeitet werden, desto höher sind jedoch auch die Ströme und die Lasten auf den

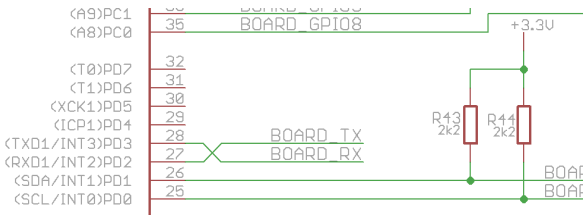


Abbildung 4.1.9-3 UART, I²C

Ausgängen der IC's – die verwendeten Werte sind Erfahrungswerte aus zuvor durchgeführten Projektarbeiten).

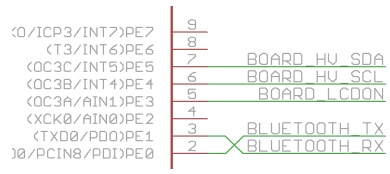


Abbildung 4.1.9-4 S-I²C, Bluetooth

Der Port E des Prozessors wird für die Steuerung des LCD bzw. der Hintergrundbeleuchtung des LCD verwendet. Die Einstellungen der Hintergrundbeleuchtung können über ein I²C Interface getätigt werden, da das Interface des DC-DC Wandlers jedoch nur sehr geringe Taktfrequenzen (bis 100kHz) unterstützt, haben wir uns entschlossen den DC-DC Wandler nicht an den Hardware I²C anzuschließen, da wir diesen sonst von maximalen 1,7 MHz auf 100kHz heruntertakten müssten, und dadurch ein Pulsen der RGB LEDs am kapazitiven Panel nicht mehr möglich ist. Da das Protokoll des I²C relativ einfach ist, beschlossen wir uns den Bus per Software zu implementieren.
Als zweite Peripherie am Port E ist das Bluetooth Modul mittels serieller Verbindung auf die USART0 des Prozessors angebunden.

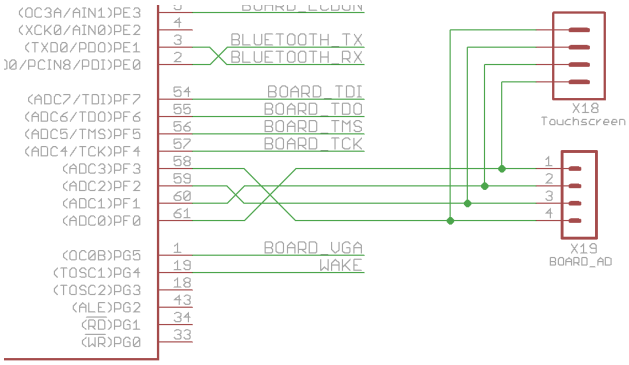


Abbildung 4.1.9-5 JTAG, Touchscreen, Wake, VGA

Der Port F war mit den JTAG – Debug Anschlüssen TDI, TDO, TMS und TCK belegt, welche auf den Debug Sockel herausgeführt wurden. Die zweite Hälfte des Port F, die einige ADC Kanäle beinhaltet, wurde zum Anschluss des optionalen Touchscreens verwendet (der nicht bestückt wurde, da er die Qualität des Bildes des Monitors verschlechtern würde) bzw. für mögliche Erweiterungen auf eine Stiftleiste geführt. Das Auskreuzen der Leitungen hat nur in diesem Fall nur den sinn, dass das Layouten dadurch deutlich erleichtert wurde.
Die Wake Leitung die mit Port G verbunden ist dient zum Aufwecken des Prozessors, wenn dieser im Schlafmodus ist, während das „BOARD_VGA" Signal zum an und Abschalten des VGA Ausgangs dient.

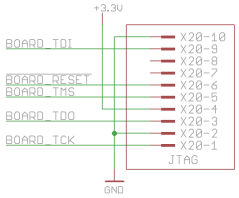


Abbildung 4.1.9-6 JTAG

Der JTAG Anschluss ermöglicht das in Circuit Debuggen und Programmieren des Boardcontrollers. Die Pinbelegung des Wannensteckers ist genormt, um zu jedem Programmiergerät von Atmel kompatibel zu sein.

4.1.10. Das Bluetooth Modul

A

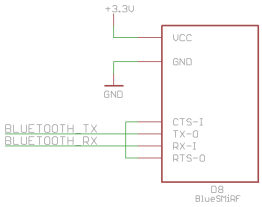


Abbildung 4.1.10-1 Bluetooth

Als Bluetooth Modul wurde wie in der xRemote das Sparkfun BlueSMiRF Modul verwendet. Es ist ein Modul, das eine serielle Schnittstelle emuliert. Es wird mit 38400 Baud mit Daten beschickt. Um die die Hardwareflusssteuerung des Moduls zu deaktivieren, werden RTS und CTS kurzgeschlossen.

4.1.11. Der kapazitive Sensorcontroller

Zur Realisierung der kapazitiven Tasten und des kapazitiven Wheels verwendeten wir den QT1106 von Quantum bzw. von Atmel (Quantum wurde von Atmel aufgekauft). Der QT1106 ist einfach über ein SPI Interface mit einigen Zusatzleitungen ansteuerbar.

Das Standard SPI Interface bestehend aus MISO, MOSI, SCLK und SS wurde beim QT1106 um die Statusleitungen CHANGE und DRDY erweitert. Die DRDY ist für die Kommunikation mit dem QT1106 unbedingt notwendig, den nur wenn sie indiziert, dass die Daten am Chip zur Abholung bereit sind, können diese in einem bestimmten Zeitfenster abgeholt werden. Um den QT1106 nicht andauernd abfragen zu müssen hat er den CHANGE Ausgang, der LOW wird, sobald sich der Status einer der Tasten bzw. des Wheels geändert hat. Um diese zwei Leitungen auf einem bestimmten Zustand zu halten, werden Pulldown Widerstände verwendet.

Die Beschaltung des OSC und des SPREAD Pins ist dem Datenblatt des QT1106 zu entnehmen und sind von der Versorgungsspannung abhängig und dienen als externe Beschaltung für den internen Oszillator (Spread Spectrum Technik für erhöhte Störungssicherheit).

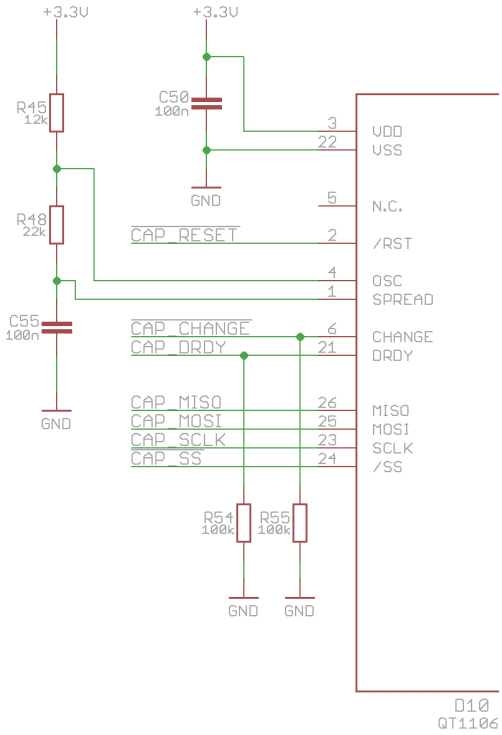


Abbildung 4.1.11-1 QT1106

Die Beschaltung des analogen Teil des QT1106, war hier etwas schwieriger, da es zwar ungefähre Angaben für die Kondensator bzw. Widerstandswerte im Datenblatt gibt, die genauen Werte jedoch stark von dem Aufbau der Sensorflächen, von den Längen der Leitungen und von der Dicke des Gehäuses abhängt.

Da die endgültigen Werte erst nach dem Einbau in das Gehäuse durch Versuche bestimmt werden können, sind im Stromlaufplan Werte die zwar funktionieren sollten eingetragen, aber nicht unbedingt Werte die die optimale Genauigkeit bzw. die optimale Geschwindigkeit erbringen. Generell ist zu sagen, dass die 10k Widerstände bis auf weiteres auf dem Wert belassen werden können und nur die Kondensatorwerte angepasst werden müssen, so bringen höhere Kondensatorwerte eine weit höhere Empfindlichkeit, jedoch auch eine um viele trägere Erfassung von Berührungen.

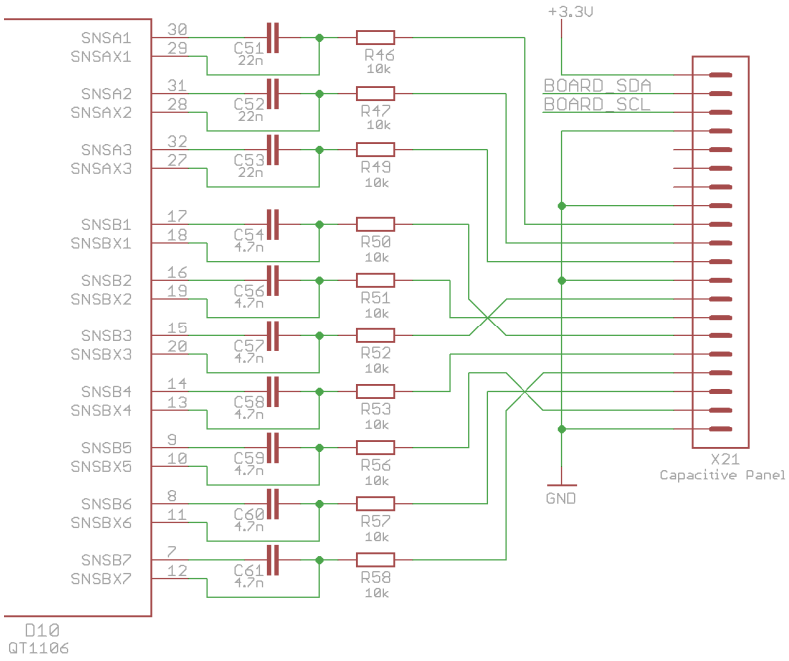


Abbildung 4.1.11-2 QT1106

Nach Revision 1 der Hardware, wurde die zuvor verwendete Stiftleite im 2.54er Raster, aufgrund einiger Probleme mit den Anschlussleitungen (zu unflexibel und zu fehleranfällig) gegen einen 20pinnigen FPC mit 1mm Pinabstand ausgetauscht. Der nun verwendete FPC ermöglicht eine flexible Verbindung des Basisboards mit dem kapazitiven Panel, außerdem ist die Verbindung relativ fehlerunanfällig. Den einzige Nachteil den diese Verbindungsmethode hat, ist eine aufwändigere Implementation und deutlich höhere Hardwarekosten, gegenüber einem Flachbandkabel.

Neben den Leitungen für die Steuerung der kapazitiven Tasten bzw. des kapazitiven Wheels wurden auf noch die High Speed I²C Leitungen zum Ansteuern der Hintergrundbeleuchtung in den FPC eingebunden.

4.1.12. Das kapazitive Panel

Das kapazitive Panel, eine separate Hardwareentwicklung, bildet des User Interface des Players. Es ist mit einem 20poligen FPC an den Player angebunden besteht eigentlich nur aus den Sensorflächen für die kapazitiven Sensoren sowie aus der Ansteuerung für die Hintergrundbeleuchtung.

Insgesamt werden am kapazitiven Panel nur vier der insgesamt 7 Sensorleitungen verwendet, an die kapazitive Sensoren angeschlossen werden können, die anderen Leitungen werden offen gelassen und die entsprechend Kondensatoren auf der Hauptplatine nicht bestückt.

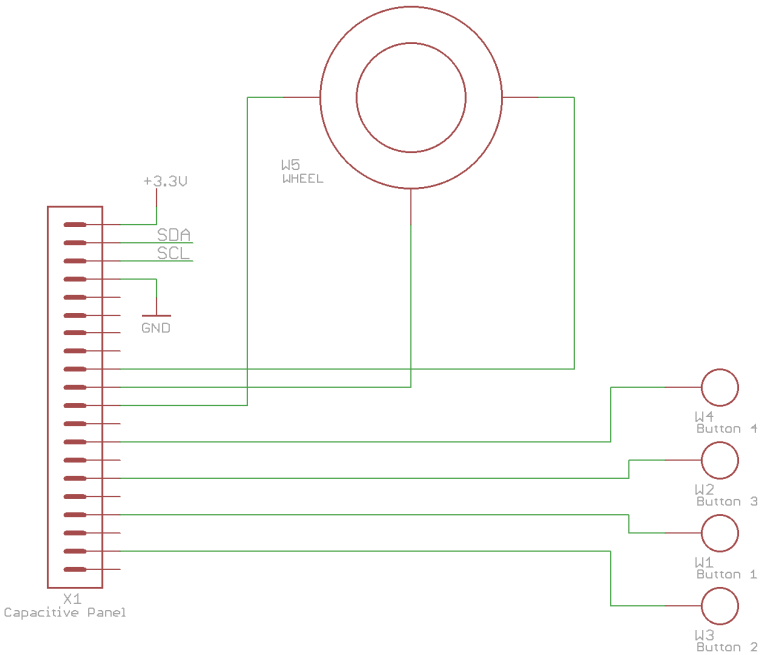


Abbildung 4.1.12-1 Kapazitives Panel

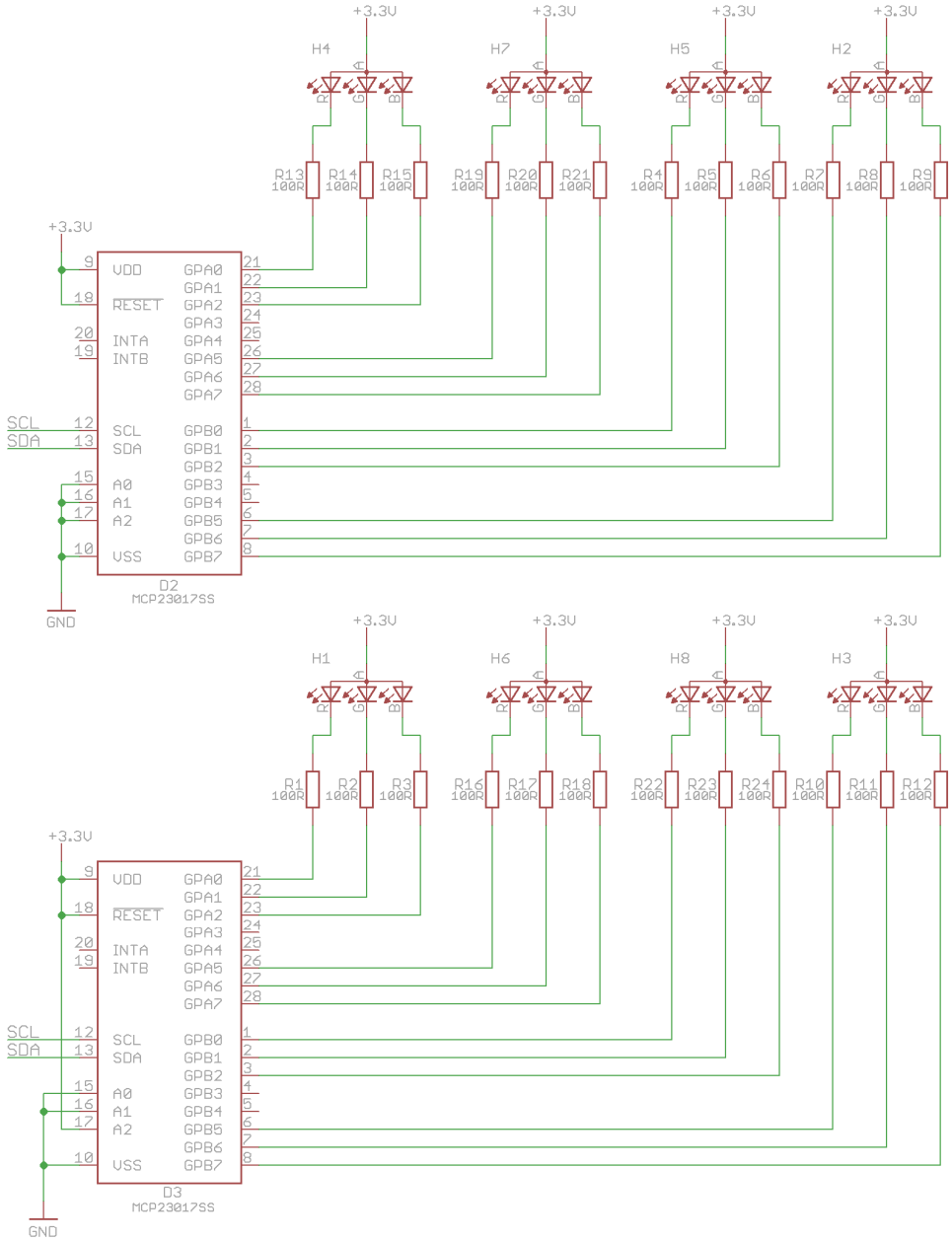


Abbildung 4.1.12-2 Hintergrundbeleuchtung

Die Hintergrundbeleuchtung wurde über insgesamt 8 RGB LEDs realisiert. Da das Verbinden aller Leitungen, mit dem Hauptboard über einen Bus nicht möglich ist (aus Platzgründen) wurden die RGB LEDs über zwei High Speed IO-Expander der Firma Microchip angesteuert. Die MCP23017 ermöglichen I²C Frequenzen von bis zu 1,7MHz, was es ermöglicht die LEDs über eine Pulsbreitenmodulation anzusteuern, um die Farbe bzw. die Helligkeit der einzelfarben zu ändern.

4.2. xRemote

Da bereits einige Erfahrungen bei Designs von 8 Bit Systemen bestanden, war der Entwurf des Stromlaufplans hier denkbar einfach. Das Layout hingegen gestallte sich, aufgrund des kleinen Formfaktors, doch recht fordernd. Bei der xRemote galt vor allem das Power Management als Herausforderung, welche jedoch mit soliden Überlegungen und guter Durchführung gemeistert wurde und anstandslos funktionierte.

Aufgrund einiger löt- und layouttechnischer Probleme musst auch von diesem Layout eine zweite Version gefertigt werden.

4.2.1. Der Prozessor

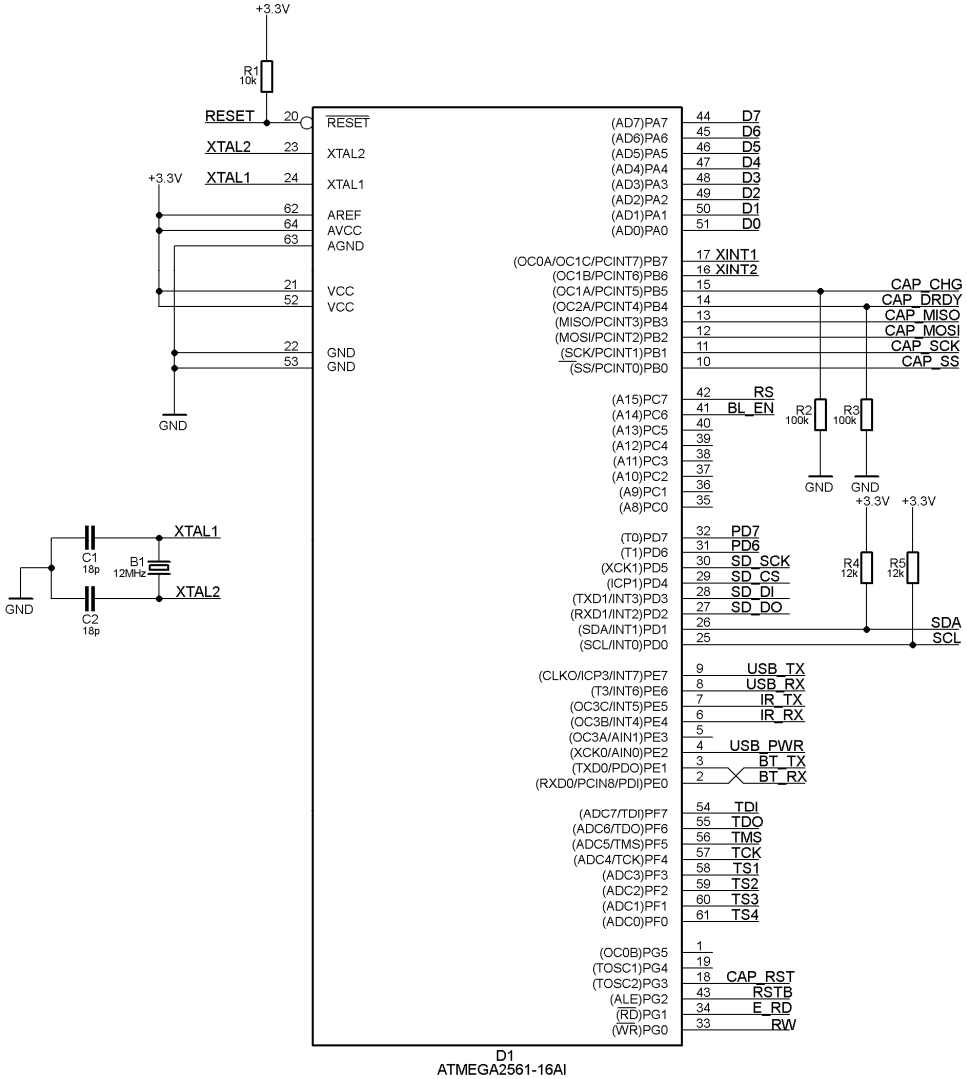


Abbildung 4.2.1-1: Prozessor

Als Prozessor für die Fernsteuerung xRemote wird ein ATMEGA 1281 verwendet, da dieser die Voraussetzungen wie I2C und SPI unterstützt. Weiters ist es auch möglich zwei serielle Schnittstellen zu realisieren. Eine zum Lesen von einer SD- Karte und die andere um das Schreiben über Bluetooth zu ermöglichen.
Der SPI- Bus wird zum Ansteuern der Kapazitiven Sensoren verwendet (siehe Seite xx) und zum Ansteuern des Neigungssensors wird ein I2C- Bus eingesetzt (siehe Seite xx).
In Abbildung 1 sind auch die Grundbeschaltungen für den Quarz und das JTAG-Interface, welche aus den jeweiligen Datenblättern entnommen wurden, zu erkennen.

4.2.2. Die Ladeschaltung

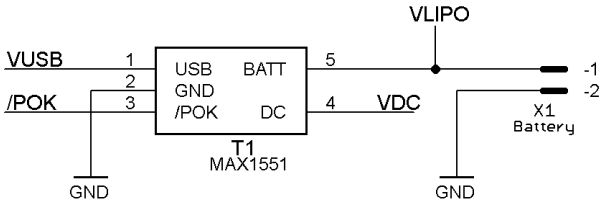


Abbildung 4.2.2-1: Akku- Ladeschaltung

In Abbildung 2 ist die Akku- Ladeschaltung abgebildet. Zum Laden des Lithium- Polymer-Akkus wird der Lade- IC MAX1551 verwendet.
Der Akku kann über VDC mit 500mA oder über USB mit 100mA geladen werden. Wird USB für den Ladevorgang verwendet, so kann der Akku geladen und das System zur gleichen Zeit versorgt werden. Falls USB und VDC gleichzeitig angeschlossen sind, dann erfolgt das Laden automatisch über USB.
Ist mindestens eine der zwei Möglichkeiten angeschlossen, so wird PowerOK ausgegeben. Diese Ausgabe ist das Zeichen, dass der Akku geladen wird.

4.2.3. Die Spannungsversorgung

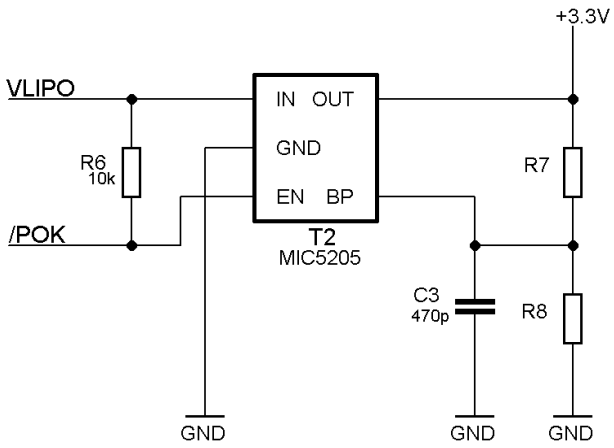


Abbildung 4.2.3-1: Festspannungsregler

Mit dem Festspannungsregler MIC5202 werden die 3,3V, die für die Versorgung des Systems verwendet wird, erzeugt. Die Widerstände R7 und R8 werden nicht bestückt, da diese nur die Möglichkeit bieten sollen den 3,3V Regulator gegen einen einstellbaren Regulator zu ersetzen und mit diesen Widerständen die Spannung einzustellen. Dieser Regulator wird nur eingeschaltet, wenn keine externe Versorgung angeschlossen ist.

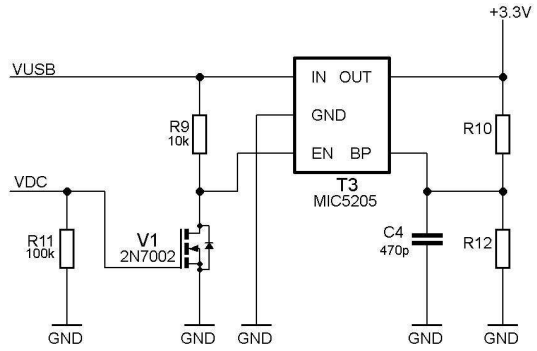


Abbildung 4.2.3-2: Festspannungsregler

Auch der MIC5205 wird zur Versorgung des Systems verwendet, jedoch nur wenn eine externe Spannungsquelle angeschlossen ist.

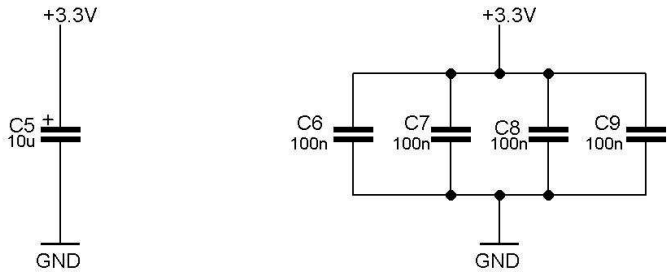


Abbildung 4.2.3-3: Stabilisations- und Glättungskondensatoren

Beim Kondensator C5 handelt es sich um einen Glättungs- bzw. Stabilisationskondensator und die weiteren Kondensatoren C6 – C9 sind Abblockkondensatoren. Die Abblockkondensatoren dienen zum Schutz vor Spannungsspitzen. Der Glättungskondensator ist zum Ausgleichen von auftretenden Versorgungsspannungsschwankungen.

4.2.4. Der Erweiterungs- & Debuganschluss

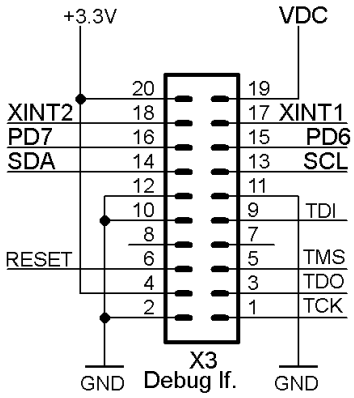


Abbildung 4.2.4-1: Debug Connector

Abbildung 6 zeigt einen Systemsanschluss mit externen Interrupt- Pins, I^2C- Bus Anschlüssen und „gerneral purpose“ IO- Pins sowie mit einem vollwertigen JTAG-Anschluss.

4.2.5. Der USB Anschluss

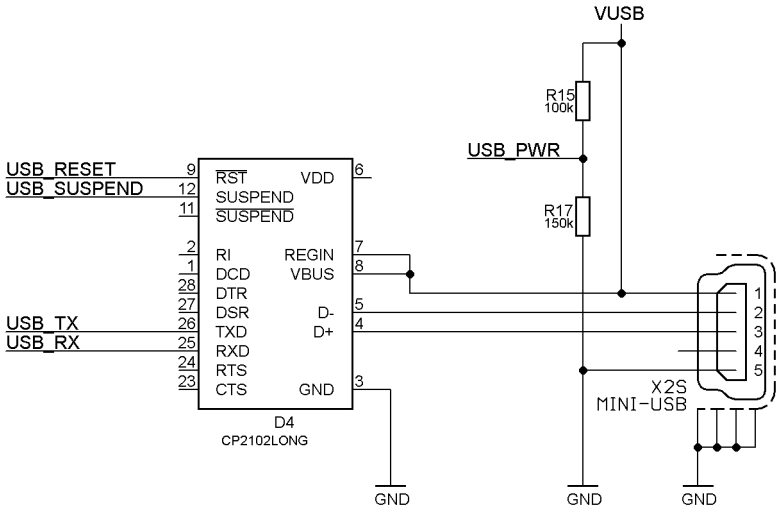


Abbildung 4.2.5-1: USB- Seriell Wandler

Der IC CP2102 ist ein USB- Seriell Wandler. Die Grundbeschaltung wurde aus dem Datenblatt des Herstellers übernommen. Der USB_RESET- Pin ermöglicht das Rücksetzen des Prozessors und der Pin USB_SUSPEND gibt ein HIGH- Signal zurück, wenn die USB Verbindung im Schlafmodus ist.
Der Spannungsteiler auf USB_PWR ermöglicht das Ermitteln ob ein Gerät angeschlossen ist oder nicht.

4.2.6. Der Beschleunigungssensor

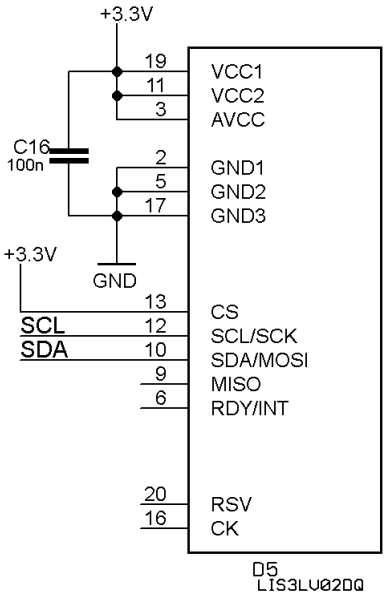


Abbildung 4.2.6-1: Beschleunigungssensor LIS3LV02DQ

Der Kondensator C16 wird als Abblockkondensator gegen Spannungsspitzen verwendet. Der Beschleunigungssensor wurde mit der I^2C- Interface verwendet. Da das Layouten der I^2C- Pins einfacher war, wurde der Chip, anstatt mit SPI, mittels I^2C angebunden.

4.2.7. Der LCD mit Touchscreen

Abbildung 9 zeigt die Steckerleiste zum Anschluss des Display-Panels mit Touchscreen.

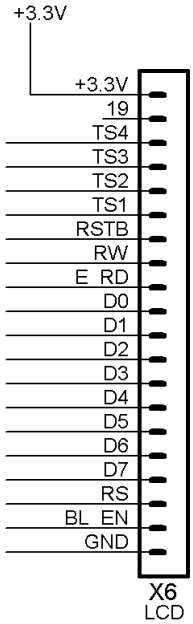


Abbildung 4.2.7-1: Display

4.2.8. Der microSD Slot

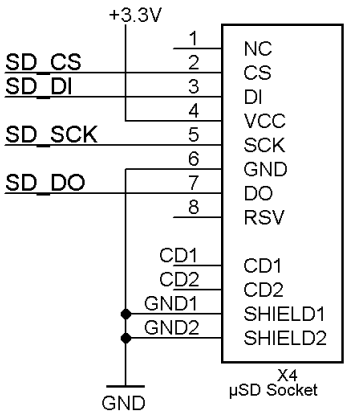


Abbildung 4.2.8-1: micro SD- Card Slot

Zum Erweitern des Speichers wurde ein micro SD- Card Slot implementiert. SD- Karten sind über den SPI- Bus ansprechbar, darum wurden die SPI- Leitungen des SPI-Interface angeschlossen.
SPI- Leitungen: Chip Select (SD_CS), Data In (SD_DI), Serial Clock (SD_SCK) und Data Out (SD_DO)

4.2.9. Das Bluetooth Modul

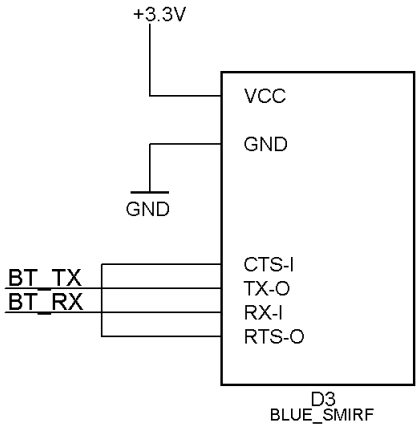


Abbildung 4.2.9-1: Blue- SMIRF

Das serielle Bluetooth- Modul Blue- SMIRF emuliert eine serielle Schnittstelle. Angesteuert wird es über BT_RX (read) und BT_TX (transmit).
CTS-I und RTS-O sind kurzgeschlossen, das heißt die Flusssteuerung wird deaktiviert.

4.2.10. Der kapazitive Sensor

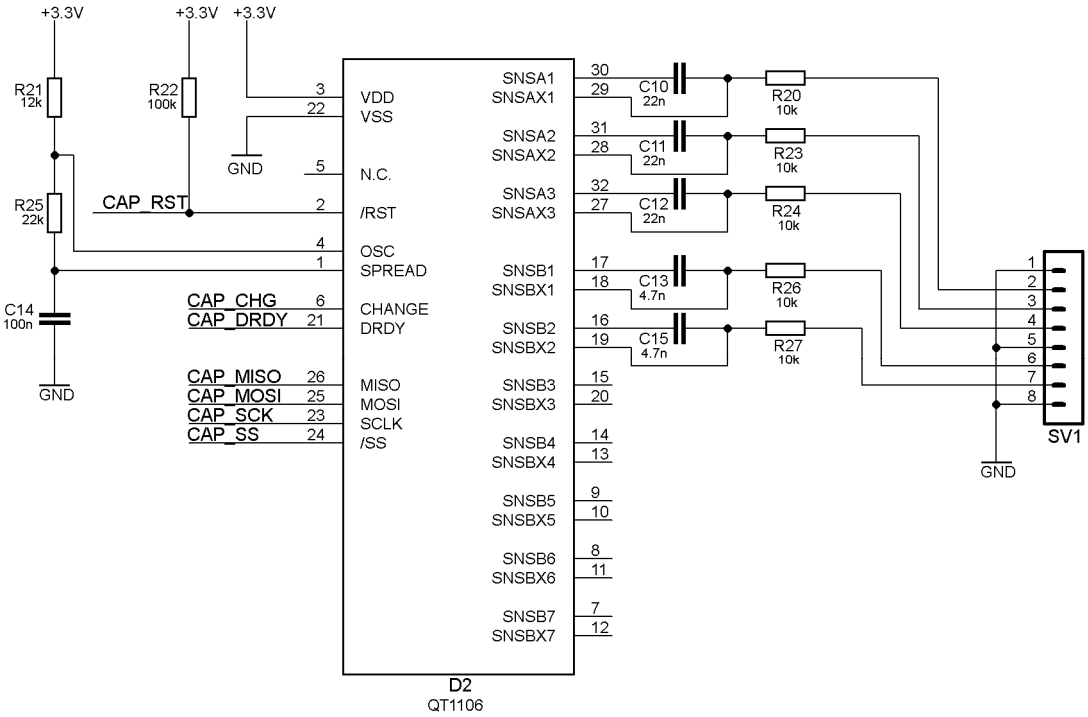


Abbildung 4.2.10-1: Kapazitiver Sensor

Bei dem IC QT1106 handelt es sich um einen kapazitiven Sensor. Die Standardbeschaltung wurde aus dem Datenblatt entnommen.
Die Werte der Kondensatoren C10 – C15 können nicht berechnet werden, sie sind durch Versuche zu bestimmen. Sie müssen durch Versuche bestimmt werden wegen der Erkennung des Fingerdrucks und der Dicke des Gehäuses.
Die Ansteuerung des kapazitiven Sensors verläuft über das SPI- Interface mit einem zusätzlichen Change- Pin. Der Change- Pin wird verwendet, damit der Prozessor weis,

ob sich etwas geändert hat (zum Beispiel: Tastendruck). Mit dem zusätzlichen Pin muss nicht dauernd nach Veränderungen nachgefragt werden.

4.2.11. Das Infrarot Interface

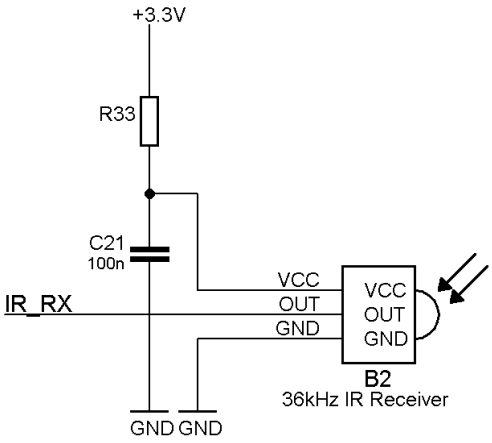


Abbildung 4.2.11-1: Infrarot Empfänger

Der Infrarot Empfänger arbeitet mit einer Frequenz von 36kHz und funktioniert im Grunde genommen wie ein Feldeffekttransistor (FET). Wenn Infrarotlicht auftrifft schaltet der Empfänger durch und zieht den Ausgang OUT gegen Masse. Der 100n Kondensator C21 dient wiederum zum Stabilisieren und Blocken (siehe Datenblatt).

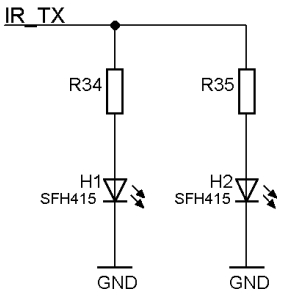


Abbildung 4.2.11-2: Infrarot Senderdioden

In Abbildung 14 sind die 950nm Infrarot- Transmitter- Dioden zu sehen. Es werden zwei Dioden platziert, eine vorne und die zweite seitlich. Zwei Stück werden verwendet, da man die Fernsteuerung sowohl aufrecht und auch um 90° gedreht (liegend) verwenden kann.
Die Vorwiderstände für die Dioden sind je nach gewünschter Intensität der Infrarot Senderdioden einzulöten. Der maximale Diodenstrom beträgt 100mA.

4.3. xDimension

Die Schaltung von xDimension beschränkte sich im Großen und Ganzen auf ein leistungsstarkes Infrarotpanel und die Spezialkamera und einer Ladeschaltung für die xRemote.

4.3.1. Das Infrarotpanel

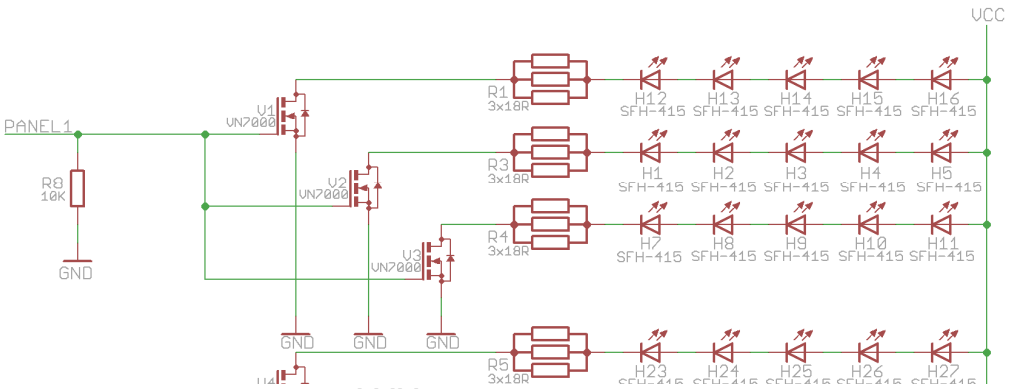


Abbildung 4.3.1-1 IR Array

Das Infrarotpanel besteht aus insgesamt 60 SFH-415 Infrarot Transmitterdioden mit einem Öffnungswinkel von 68° und die Leistung beträ gt 40mW/Sr. Da das Board direkt mit 12V versorgt wird, können bei einer Flussspannung von 2.3V pro Diode insgesamt 5 Dioden versorgt werden. Da ein Vorwiderstand unbedingt benötigt wird, jedoch die Leistung eines normalen 0805er oder 1206er SMD Widerstand bei weitem überstiegen wird, werden drei SMD Widerstände parallel geschaltet um die Leistung moderat zu halten. Als Schalttransistoren werden insgesamt 12 2N7002 verwendet, die mit einem maximalen Dauerstrom von 115mA optimal für unsere Anwendung geeignet sind (da der Diodenstrom auf ungefähr 90mA eingestellt ist). Es werden immer drei Fünfergruppen zu einem Array, das komplette Panel besteht aus vier Arrays. Jedes Array kann von der xRemote nun mit einer Pulsbreite in der Helligkeit gesteuert werden. Der Stromverbrauch des kompletten Panels wurde mit ungefähr 1A berechnet, durch intelligentes An und Abschalten der einzelnen Panels – je nach Entfernung des Benutzers – kann der Stromverbrauch in Normalbetrieb auf rund 500mA reduziert werden.

4.3.2. Das xRemote Interface mit Ladeschaltung

In der xRemote wird ein spezieller Anschluss zum Laden, Debuggen und für Erweiterungen verwendet.

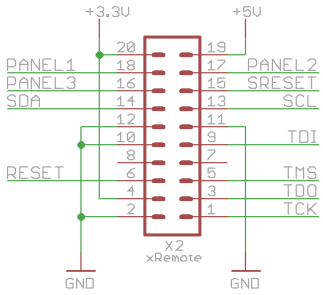


Abbildung 4.3.2-1 Debug Port

Dieser Anschluss ist im Grunde genommen im 2,54er Raster gehalten, jedoch wurde er im Layout auseinandergezogen, so dass die Pin 9 und 10 von den Pins 11 und 12 weiter entfernt sind, was es ermöglicht jedes Standard JTAG Programmiergerät an die xRemote anzuschließen. Das Gegenstück zu diesem Anschluss wird nun in xDimension verbaut, womit es möglich ist die xRemote auf xDimension zu stecken. Die I²C Pins und ein GPIO Pin werden für die IR Kamera verwendet, während die drei noch freien GPIO Pins zum Ein und Ausschalten der IR Arrays verwendet. Das IR Array wird ohne den Einsatz jeglicher

Diode an die 12V Versorgungsspannung angeschlossen, um die Verluste zu minimieren. Im Layout wurden die Leitungen außerdem so dick wie möglich gezogen, um den Maximalstrom von 1A ohne Erwärmung bereitstellen zu können.

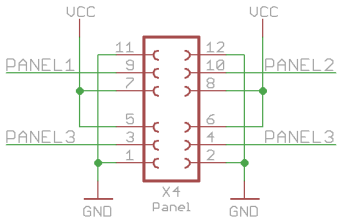
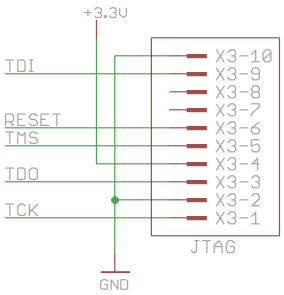


Abbildung 4.3.2-2 IR Panel



Um die xRemote debuggen zu können wurden die JTAG Leitungen der xRemote auf einen neuen separaten JTAG Anschluss herausgeführt.

Abbildung 4.3.2-3 JTAG

Die xRemote benötigt zum Laden insgesamt zwei Spannungen 5V für den Ladecontroller und 3.3V um das System zu versorgen (der interne Spannungswandler ist bei aktiver Ladeschaltung abgeschaltet).

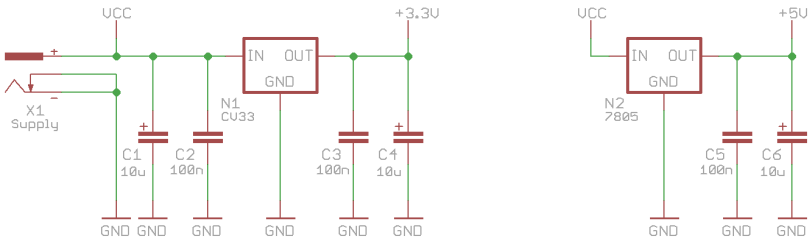


Abbildung 4.3.2-4 Spannungsversorgung

Um den Schaltungsaufwand gering zu halten wurden einfach 3 Terminal Spannungsregler in 5V und 3.3V Ausführung verwendet, die mit den entsprechenden Kondensatoren ausgestattet wurden. Dies Garantier zwar keine optimale Effizienz, vereinfach jedoch das Hardwaredesign und minimiert die Kosten (da der Kostenfaktor und der Zeitfaktor vor allem im letzten Projektstadium eine große Rolle spielten)

4.3.3. Die IR-Kamera

Um die Position der markierten Finger zu erfassen wird die Infrarot Kamera der Wiimote (der Fernsteuerung der Nintendo Wii) verwendet. Diese Kamera ist eine in ihrer Leistung und Integrationsdichte unübertroffen und außer in der Wiimote nicht auf dem Markt zu erhalten.

Die Infrarotkamera ist denkbar einfach über den I²C Bus anzusteuern und benötigt extern nur einen 25MHz Quarzoszillator und 3.3V Versorgungsspannung.
Außerdem wurde die Reset Leitung der Kamera noch mit der xRemote verbunden, um die Kamera – sollte es zu Instabilitäten zu kommen – auf den Ausgangszustand zurückzusetzen.

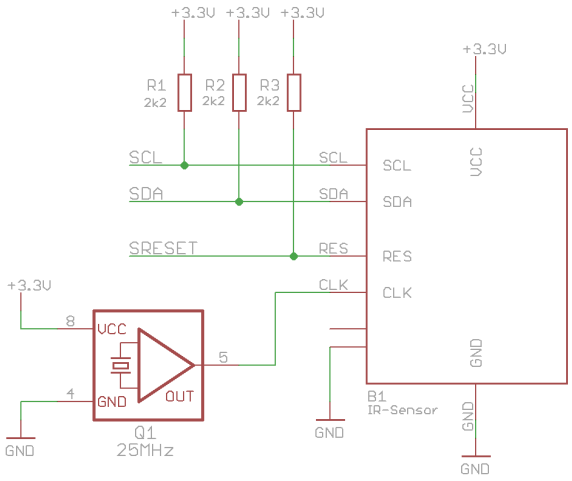


Abbildung 4.3.3-1 Spannungsversorgung

4.4. Das Layout

Da die Erläuterung des ganzen Layouts den Rahmen dieser Arbeit deutlich sprengen würde, möchten wir nur einige Guidelines für das Routing eines solchen Systems nennen.

- Bus Routing – Mehrere Leitungen mit ähnlichem Ziel zu einem Bus zusammenzufassen und für diesen Bus dann das optimale Routing zu finden. (Entgegen der Strategie auf einem Layer nur horizontal und auf anderen Layer nur vertikal zu routen)
    - Vereinfachung des Layouts
    - Vermeidung von Laufzeitdifferenzen vor allem bei Hochgeschwindigkeitsbussen
- Intelligentes Plazieren der Bauteile – Nicht nur um das Layouten zu vereinfachen, sondern auch zur Reduzierung von Störungen benachbarter Bauteile (kapazitive Sensoren und Audioteile so weit entfernt von Störungsquellen wie Schaltwandlern)
    - Vereinfachung des Layouts
    - Geringere Störungen
- Immer nur auf einem Layer routen – Busse und Hardwaregruppen immer versuchen auf nur einem Layer zu routen
    - Vereinfachung des Layouts
    - Möglichkeit Verbindungsleitungen über den zweiten Layer zu führen
- Durchkontaktierungen reduzieren – Versuchen möglichst keine Durchkontaktierungen auf Signalleitungen zu benutzen
    - Weniger Fehlerquellen
    - Verbesserung der Signalqualität bei Hochgeschwindigkeitsbussen
- Versorungsleitungen ausrouten – Alle Versorungsleitungen (d.h. die Massen und die Systemspannungen) vor dem Überlagern des Polygons in richtiger und erforderlicher Leiterbahnbreite routen
    - Keine Probleme beim Leiterplattenhersteller
    - Keine Masse Problem (durch teil – abgeschnürte Masseflächen)
- Masseflächen groß und stabil halten – Versuchen eine große Massefläche über das komplette Board zu ziehen und so oft wie möglich (alle 5 bis 10mm) Bottom und Top Layer durchkontaktieren
    - Stabilisierung der Massefläche
    - Reduzierung von Schwingungen
    - Bessere EMV Stabilität
    - Geringeres Nebensprechen bei Bussen
- Masseführung bei Audio Layout beachten – Zentralen Massepunkt setzen und trennen von analoger und digitaler Masse
    - Störungsfreie Audiosignale
    - Keine Beeinflussung zwischen Ein- und Ausgängen
- 45° Leitungen, jedoch allgemein so wenig (scharf – 90°, 180°) Knicke wie möglich
    - Geringere EM Abstrahlung bei Hochgeschwindigkeitsbussen
- Leistungsführende Leitungen so kurz und so breit wie möglich halten
    - Geringere – durch den Leitungswiderstand verursachte – Verluste
    - Geringere Erwärmung
- Keine Durchkontaktierungen unter QFN Packages
    - Keine Kurzschlüsse durch unter dem Chip befindliche Masseflächen
- Leitungen gerade von TQFP, QFN und SSOP Packages sowie NAIS und ZIF Anschlüssen wegziehen
    - Vereinfacht den Lötvorgang und vermindert die Fehleranfälligkeit
- Testpunkte setzen (oder oft reichen Durchkontaktierungen) sowie Massepunkte zum Anklemmen
    - Vereinfachtes debuggen (Oszilloskop)

5. Die Firmware

Da wir sowohl bei der xRemote als auf beim Boardcontroller des Players auf den ATmega1281 setzten vereinfachte sich die Firmwareentwicklung weitestgehend, da viele Komponenten doppelt verwendet werden konnten.

Die Entwicklung der Firmware erfolgte im frei verfügbaren AVR Studio in der Version 4.14 (http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725) mit dem freien WinAVR Compiler in der Version 20070525 (http://winavr.sourceforge.net). Zum In Circuit Debuggen des Prozessors wurde der Atmel AVR Dragon (nur in frühen Projektphasen, wo für Testzwecke eine ATmega32 verwendet wurde) bzw. das JTAGIC mkII verwendet,

Zum Debuggen der Hardware wurde ein Saleae Logic Analyzer im Digitalbereich (erst in der späteren Projektphase) sowie ein Fluke 97 Scopemeter und die Tektronix Oszilloskope der HTL verwendet.

.

5.1. Der Boardcontroller

5.1.1. Übersicht

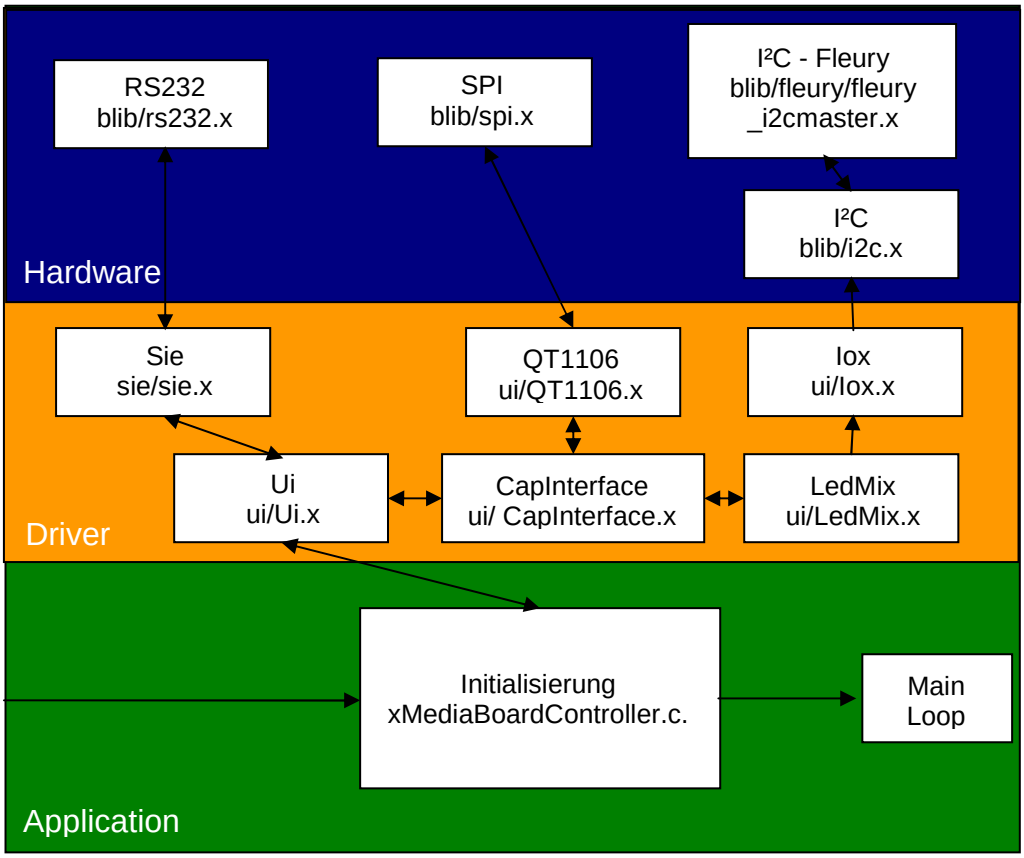


Abbildung 5.1.1-1 Firmware Grobübersicht

Der Boardcontroller hat die Aufgabe, einerseits, das kapazitive Panel, mit der entsprechenden Hintergrundbeleuchtung (RGB – LEDs, die über High Speed IO Extender gesteuert) anzusteuern , den Boost Konverter um den LCD zu steuern, das Power Management für einige Komponenten sowie den VGA Ausgang und die von der xRemote ankommenden Daten an die CPU weiterzuleiten.

Das Auslesen der Daten des kapazitiven Controllers gestaltet sich durch den von Atmel vorgefertigten Treiber relativ einfach und problemlos, das Pulsen der Hintergrund RGB LED über die IO-Extender, stellte hier bedeutend mehr Softwareaufwand da, da das Timing hier kritisch ist, und die ganzen Routinen durch die Anbindung der Extender über I²C relativ zeitaufwändig sind.

Das Weiterleiten der Daten von der xRemote zur AP7000 CPU ist hier ein bedeutend leichteres Unterfangen gewesen, da durch die bereits voll ausgereifte xSie das Commandrouting hier innerhalb weniger Minuten erledigt wurde.

5.1.2. Die xSIE

Als wichtigster Firmware Teil galt die xSie. Die xSie regelt jeden Austausch von Daten zwischen dem Boardcontroller und dem Hauptprozessor und dem Boardcontroller und der xRemote. Die xSie stellt einen Abstraktionslayer zwischen dem seriellen UART Treiber und der eigentlichen Applikation dar. xSie ist zum Packen der Daten in einem speziell für xMedia entworfenen Protokoll verantwortlich, dass es ermöglicht, eine sichere Übertragung zu gewährleisten und abgebrochene Datenübertragungen wieder aufzunehmen. Grundsätzlich sind die gesendeten Packete in SLIP (Serial Line Internet Protocol) Frames verpackt. Diese Packete bestehen nun aus mehren Komponenten, einer Packet ID, einem Commandcode, der Länge der Nutzdaten, und der eigentlichen Nutzdaten. Zur Fehlererkennung wird hier ein CRC (cyclic redundancy check) eingesetzt

Die xSie oder nur Sie implementiert die Protokoll Ebene eines von uns entwickelten Protokolls, dass es ermöglicht Daten zwischen Mikroprozessor Systemen bzw. Linux Computer schnell und unkompliziert auszutauschen.

Folgende Vorraussetzungen sollten erfüllt werden:
- Geeignet für serielle Schnittstellen ohne Handshake
- Paketorientiert
- Paket Identifizierung
- Geschwindigkeit
- Geringe Systembelastung
- (Sicherheit)

5.1.2.1. Paketform

Um unseren Anforderungen gerecht zu werden wählten wir folgendes Paketformat:

Control (16bit)	Command (8bit)	Length (8bit)	Data (variable)	CRC8 (8bit)

Control:

Das erste Feld, das „Control“-Feld, beinhaltet Felder zur eindeutigen Identifizierung eines übertragenen Paketes.

Answer (1bit)	Packet ID (15 bit)

Paket ID (15bit):

Die Paket ID ist eine Zahl, die jedes Paket eindeutig identifiziert. Die ID wird intern über einen Zähler generiert welcher iteriert wird, um sicher zu stellen, dass diese wirklich eindeutig ist. Mit einer Größe von 15bit können somit 32768 (2 ^ 15) Pakete übertragen werden, bevor eine ID wieder auftritt.

Answer:

Um eine Antwort auf ein Paket übertragen zu können, wurde das Answer Feld implementiert. Ist das Answer Feld gesetzt, wird angenommen, dass das übermittelte Paket eine Antwort auf ein Paket (mit der gleichen ID) ist.

Command:

Das Command Feld ermöglicht die Zuordnung der Übertragenen Daten zu einem bestimmten Befehl. Mit einer Größe von 8bit können maximal 256 Befehle verschiedene Befehle übertragen werden.

Length:

Das „Length“ Feld spezifiziert die Länge der übertragenen Nutzdaten, mit einer Größe von 8-bit können maximal 255 Byte Nutzdaten übertragen werden. Diese Limitation wurde in Kauf genommen, da nicht mehr Daten erwartet werden und die Performance dadurch gesteigert wird.

Data:

Das „Data“ Feld beinhaltet die eigentlichen Nutzdaten, die Länge dieses Paketes ist variabel und wird durch das „Length“ Feld festgelegt. Die übertragenen Nutzdaten haben keine definierte Form und hängen von der Anwendung ab.

CRC8:

Das „CRC8“ ist für eine CRC Checksumme gedacht, die die Erkennung von Übertragungsfehlern ermöglicht. Die Checksumme wird über das komplette Packet gebildet, um Fehler in jedem der Felder erkennen zu können.

Die CRC Checksumme wurde aus Performancegründen in den Mikroprozessorsystemen nicht implementiert.

5.1.2.2. Übertragung und Kapselung

Um die Datenpakete über eine serielle RS232 übertragen zu können, werden die einzelnen Datenpakete mittels der Mechanismen des altbewährten SLIP Protokolls gekapselt und übertragen.

Als Basis für die SLIP Implementation diente die öffentlich zugängliche RFC1055, die unter http://tools.ietf.org/html/rfc1055 verfügbar ist.

5.1.2.3. Unterstützte Befehle

Status Befehle:

Befehl	Name	Daten	Rückgabewert	Bemerkung
0x01	CMD_OK	-	-	Befehl wurde erfolgreich ausgeführt (wird nur als Antwort verwendet)
0x02	CMD_FAIL	-	-	Fehler beim Ausführen des Befehls (wird nur als Antwort verwendet)
0x03	CMD_BUSY	-	-	Fehler beim Ausführen des Befehls, das Gerät ist nicht bereit (wird nur als Antwort verwendet)

Info Befehle:

Befehl	Name	Daten	Rückgabewert	Bemerkung
0x11	CheckStatus	-	CMD_OK – Gerät ist bereit CMD_BUSY – Gerät ist nicht bereit timeout – Gerät ist nicht bereit	Zum überprüfen ob die Gegenstelle bereit ist, wird der „CheckStatus“ Befehl eingesetzt, welcher eine Antwort von der Gegenstelle anfordert.

Power Management Befehle:

Befehl	Name	Daten	Rückgabewert	Bemerkung
0x21	LCD_ Backlight	0x00 disable 0x01 enable	CMD_OK CMD_FAIL	Ein- bzw. Ausschalten der Hintergrundbeleuchtung des LCD
0x22	LCD_ Backlight_ Current	0x00 to 0xFF	CMD_OK CMD_FAIL	Steuern der Stärke der LCD Beleuchtung
0x23	VGA	0x00 disable 0x01 enable	-	Aktivieren und deaktivieren des VGA Ausgangs

User Interface Befehle:

Befehl	Name	Daten	Rückgabewert	Bemerkung
0x31	Button_OK	-	-	Button „OK“ wurde gedrückt
0x32	Button_ Back	-	-	Button „Zurück“ wurde gedrückt
0x33	Button_GP1	-	-	Button „GP1“ wurde gedrückt
0x34	Button_GP2	-	-	Button „GP2“ wurde gedrückt
0x35	Wheel_ Right	-	-	Wheel oder Slider: Es wurde nach recht gescrollt
0x36	Wheel_ Left	-	-	Wheel oder Slider: Es wurde nach links gescrollt
0x37	4Way_UP	-	-	4 Wege Steuerung: Nach oben

0x38	4Way_ DOWN	-	-	4 Wege Steuerung: Nach unten
0x39	4Way_ LEFT	-	-	4 Wege Steuerung: Nach links
0x3A	4Way_ RIGHT	-	-	4 Wege Steuerung: Nach rechts

Mouse/Touchpad Emulation Befehle:

Befehl	Name	Daten	Rückgabewert	Bemerkung
0x51	Position_ Absolute	**data[0..1]** x Koordinate unsigned int **data[2..3]** y Koordinate unsigned int **data[4]** Linke Taste gedrückt **data[5]** Rechte Taste gedrückt	-	Absolute Position des Zeigergeräts (in Bezug einen in der Linken unteren Ecke Nullpunkt) Koordinaten: Wertebereich 0 … 1000
0x52	Position_ Relative	**data[0..1]** x Koordinate signed int **data[2..3]** y Koordinate signed int **data[4]** Linke Taste gedrückt **data[5]** Rechte Taste gedrückt	-	Relative Position des Zeigergeräts (in Bezug auf die aktuelle Cursor Position)

5.1.2.4. Implementierung

Die Implementierung der SIE wurde in der Datei „ui/Sie.c“ bzw. „ui/Sie.h“ erledigt.

Für die Verwendung in übergeordneten Schichten stehen folgende Funktionen zur Verfügung:

void **sieInit**()

SIE Initialisieren

unsigned char **sieCreate**(void (*receive_callback)(siePacket), void (*send_callback)(unsigned char), void (*error_callback)(unsigned char));

Neue SIE erzeugen

receive_callback	… Callback Funktion die aufgerufen wird, wenn ein Paket erfolgreich empfangen wurde. Als Übergabewert wird ein SIE Paket übergeben.
send_callback	… Funktion die aufgerufen wird, wenn ein Paket gesendet werden soll (Zugriff auf Hardware Schicht – meist rs232Write funktion) Als Übergabewert, wird ein Datenbyte übergeben
error_callback	… Callback Funktion die aufgerufen wird, wenn ein Fehler beim Empfangen eines Paketes aufgetreten ist. Als Übergabewert wird ein Error Code übergeben.

Als Rückgabewert wird eine ID zurückgegeben, die jede SIE eindeutig identifiziert.

void **sieSubmitPacket**(unsigned char sie_id, siePacket data)

Diese Funktion sendet ein siePacket über die gewählte SIE.

sie_id	… ID der SIE über die die Daten übermittelt werden
data	… Daten Packet, dass über die SIE übermittelt werden soll

void **sieSubmit**(unsigned char sie_id, unsigned char command, unsigned char length, unsigned char *data)

Diese Funktion sendet die einzelnen Felder eines SIE Packets über die gewählte SIE.

sie_id	… ID der SIE über die die Daten übermittelt werden
command	… Befehl der gesendet werden soll
length	… Länge der Nutzdaten
data[length]	… Nutzdaten (Array mit der Länge length)

void **sieRespond**(unsigned char sie_id, unsigned char pid, unsigned char command, unsigned char length, unsigned char *data)

Diese Funktion sendete eine Antwort auf ein empfangenes SIE Paket

void **sieReceive**(unsigned char sie_id, unsigned char data)

Diese Funktion muss vom Daten-Empfänger (dem RS232 Treiber) aufgerufen werden, wenn Daten empfangen wurden.

5.1.3. Der kapazitive Sensorcontroller Treiber

Der QT1106 ist der kapazitive Sensor, der sowohl auf der Hauptplatine, als auch auf der xRemote verwendet wird. Der Hersteller des QT1106, die Firma Quantum bzw. Atmel, bietet bereits einen vorgefertigten Treiber für 8051er basierte CPU, welcher nur mehr adaptiert werden musste.

Die Hardware – Zugriffsroutinen wurden auf den verwendeten ATmega1281 angepasst, und die Software wurde von einem Polling Betrieb, auf einen Interrupt basierten effizienteren Interrupt Betrieb umzustellen.

Nähere Informationen zum Treiber des QT1106 sind in der auf der Quantum Homepage in Form einer Application Note verfügbar. Die Application Note ist unter http://www.qprox.com/assets/Downloadablefile/AN-KD06_QT1106_demo_code_1.01.pdf verfügbar.

5.1.4. Der IO-Expander Treiber

Der IO-Expander Treiber ist relativ einfach aufgebaut, der MCP23017 16-bit High Speed IO-Expander ist in Register aufgeteilt. Er wird mit I²C mit einer Frequenz von bis zu 1.7MHz angesteuert.

5.1.4.1. Implementierung

void **IoxDDRWrite**(unsigned char port, unsigned char value);

Setzen des Data Direction Registers für die einzelnen IO-Pins, ist ein der Wert des DDR für einen PIN auf 1, so wird dieser als Ausgang angesehen.

port	… Port für den das DDR gesetzt werden soll, der Treiber ist dafür ausgerichtet, dass zwei IO-Expander mit jeweils zwei Ports mit je 8bit angesteuert werden können (Port 0 bis 3)
value	… Neuer Wert des DDR

unsigned char **IoxDDRRead**(unsigned char port);

Lesen des Data Direction Registers für die einzelnen Ports

port	… Port für den das DDR gelesen werden soll, der Treiber ist dafür ausgerichtet, dass zwei IO-Expander mit jeweils zwei Ports mit je 8bit angesteuert werden können (Port 0 bis 3)

void **IoxPORTWrite**(unsigned char port, unsigned char value);

Setzen des PORT Registers für die einzelnen IO-Pins, ist ein der Wert des PORTS für einen PIN auf 1, so wird dieser auf 1 gesetzt.

port	… Port für den das Port Register gesetzt werden soll, der Treiber ist dafür ausgerichtet, dass zwei IO-Expander mit jeweils zwei Ports mit je 8bit angesteuert werden können (Port 0 bis 3)
value	… Neuer Wert des DDR

unsigned char **IoxPORTRead**(unsigned char port);

Lesen des PORT Registers für die einzelnen Ports

port	… Port für den das Port Register gelesen werden soll, der Treiber ist dafür ausgerichtet, dass zwei IO-Expander mit jeweils zwei Ports mit je 8bit angesteuert werden können (Port 0 bis 3)

unsigned char **IoxPINRead**(unsigned char port);

Lesen des Pin Registers für die einzelnen Ports, welches die Zustände der einzelnen Eingangs-Pins repräsentiert port … Port für den das Pin Register gelesen werden soll, der Treiber ist dafür ausgerichtet, dass zwei IO-Expander mit jeweils zwei Ports mit je 8bit angesteuert werden können (Port 0 bis 3)

5.1.5. Der LED Mix Treiber

Der LED Mix Treiber ermöglicht das Puls-Breiten-Modulieren der, an die IO-Expander angeschlossenen RGB, LEDs. Die Routine ermöglicht die Einstellung der Farbe der RGB LED, es können 64 verschiedene Farben eingestellt werden. Der Treiber arbeitet über einen Timer, welcher die Hauptroutine mit einer Frequenz von 300 Hz anspricht und somit das Einstellen der Farbe mittels PWM ermöglicht, ohne das diese sichtbar Flackern.

5.1.5.1. Implementierung

void **ledMixInit**() Initialisieren der LED Mix Routine

void **ledMix**(unsigned char led, unsigned char red, unsigned char green, unsigned char blue) Setzen der Farbwerte für die einzelnen LEDs led … LED für den die Farbe eingestellt werden sollen red … Roter Farbwert green … Grüner Farbwert blue … Blauer Farbwert

void **ledMixInterruptRoutine**() Interrupt Routine, diese Funktion muss vom Timer mit einer Frequenz von ~300 Hz (oder mehr) aufgerufen werden, um die Farbe flackerfrei mittels PWM einzustellen.

void **ledMixWheel**(unsigned char index, unsigned char red, unsigned char green, unsigned char blue); Einstellen der Farbwerte für die einzelnen LEDs, der Hintergrundbeleuchtung des Wheels. Index … Index der Hintergrundbeleuchtungs-LEDs (0 bis 3) red … Roter Farbwert green … Grüner Farbwert blue … Blauer Farbwert

void **ledMixButton**(unsigned char index, unsigned char red, unsigned char green, unsigned char blue);

Einstellen der Farbwerte für die einzelnen LEDs, der Hintergrundbeleuchtung der einzelnen Buttons.	
Index	… Index des Button-LEDs (0 bis 3)
red	… Roter Farbwert
green	… Grüner Farbwert
blue	… Blauer Farbwert

5.1.6. Das kapazitive Interface

Der “capacitiveInterface” Treiber, vereint nun Treiber des QT1106 und die LED Mix Routine zu einem abgeschlossenen Package. Die Routine stellt sicher, dass jeder Button und jedes Wheel Element richtig Hintergrundbeleuchtet ist. Außerdem initialisiert die Routine beide Treiber und leitet die Steuer-Befehle an die SIE weiter.

5.1.6.1. Implementierung

void **capInterfaceInit**();

Initialisieren des kapazitiven Interface

void **capInterfaceMain**();

Hauptroutine des Treibers, die Routine muss entweder periodisch, oder – wie verwendet – mittels Change Interrupt aufgerufen wird.

void **capInterfaceBlinkInitialSequence**(unsigned char index);

Initial Sequenz des Wheels blinken (drehen des Wheels, ansprechen der LEDs im Kreis)

Index	… Index, welcher angibt, welches LED aktiv ist

void **capInterfaceButtonsClear**();

Alle Buttons auf den Ausgangszustand zurücksetzen.

void **capInterfaceWheelClear**();

Alle LEDs der Wheels auf den Ausgangszustand zurücksetzen.

5.1.7. Der User Interface Treiber

Der User Interface Treiber bildet nun die Brücke zwischen der SIE und dem kapazitiven Interface, er vereint alles zu einem komplett Interrupt getreibenen Packet, welches nach der Initialisierung autonom arbeitet. Der Treiber sendet die Daten des kapazitiven Interfaces an die SIE und somit an den Hauptprozessor.

5.1.7.1. Implementierung

void **uiInit**(); UI Initialisieren, es wird alles auf Interrupt Basis initialisiert.

void **uiSieUnlock**(); SIE des User Interface freischalten, damit Daten übertragen werden können (bei der Initialisierung ist die SIE gesperrt)

5.2. Die xRemote

5.2.1. Übersicht

Das User Interface Konzept der xRemote

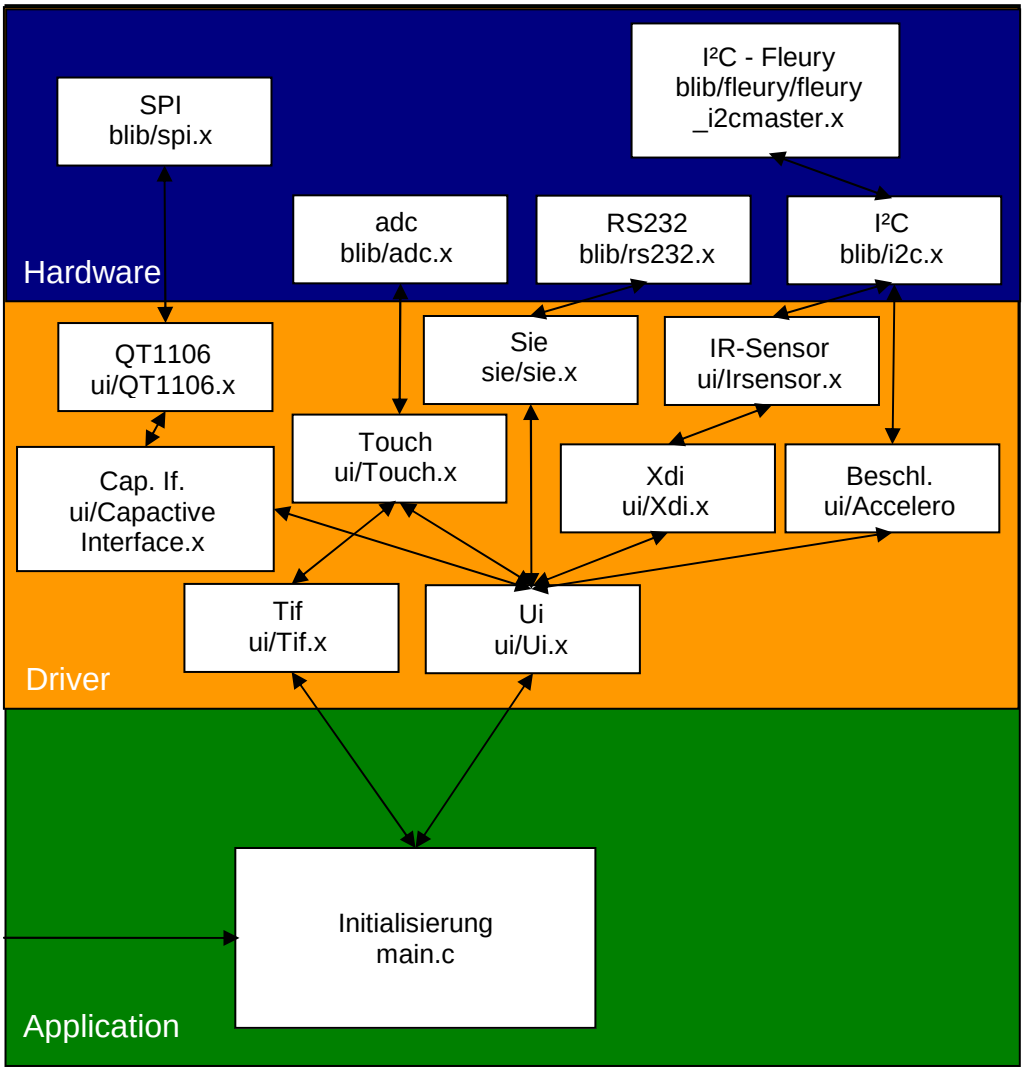


Abbildung 5.2.1-1 xRemote User Interface Konzept

Die xRemote hatte sehr viele Aufgaben, jedoch drehte sich alles um Usereingaben, d.h. um meist 4 bis 8 Kommandos (rechts, links, oben, unten oder weiter, zurück, Taste 1, Taste 2, Taste 3, Taste 4). Es musste der Touchscreen ausgelesen werden, die Werte des kapazitiven Sensors, sowie die Werte des Beschleunigungssensors verarbeitet werden. Außerdem dient der Prozessor der xRemote als Hauptprozessor für die xDimension, und so musste auch die Infrarot Kamera angesteuert werden.

Zur Abstraktion der ganzen Benutzereingaben wurde hier ein UI Layer eingeführt, der die Daten aller Benutzereingaben an zwei Ziele weiterleiten kann, einerseits an die Applikation die auf der xRemote selbst läuft und andererseits über das Bluetooth Modul direkt an de Boardcontroller und somit an die CPU. Mit einem sehr effizienten Event Routing ist es von der Applikation der xRemote sehr einfach Möglich die Ziele der Eingabedaten zu wählen.

Das GUI Konzept der xRemote

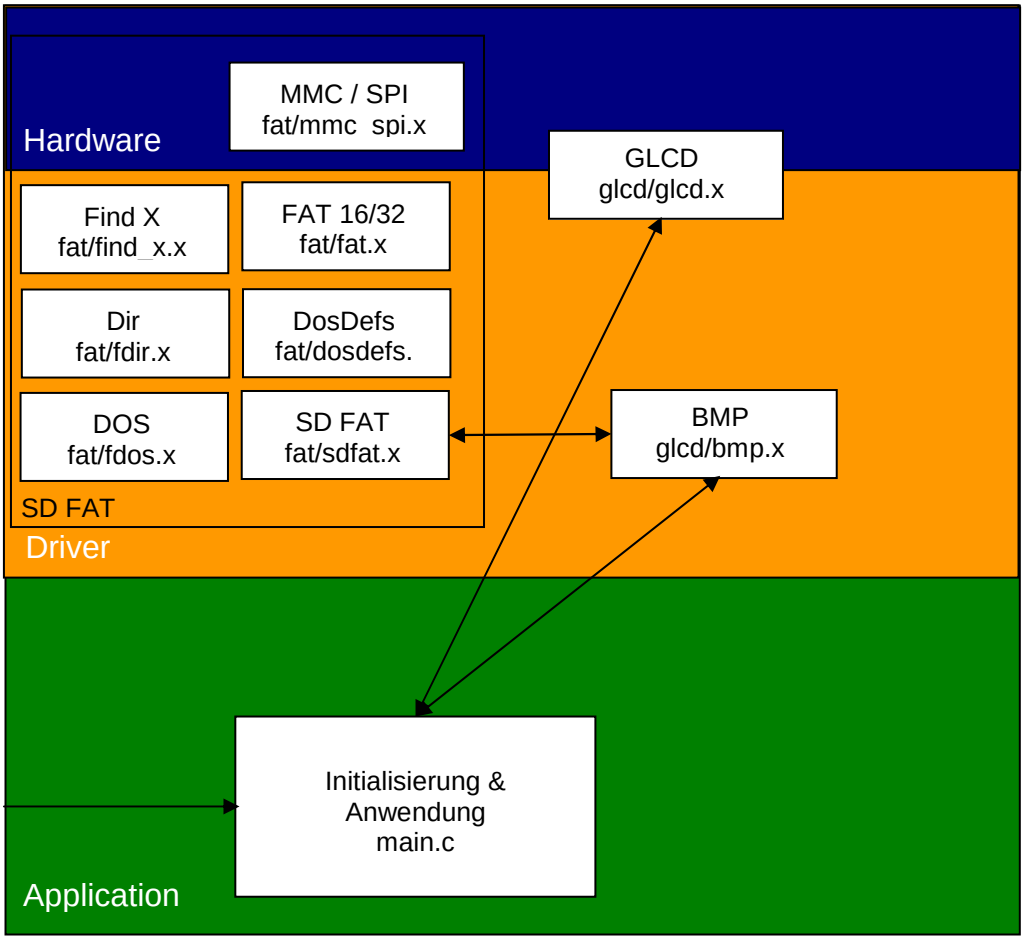


Abbildung 5.2.1-2 xRemote GUI Konzept

Außerdem bietet die xRemote die Möglichkeit SD / MMC und µSD Karten jeweils bis zu 2GB Kapazität zu lesen und auch das FAT16/12 Dateisystem zu verarbeiten, um Bilder, die im Flash – Speicher sehr viel Platz benötigen würden von einem Massenspeicher lesen zu können.

5.2.2. Der LCD Treiber

Der Treiber für den grafischen LCD basierte auf der vom Hersteller bzw. vom Verkäufer mitgelieferten Implementation für einen Philips LPC2138 ARM. Diese Implementation wurde überarbeitet und and den ATmega128 angepasst, außerdem wurde der Treiber stilistisch überarbeitet und dem Programmierstil im Projekt angepasst. Der Treiber ist bis an seine Grenzen auf optimale Performance optimiert, um schnellstmögliche Anzeige zu gewährleisten.

http://www.sparkfun.com/commerce/product_info.php?products_id=257

void **glcdInit**();

Den Grafik LCD initialisieren

void **glcdWrite**(char *line, unsigned char x, unsigned char y);

Einen Text - String auf das Display ausgeben.
line … Text Zeile als Zeichen Array

x	… Zeile (0 bis 8)
y	… Zeichen (0 bis 128)

void **glcdPutChar**(unsigned char c, unsigned char x, unsigned char y);

Ein Zeichen auf das Display schreiben

c	… Zeichen das auf den Display geschrieben werden soll
x	… Zeile (0 bis 8)
y	… Zeichen (0 bis 128)

void **glcdWriteData**(unsigned char data, unsigned char page, unsigned char column);

Ein Rohdaten Byte auf das Display schreiben (das Display wird zeilenweise beschreiben, jedes Bit stellt ein Pixel in der Zeile dar)

data	… Rohdatenbyte das auf das Display geschrieben werden soll
page	… Zeile (0 bis 8)
column	… Zeichen (0 bis 128)

void **glcdWriteDataLine**(char *data, unsigned char length, unsigned char page, unsigned char column);

Rohdaten auf den Display schreiben (das Display wird zeilenweise beschreiben, jedes Bit stellt ein Pixel in der Zeile dar)

data	… Rohdaten Array das auf das Display geschrieben werden soll
page	… Zeile (0 bis 8)
column	… Zeichen (0 bis 128)

void **glcdDrawPixel**(unsigned char value, unsigned char x, unsigned char y);

Ein Zeichen auf das Display schreiben

value	… Bit setzen / löschen
x	… X – Koordinate (0 bis 64)
y	… Y – Koordinate (0 bis 128)

5.2.3. Der SD Karten Treiber

Der SD Karten Treiber war einer der kompliziertesten Punkte des ganzen Projektes. Es musste zunächst auf die Mikro SD Karte zugegriffen werden, diese Initialisiert und anschließende das FAT Dateisystem gelesen werden – das alles mit einem mit 12MHz getakteten Mikroprozessor.

Da das Schreiben einer kompletten neuen FAT / SD Karten Implementation bei weitem zu zeitaufwändig gewesen wäre, entschieden wir uns für einen vorgefertigten Treiber. Dieser Treiber musste nun noch für den Prozessor angepasst und optimiert werden.

http://www.holger-klabunde.de/avr/avrboard.htm

Da dieser Treiber jedoch sehr unübersichtlich und unsauber war, wurden für den effizienten Zugriff einige vereinfachende Funktionen über den gesamten Treiber gelegt.

void **sdfatInit**();

Initialisieren des SD Karten Treibers

void **sdfatLoad**();

Dateisystem laden
Es wird 100mal versucht ein Dateisystem von der SD Karte zu laden

unsigned int **sdfatCapacity**();

Ermitteln der Größe der eingesetzten SD Karte, die Größe wird in MB zurückgegeben.

unsigned char **sdfatOpen**(char *name, unsigned char flag);

Eine Datei auf der SD Karte öffnen, Dateizeiger wird zurückgegeben
name …Name der zu öffnenden Datei
flag … F_READ – Datei lesen
F_WRITE – Datei schreiben
F_APPEND – Daten anfügen

unsigned char **sdfatSeek**(int offset, unsigned char mode);

Zu einer bestimmten Stelle in der geöffneten Datei springen
offset … Stelle an die gesprungen werden soll
mode … Modus:
SEEK_SET – Offset wird vom Beginn der Datei gezählt
SEEK_CUR – Offset wird von der aktuellen Position gezählt
SEEK_END – Offset wird vom Ende der Datei gezählt (offset negative

unsigned int **sdfatRead**(unsigned char *buf, unsigned int count);

Aus der geöffneten Datei lesen, die Anzahl der gelesenen Bytes wird zurückgegeben
buf … Puffer in den gelesen werden soll
count … Anzahl der Bytes die gelesen werden sollen

unsigned int **sdfatWrite**(unsigned char *buf, unsigned int count);

In die geöffnete Datei schreiben, die Anzahl der geschriebenen Bytes wird zurückgegeben
buf … Puffer aus dem geschrieben werden soll
count … Anzahl der Bytes die geschrieben werden sollen

void **sdfatClose**();

Geöffnete Datei schließen.

5.2.4. BMP Loader

Die Kombination zwischen dem Grafik LCD Treiber und dem SDFAT Treiber bildet nun der BMP Loader. Diese Routine ermöglicht es, monochrome BMP Bilder von der SD Karte zu lesen und auf dem Display anzuzeigen. Diese Routine bringt den Prozessor bis an seine Leistungsgrenze, und wurde teilweise so weit optimiert, dass die Kern-Funktion sehr komplex wirkt.

Der BMP Loader stellt folgende Interface Funktionen zur Verfügung:

unsigned char **bmpLoad**(char *file, unsigned char page, unsigned char column);

Lädt ein Bild von der Speicherkarte und zeigt es am Display an.
file … Dateiname des zu ladenden Files
page … Display „Zeile“ in der mit dem Anzeigen des Bildes begonnen werden soll. (0 bis 8)
column … Display „Spalte“ in der mit dem Anzeigen des Bildes begonnen werden soll (0 bis 128)

5.2.5. Der Beschleunigungssensortreiber

Der Treiber für den Beschleunigungssensor ist für die Verwendung mit dem UI / Event Routing System implementiert, er wird einmal initialisiert und dann periodisch aufgerufen um neu Daten vom Sensor abzufragen. Nach dem Abfragen, werden die Daten analysiert, wurde eine Bewegung entdeckt, die stark genug ist, wird Signal (callback) an die UI Routine ausgegeben.

void **accelerometerInit**(void (*ui_callback)(unsigned char command));

Initialisieren des Beschleunigungssensors
ui_callback … Funktion die aufgerufen wird, sobald ein Event (Bewegung) aufgetreten ist.

void **accelerometerMain**();

Hauptroutine, diese Routine muss periodisch aufgerufen werden!

5.2.6. Touch Screen & Touch Interface

Der Touchscreen Treiber besteht aus zwei Routinen, dem Low-Level Touchscreen Treiber und dem abstrahierenden Touch Interface Treiber.

Der Touch Screen Treiber arbeitet mit dem ATmega internen ADC. Der ADC läuft im Continous Mode, d.h. er wandelt andauernd im Interrupt Betreib. Des ermöglicht es, dass der komplette Treiber nur einmal initialisiert werden muss und danach komplett eigenständig arbeitet. Der Touch Screen Treiber mittel jeweils 200 ADC Wert für den x Wert und 200 für den y Wert, um maximale Genauigkeit und minimale Störungsanfälligkeit zu erreichen.

void **touchInit**(void (*position_callback)(unsigned int, unsigned int, unsigned char));

Touch screen & ADC initialisieren, die A/D Wandlung läuft danach im Interrupt Betrieb im Hintergrund ab.
position_callback … Diese Callback Funktion wird aufgerufen, wenn ein Druck auf den Touchscreen erkannt wurde.

Der Touch Interface Treiber hat nun die Aufgabe, die Zwischenschicht zwischen Anwender und den Touch Screen Treiber zu bilden. Er ermöglicht es, sensitive Bereiche auf dem Display festzulegen, die ein Event (callback) hervorrufen.

void **tifInit**(void (*callback)(unsigned int, unsigned int, unsigned char));

Touch Interface initialisieren.
Callback … Callback Funktion die aufgerufen wird, wenn ein Event aufgetreten ist. Als Übergabeparameter werden die x und die y Koordinaten, sowie die Event ID übergeben.

unsigned char **tifEventRegister**(unsigned char uid, unsigned int x, unsigned int y, unsigned int width, unsigned int height);

Neues Event registrieren (ein Event ist eine Berührung in einer bestimmten Region des Touch Screens)
uid … Frei festlegbare Identifikationsnummer, die das Event eindeutig identifizieren soll.
x … X Koordinate des Linken oberen Eckes des Event Rechteckes
y … Y Koordinate des Linken oberen Eckes des Event Rechteckes

width	… Breite des Event Rechteckes
height	… Höhe des Event Rechteckes

void **tifEventClearAll**();

Alle registrierten Events löschen

void **tifEventClear**(unsigned char event);

Ein spezifisches Event löschen
event … ID des Events

5.2.7. QT1106 & Capactive Interface

Das kapazitive Interface in der xRemote wurde im Vergleich zum Interface im Boardcontroller stark reduziert, da keine Ansteuerung von Hintergrundbeleuchtung nötig ist. So stehen nur zwei öffentliche Funktionen zur Verfügung.

void **capInterfaceInit**(void (*ui_callback)(capInterfaceKey, char));

Das kapazitive Interface initialisieren, als Übergabeparameter wird eine Callback Funktion übergeben, die aufgerufen wird, sobald eine Änderung auftritt.
ui_callback … Callbackfunktion, die bei Änderungen aufgerufen wird.

void **capInterfaceMain**();

Diese Funktion muss periodisch oder nach einem CHANGE Interrupt des QT1106 aufgerufen werden und ist der Haupt Part des kapazitiven Interfaces.

Der Treiber für den QT1106 ist wieder ein leicht modifizierter Treiber vom Prozessorhersteller Atmel. Näheres unter http://www.qprox.com/assets/Downloadablefile/AN-KD06_QT1106_demo_code_1.01.pdf.

5.2.8. User Interface

Einer der größten Firmwareteile der xRemote ist der User Interface Treiber, dieser kombiniert Capactive Interface, Touch Screeen, Accelerometer und xDimension zu einem generalisierten Interface. Die einzelnen Sub-Treiber senden alle ihre Event an den UI Treiber und dieser verarbeitet diese dann und routet sie ans richtige Ziel. Grundsätzlich gibt es zwei Ziele, die xSie und somit der xMedia Player oder das Programm selbst. Über ein ausgeklügeltes Event Routing System können nun die Ziele für die Events von jedem Eingabegerät gewählt werden.

void **uiInit**();

User Interface initialisieren. Im Hintergrund wird die Initialisierung für die SIE, das kapazitive Interface, den Touch Screen, den Beschleunigungssensor und die xDimension Firmware durchgeführt.

void **uiMain**();

Diese Funktion muss periodisch aufgerufen werden, im Hintergrund ruft sie die Hauptroutinen für alle verwendeten Treiber auf. Das Timing wird intern mit einem Timer

geregelt, d.h. die Funktion muss nur aufgerufen werden, jedoch nicht in einem bestimmten Takt.

void **uiSetCommandHandler**(void (*cmd_callback)(unsigned char));

Diese Funktion setzt eine Callback Funktion, die aufgerufen wird, wenn ein Event an eine interne Quelle geroutet wird, d.h. wenn das Ziel für Kommandos des Kapazitiven Interfaces auf intern gestellt wird, wird die Callback Funktion aufgerufen, und das Kommando übergeben.

cmd_callback … Callback Funktion die aufgerufen wird, wenn intern ein Kommando gesendet werden soll (als Parameter wird das Kommando übergeben)

void **uiSetPositionHandler**(void (*position_callback)(unsigned int, unsigned int, unsigned char));

Diese Funktion setzt eine Callback Funktion, die aufgerufen wird, wenn eine Positionsänderung an eine interne Quelle geroutet wird, d.h. wenn das Ziel für Positionsänderungen des Touchscreens auf intern gestellt wird, wird die Callback Funktion aufgerufen, und die Positionswerte übergeben.

position_callback … Callback Funktion die aufgerufen wird (als Parameter werden die x und die y Koordinate übergeben, angepasst auf die eingestellte Bildschirmgröße)

void **uiSetScreen**(unsigned int width, unsigned int height);

Diese Funktion setzt die Größe des Bildschirms auf den die Positionswerte hochgerechnet werden.
width … Breite des Bildschirms
height … Höhe des Bildschirms

void **uiSetRouting**(unsigned char key, unsigned char value);

Aktivieren und deaktivieren der Event Weiterleitung für bestimmte Events, folgende Werte für „key“ stehen zur Verfügung:

Touch Screen:
UI_ROUTE_TOUCH_POSITION_INTERNAL
UI_ROUTE_TOUCH_POSITION_EXTERNAL

Kapazitives Interface
UI_ROUTE_CAPACITIVE_COMMANDS_INTERNAL
UI_ROUTE_CAPACITIVE_COMMANDS_EXTERNAL

xDimension
UI_ROUTE_XDI_COMMANDS_INTERNAL
UI_ROUTE_XDI_COMMANDS_EXTERNAL
UI_ROUTE_XDI_POSITION_INTERNAL
UI_ROUTE_XDI_POSITION_EXTERNAL

Beschleunigungssensor
UI_ROUTE_ACCELEROMETER_COMMANDS_INTERNAL
UI_ROUTE_ACCELEROMETER_COMMANDS_EXTERNAL

key … Schlüssel der das Event identifiziert
value … Aktivieren / Deaktivieren des Events

5.2.9. IR – Sensor & XDimension

Die Ansteuerung des Infrarot Sensors und die Bearbeitung der Daten wird auf Grund des Wunsches nach der Patentierung des Systems hier nicht angeführt!

6. Betriebssystem und der Software

Der xMedia Player sollte sich nicht nur auf Hardware Ebene als ausgewachsener und marktreifer Player präsentieren, sondern auch auf der Software Ebene.

Als Basis für den Player wählten wir den Linux Kernel mit einer Busybox Umgebung.

Das Buildroot System (welches unter anderem von Atmel, dem Hersteller des AP7000 Prozessors angeboten wird) ermöglicht es relativ einfach eine komplett eigene Linux Distribution zu erstellen. Außerdem können ausgewählte Anwendungen sehr simpel crosskompiliert und in Distribution eingebunden werden.

Die Auswahl der richtigen Programmiersprache und der richtigen Programmierumgebung war auch ein wichtiger Punkt. Grundsätzlich kamen hier die Programmiersprachen Java und C++ in nähere Betrachtung. Die AP7000 CPU hat zwar eine spezielle Java Optimierung und Acceleration, bleibt jedoch eine halb interpretierte Programmiersprache und hat einen merklichen Performancenachteil gegenüber des machinennahen C++. Da der xMedia Player über stark begrenzte Ressourcen verfügt, wählten wir daher C++.

Einer der wichtigsten Punkte war die Wahl eines Window Systems für Linux bzw. einer Grafikbibliothek. Zunächst tendierten wir zu, dem bei Desktopsystemen eingesetzten, XWindows bzw. zu dem normalerweise auf Embeddedsystemen eingesetzten nanoX Dies wollten wir mit der Grafikbibliothek GTK2 kombinieren. Doch bei näherer Betrachtung tendierten wir immer mehr zu QT. QT wurde zunächst von Nokia für die Verwendung auf ihren Mobiltelefonen entwickelt. Durch die Firma Trolltech, die später um QT entstanden ist, entwickelte sich die Bibliothek QT schnell weiter und wurde, aus unserer Sicht, zu einer der mächtigsten C++ Bibliotheken auf dem Markt. C++ mit QT ist mit C++ ohne QT nicht mehr vergleichbar, die Sprache ändert mit der Bibliothek ihren kompletten Charakter. Von einer viel zu komplexen, zwar mächtigen, aber in die Jahre gekommenen Programmiersprache zu einer modernen Sprache die ohne weiteres mit Sprachen wie Java und C# mithalten kann. Einer der weiteren Vorteile von QT ist, dass es Plattform unabhängig ist. Die Anwendungen müssen neu kompiliert werden, jedoch die Funktionalität der Applikation bleibt ohne Änderung des Codes gleich, egal ob Linux, Windows oder OS X. Des Weiteren inkludiert QT Embedded die Möglichkeit die Applikationen direkt, ohne Window System zu starten, d.h. die Anwendung bzw. QT schreibt direkt in den Linux Framebuffer.

Abbildung 16: QT Logo

Das Phonon Framework zum Abspielen multimedialer Inhalte in QT war leider nicht für die Benutzung in integrierten Systemen konzipiert und es wurde stattdessen der von Atmel spezielle optimierte MPlayer verwendet. MPlayer ist ein gut anpassbarer Multimediaplayer und ist ein echtes Leichtgewicht was Prozessor und RAM Auslastung betrifft.

6.1. Software Übersicht

Betrachtet man den Software Aufbau genauer, ist die Software des xMedia Players in mehreren Schichten aufgebaut.

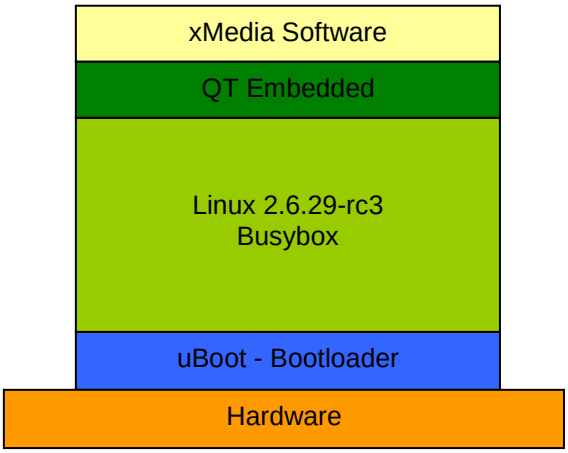


Abbildung 6.1-1 Software Architektur

Direkt auf, auf unterster Ebene setzt uBoot Bootloader auf, der Bootloader setzt die Grundeinstellungen des Prozessors (Taktfrequenzen,…), ermöglicht das Booten von Linux und Programmieren des Flash Speichers.

Die Hauptkomponente bildet die mit Buildroot erstellte Linux Umgebung mit der Busybox Konsolenumgebung.

Darauf setzt die QT Embedded Bibliothek auf, die das Windowsystem und diverse Bibliotheken bereitstellt.

Über all dem steht die entwickelte xMedia Software, die die gesamte grafische Oberfläche des Players bildet und die Wiedergabe von Musikdateien ermöglicht.

6.2. Installation eines Entwicklungssystems

Als Entwicklungssystem wurde die Desktop Version vom Linux Derivat Ubuntu in der Version 8.04 gewählt. Da das Entwicklungssystem auf mehreren PCs eingesetzt werden sollte, entschieden wir uns für die Virtualisierung des Entwicklungssystems mittel der OpenSource Software Virtual Box.

Nach dem Download von Virtual Box von http://www.virtualbox.org/wiki/Downloads präsentiert sich Virtual Box mit folgendem Interface:

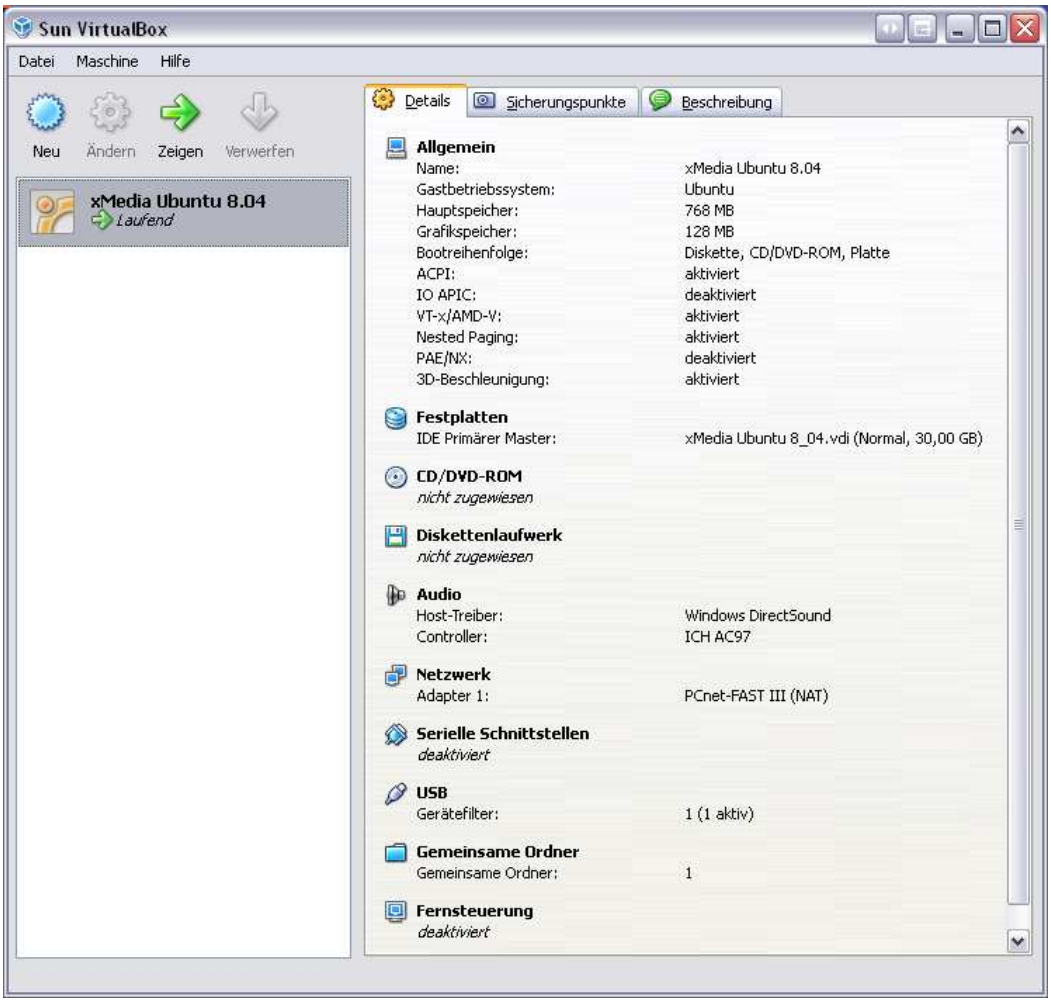


Abbildung 6.2-1 Virtual Box Screenshot

Nach dem Anlegen einer neuen virtuellen Maschine, in unserem Fall mit 768Mb Arbeitsspeicher (Lauffähig auf Notebook – Core 2 Duo 2.4Ghz, 4GB Ram – und auf einem Stand PC – Core 2 Quad, 2GB Ram), 128 MB Grafikspeicher und allen Funktionen die die virtuelle Maschine beschleunigen (VT-x/AMD-V, 3D-Beschleunigung, …) aktiviert. Als Festplattengröße wurde eine 30GB große dynamisch wachsende Festplatte gewählt. Es Stellte sich jedoch heraus, dass die 30Gb etwas knapp bemessen waren. Das Festplatten Image sollte wenn möglich auf eine externe Festplatte gespeichert werden, um dieses portabel und auf mehreren PC's einsetzbar zu machen.

Nach dem Anlegen der virtuellen Maschine und dem Download des Betriebssystems von http://www.ubuntu.com/ kann das herunter geladene ISO Image einfach und ohne vorheriges Brennen eingebunden werden. Beim ersten Starten der virtuellen Maschine führt ein Assistent durch das Einbinden des Images, hier muss nur das Heruntergeladene Image ausgewählt werden.

Nach dem Starten muss zunächst die Sprache ausgewählt werden, je nach belieben Deutsch oder Englisch.

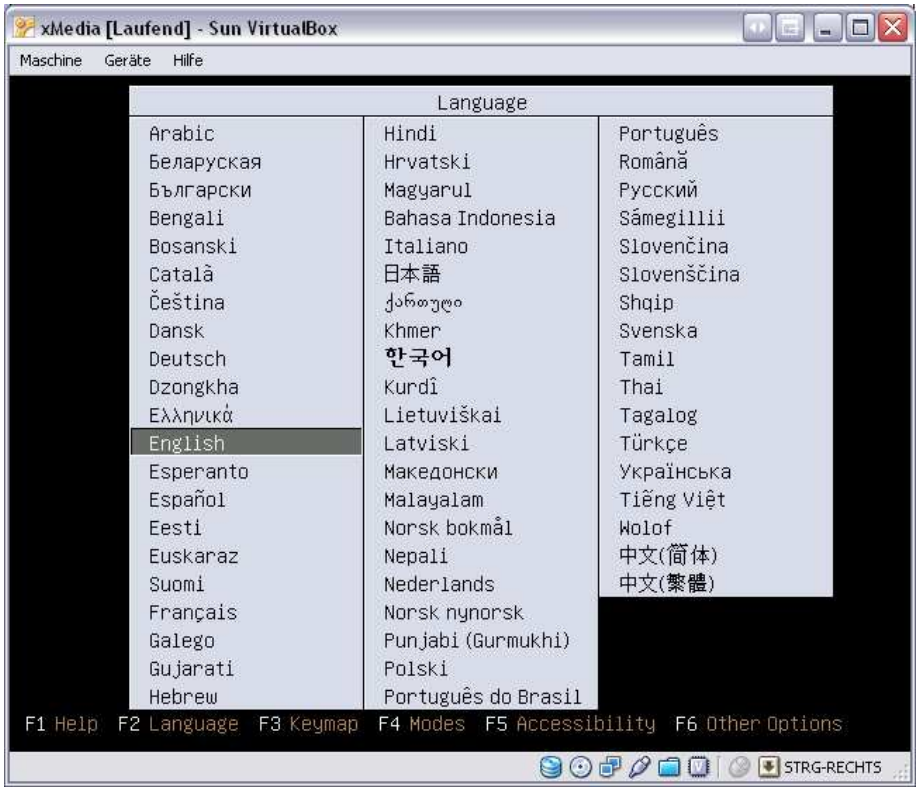


Abbildung 6.2-2 Installation - Sparchauswahl

Der einfachste Weg um Ubuntu nun zu installieren, ist nicht über den Punkt „Ubuntu installieren“, sondern über den Punkt „Ubuntu ausprobieren“.

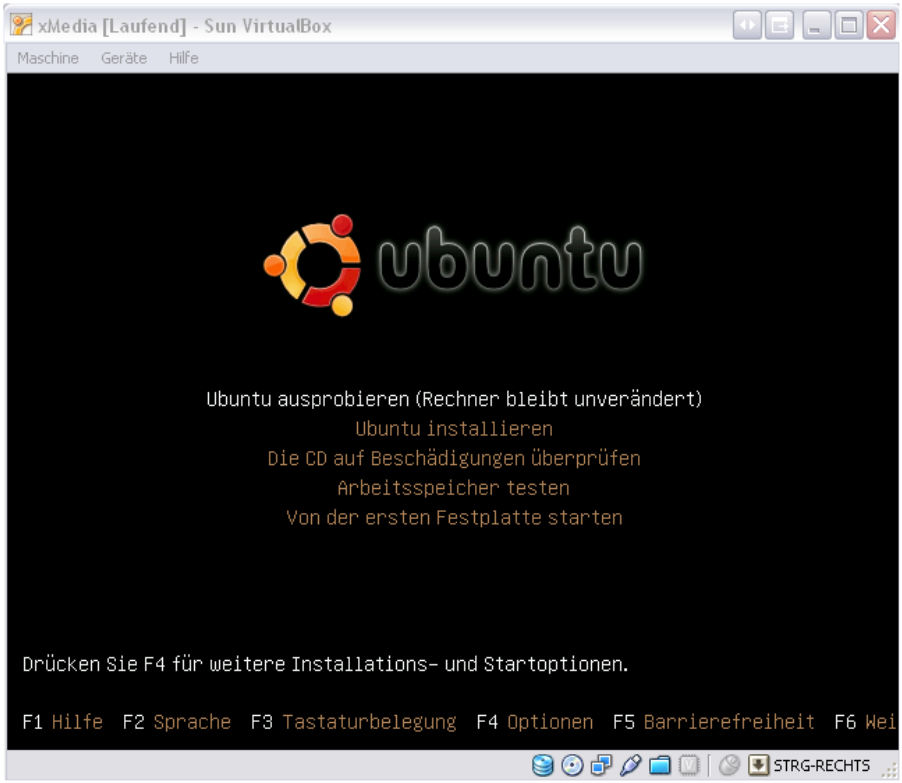


Abbildung 6.2-3 Installation starten

Nach dem Starten, gelangt man direkt zu Desktop des Betriebssystems. Der einfachste Weg das System nun zu Installieren ist über einen einfachen Klick auf die „Installieren“ Verknüpfung.

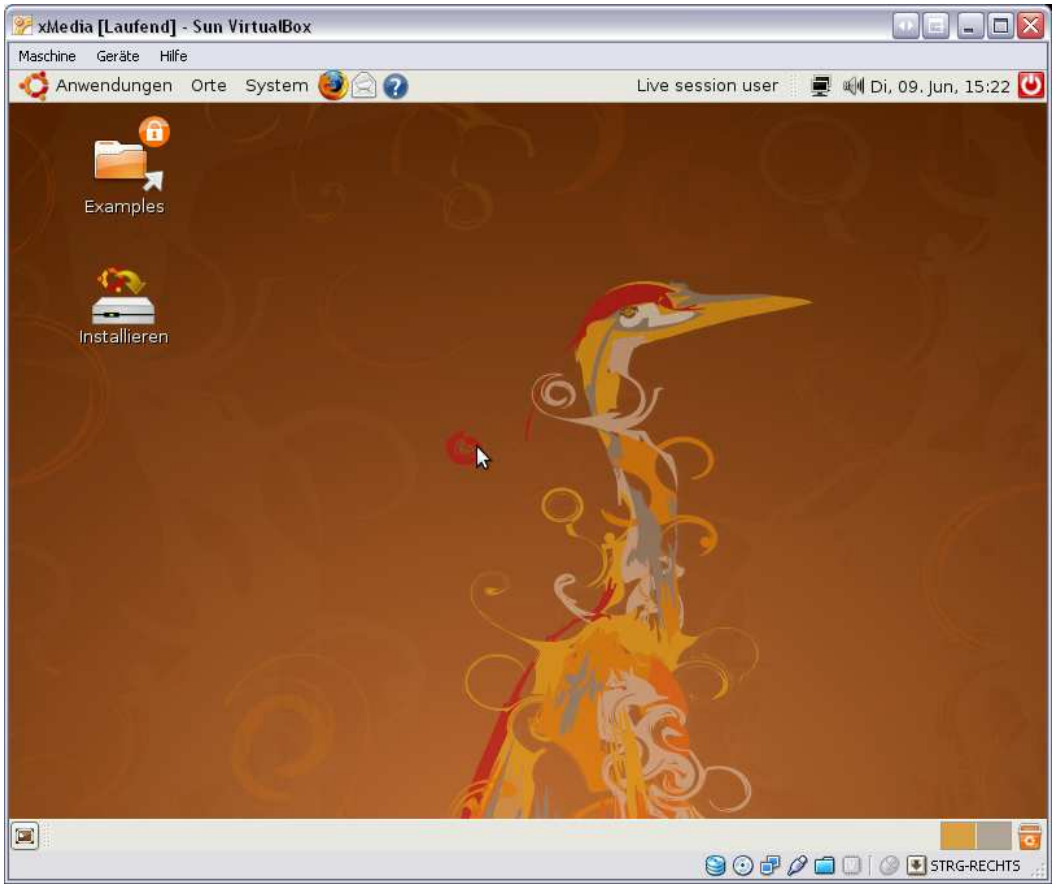


Abbildung 6.2-4 Installation Desktop

Nach dem Starten des Installers für ein einfacher Schritt-für-Schritt Wizzard führt im Anschluss durch die komplette Installation. Die meisten Punkte sind klar und verständlich erklärt und die Installation geht so recht einfach von der Hand.

Bei der Partitionierung wurden einige manuelle Einstellungen getroffen, und das Dateisystem auf zwei bzw. drei Partitionen aufgesplittert.
Nach dem Anlegen einer neuen Partitionstabelle, wurde zunächst eine Hauptpartition mit 13GB erstellt, eine SWAP Partition mit 2GB und der Rest wurde für eine Datenpartition verwendet.

Größe	Dateisystem	Mountpoint
13 GB	EXT3	/
2 GB	SWAP	swap
15 GB	EXT3	/xmedia

Der Einhängepunkt für die Datenpartition wurde mit /xmedia gewählt um schnell und einfach darauf zugreifen zu können.

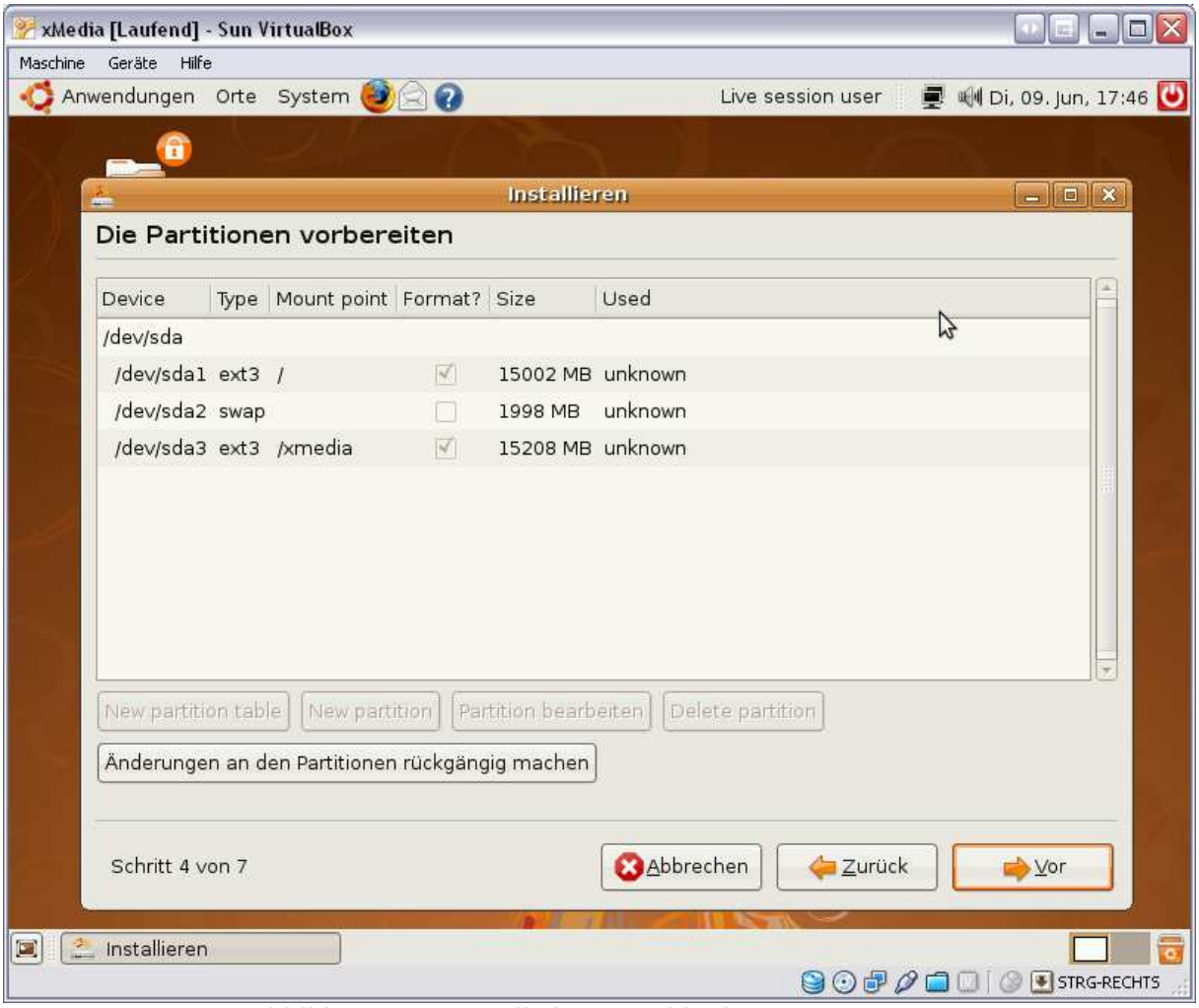


Abbildung 6.2-5 Installation - Partitionierung

Nach der Partitionierung und der Eingabe diverser Benutzerdaten, ist das System bereit für die Installation. Der Installationsprozess dauert einige Minuten und funktioniert auf Vritual Box fehlerfrei.

Nach einem Neustart und der Anmeldung mit dem gewählten Benutzernamen und Passwort ist das System einsatzbereit.

6.3. Installation der Gasterweiterung

Um Nahtlos mit dem virtuellen Entwicklungssystem arbeiten zu können, muss die Gasterweiterung auf dem Ubuntu System installiert werden. Dies ist sehr einfach, da VirtualBox bereits eine fertige Installationsroutine bereitstellt. Über den Menüpunkt „Gerät“ -> „Gasterweiterungen installieren“ wird ein neues CD Laufwerk eingebunden

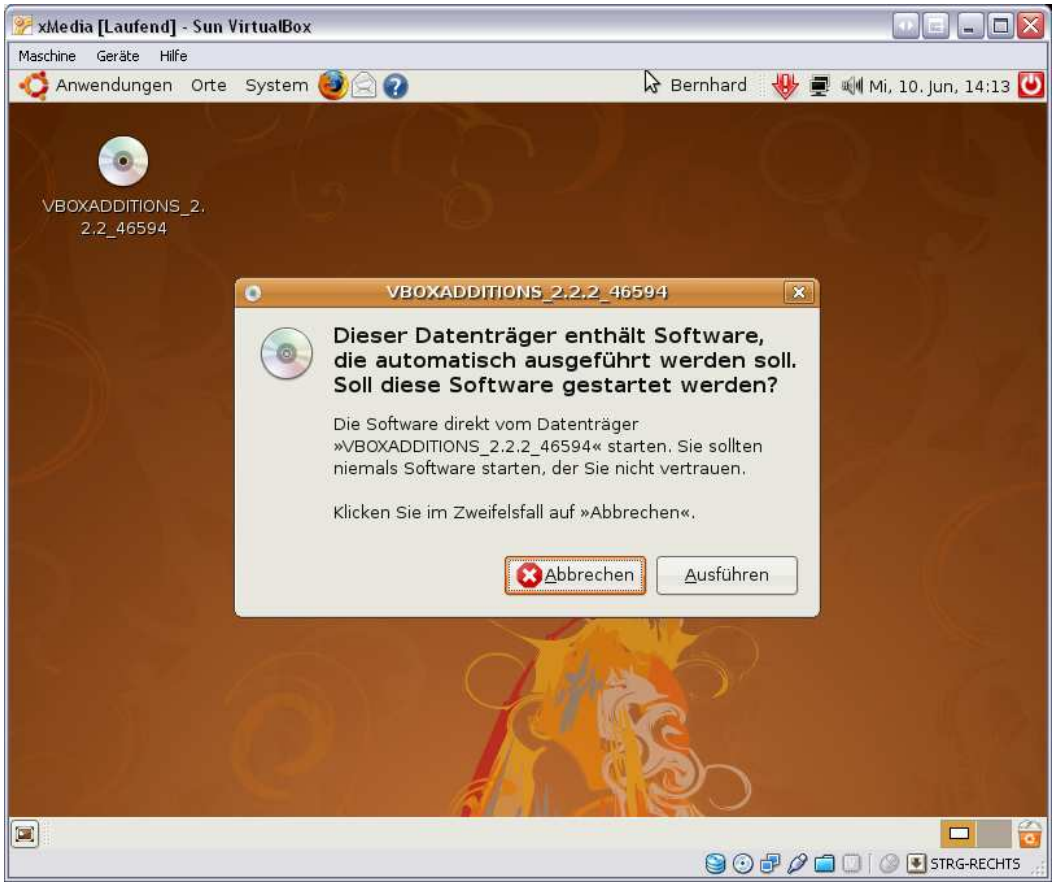


Abbildung 6.3-1 Installation Gasterweiterung

Nach einem Klick auf „Ausführen“, wird die Gasterweiterung installiert und in das Betriebssystem integriert. Nach einem erneuten Neustart die nun alles einsatzbereit (gemeinsame Zwischenablage, gemeinsame Ordner, nahtloser Mauszeiger, Grafiktreiber, …).

6.4. Gemeinsame Ordner erstellen

Um nun Daten zwischen dem Hostsystem und dem Entwicklungssystem austauschen zu können muss ein gemeinsamer Ordner erstellt werden.

Dieser kann über „Gerät“ -> „Gemeinsamer Ordner“ erstellt werden. Der „Ordner-Pfad“ gibt den Pfad auf dem Host Dateisystem an und der „Ordner-Name“ gibt einen Namen an, mit dem der Ordner im Gastsystem identifiziert werden kann.

Die gewählten Einstellungen sind „T:\xChange“ für den Ordner-Pfad und „xChange“ für den Ordner-Namen, außerdem muss „Permanent erzeugen“ gewählt werden.

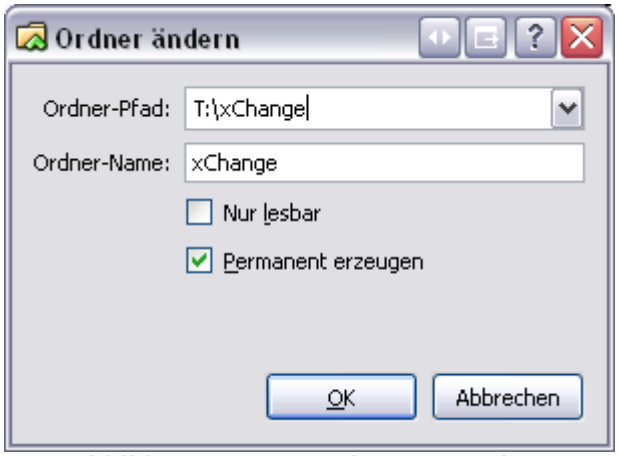


Abbildung 6.4-1 Gemeinsamer Ordner

Nachdem der Ordner am Host erstellt worden ist, muss er noch am Gast eingebunden werden. Zunächst muss ein neuer Ordner im Host Dateisystem angelegt werden, da dies nur mit administrativen Rechten möglich ist, muss alles über Konsole ausgeführt werden.

```
sudo mkdir /xchange/
sudo chmod 0777 /xchange/
```

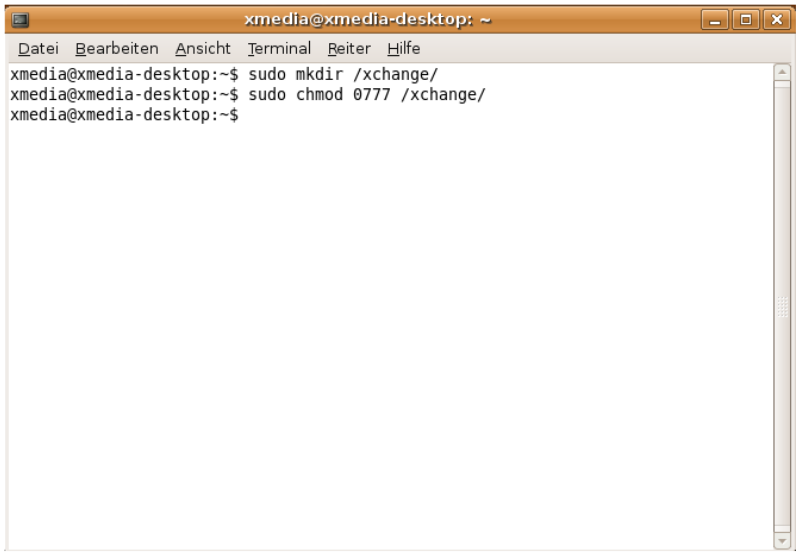


Abbildung 6.4-2 Ordner erstellen

Nun muss der gemeinsame Ordner noch automatisch in das Dateisystem eingebunden werden. Dies geschieht über einen Eintrag in der Datei, welche aus Rechtegründen wiederum von der Konsole aus editiert werden muss.

```
sudo vim /etc/fstab
```

Über einen Druck auf „i“ wird beim VI-Editor in den Eingabemodus gewechselt, nun muss folgender Eintrag angehängt werden:

```
xChange   /xchange      vboxsf  defaults        0       0
```

Über einen Druck auf die ESC-Taste und das eingeben von „:wq“ wird die Datei gespeichert und geschlossen.

Nach einem erneuten Neustart (oder dem Ausführen des Befehls „sudo mount –a“) wird der Ordner nun automatisch eingebunden und ist nun einfach benutzbar.

6.5. Installieren der benötigten Software

Zum Kompilieren aller Softwarekomponenten müssen noch einige Tools installiert werden.

Um alle Benötigten Abhängigkeiten automatisch zu installieren, wurde eine Konfigurationsdatei erstellt, der dies automatisch erledigt. Die Datei ist auf dem beiliegenden Datenträger unter „Linux/installed-software.is“ zu finden.

Doch zunächst muss noch ein entsprechender Paketmirror zur Konfigurationsdatei „/etc/apt/soures.list“ hinzugefügt werden.

Die muss wieder über die Konsole erledigt werden:

```
sudo vim /etc/apt/sources.list
```

Im Editor muss anschließend die Zeile

```
deb http://www.atmel.no/avr32/ubuntu/hardy binary/
```

ans Ende der Datei eingefügt werden. (i um in den Einfügemodus zu wechseln, ESC + „:wq“ zum Speichern)

Um nun alle gewünschten Packages auszuwählen, muss der folgende Befehl ausgeführt werden:

```
sudo dpkg --set-selections < installed-software.is
```

Um nun alle Pakete herunterzuladen, muss folgender Befehl ausgeführt werden (alle verfügbaren Updates werden auch automatisch mitinstalliert):

```
sudo apt-get update
sudo apt-get install dselect
sudo apt-get upgrade
sudo dselect
```

Nun muss im erscheinenden Menü nur noch “Install” gewählt werden. Im Laufe des Update Prozesses werden bis zu 2Gb an Daten heruntergeladen.

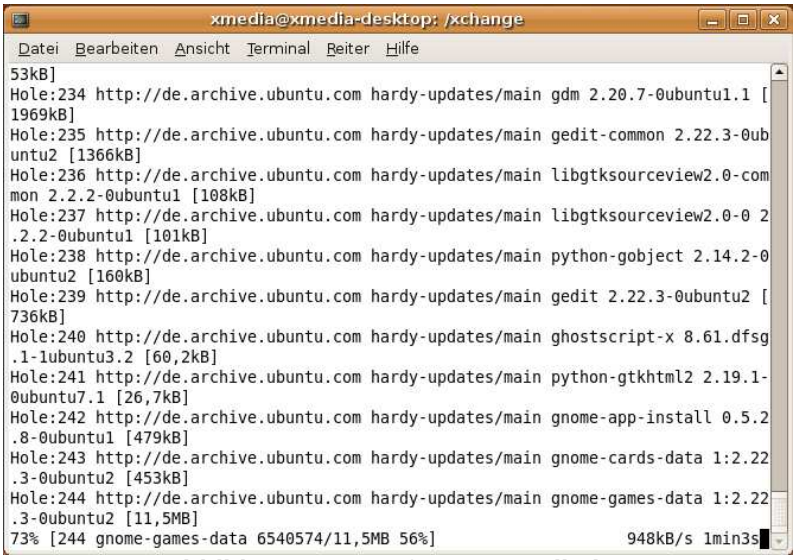


Abbildung 6.5-1 Software Installation

Der Installationsvorgang wird einige Minuten (je nach Internetanbindung) in Anspruch nehmen.

6.6. Installation von Buildroot

Nun geht es daran die Entwicklungsumgebung für den AVR32 Prozessor zu installieren. Hier wird das Toolset „Buildroot“ in einem speziell von Atmel abgewandelten Derivat in der Version 2.3 eingesetzt. Buildroot kann unter http://www.atmel.no/buildroot/ heruntergeladen werden.

Nach dem Download des Buildroot Packages, muss es in das Datenverzeichnis auf dem Entwicklungssystem entpackt (in das Verzeichnis „/xmedia/buildroot2_3“) werden. Nach dem Entpacken des Buildroot Tools, kann dieses eigentlich auch gleich verwendet werden.

Um nun jedoch noch die richtigen Einstellungen (d.h. welche Software soll wie installiert werden) zu laden müssen die Dateien mit dem Namen „.config“ und „.config.cmd“ noch mit den Dateien im „Linux/buildroot2_3“ Ordner des beiliegenden Datenträgers überschrieben werden. (Verborgene Dateien müssen im Datei Browser angezeigt werden, damit die Dateien sichtbar sind)

Nun muss die Buildroot Umgebung das erste Mal kompiliert werden. Dies dauert meist mehrere Stunden, da viele Dateien heruntergeladen und entpackt und kompiliert werden müssen.

Dazu muss nur in den Buildroot Ordner gewechselt werden und zunächst die Konfiguration kompiliert werden:

```
cd /xmedia/buildroot2_3/
make menuconfig
```

Die Konfiguration wieder beendet werden.

Und anschließend der

```
make
```

Befehl ausgeführt werden muss.

6.6.1. Fehler in Webkit beheben

Während des ersten Kompilieren auf der AVR32 Plattform, wirft die Bibliothek QT, genauer gesagt die Webkit Browser Engine, die in QT implementiert ist, einen Fehler.

Dieser kann durch das Anwenden eines vorgefertigten Patches behoben werden.

```
cd /xmedia/buildroot2_3/build_avr32/qt-embedded-linux-opensource-src-
4.4.3/src/3rdparty/webkit

patch -p1 < /xchange/Files/linux/patches/webkit/webkit-avr32.patch
```

Anschließend muss das Kompilieren wieder neu gestartet werden.

```
cd /buildroot2_3/
make
```

6.7. Modifizieren des Bootloaders uBoot

Der Bootloader, der mit Buildroot erstellt wird, ist auf eine Taktfrequenz von 140MHz und nur 32Mb Ram ausgelegt. Im Projektverlauf hat es sich jedoch herausgestellt, dass 140 bzw. 150 MHz zu wenig sind, weshalb entschlossen wurde mit 190MHz zu arbeiten.

Darum wurde ein Patch erstellt, dass das die Einstellungen des Bootloaders ändert.

```
cd /xmedia/buildroot2_3/project_build_avr32/xMedia/u-boot-1.3.4/
patch -p1 < /xchange/Files/linux/patches/uboot/clock.patch
```

6.7.1. Neu kompilieren von uBoot

Das neu Kompilieren von Uboot funktioniert wiederum über Buildroot:

```
cd /xmedia/buildroot2_3/
make u-boot-clean
```

Nun müssen noch die Dateien „uboot“ und „uboot.bin“ im Verzeichnis gelöscht werden und anschließend muss uBoot neu kompiliert werden:

```
rm /xmedia/buildroot2_3/project_build_avr32/xMedia/u-boot-1.3.4/u-boot
rm /xmedia/buildroot2_3/project_build_avr32/xMedia/u-boot-1.3.4/u-boot.bin
make u-boot
```

Nun sollte uBoot neu kompiliert worden sein und liegt mit der Datei „u-boot.bin“ in Binärform vor.

6.7.2. Bootloader Flashen

Für das erneute Flashen des Bootloaders gibt es im Grund drei Möglichkeiten:

1. Mit der alten U-Boot Version
2. Mit Linux und einem Update Programm
3. Mit einem Programmiergerät

Im Projektverlauf wurde der Bootloader meist mit einem Programmiergerät aktualisiert, in unserem Fall mit einem Atmel JTAGICE mkII und AVR32 Studio, der dafür erforderlichen Software (http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=4116).

Abbildung 6.7.2-1 AVR32 Studio

Nach dem Starten muss ein neues Target eingerichtet werden.

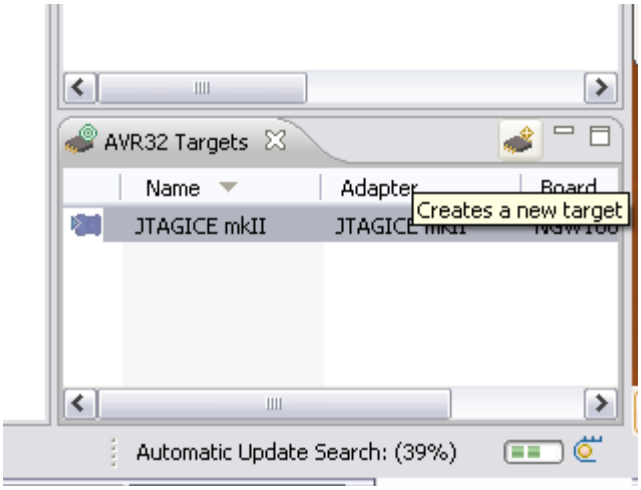


Abbildung 6.7.2-2 AVR32 Studio Targets

Hier muss das Programmiergerät (in diesem Fall das JTAGICE mkII das mit dem PC und der JTAG Schnittstelle des NGW100 verbunden ist) ausgewählt werden.

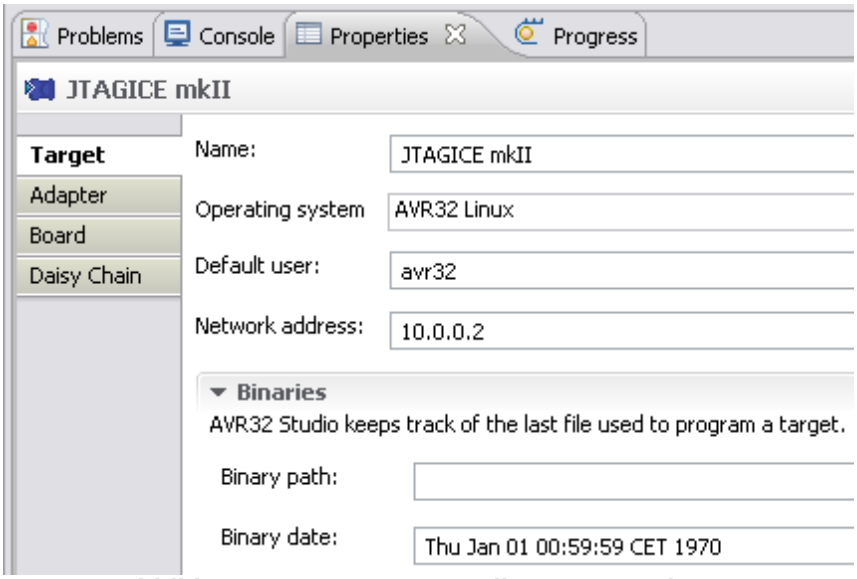


Abbildung 6.7.2-3 AVR32 Studio Target Optionen

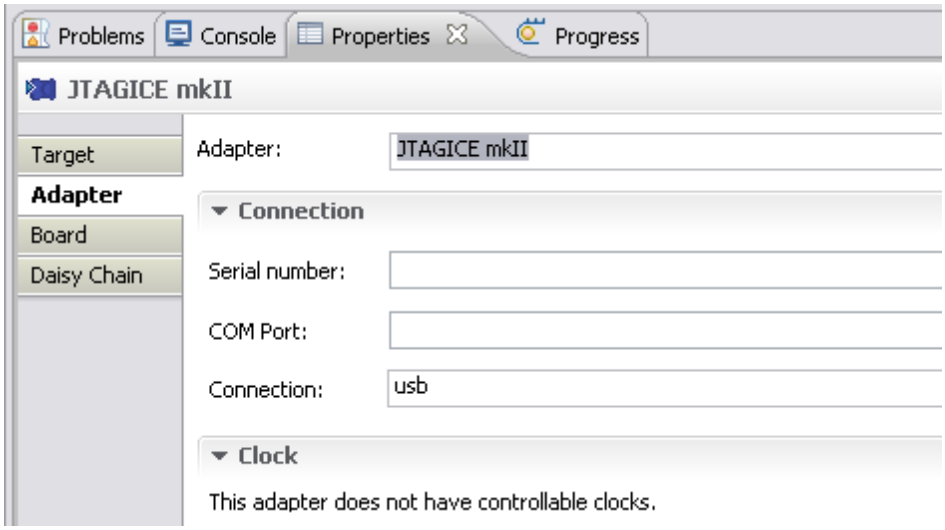


Abbildung 6.7.2-4 AVR32 Studio Adapter Optionen

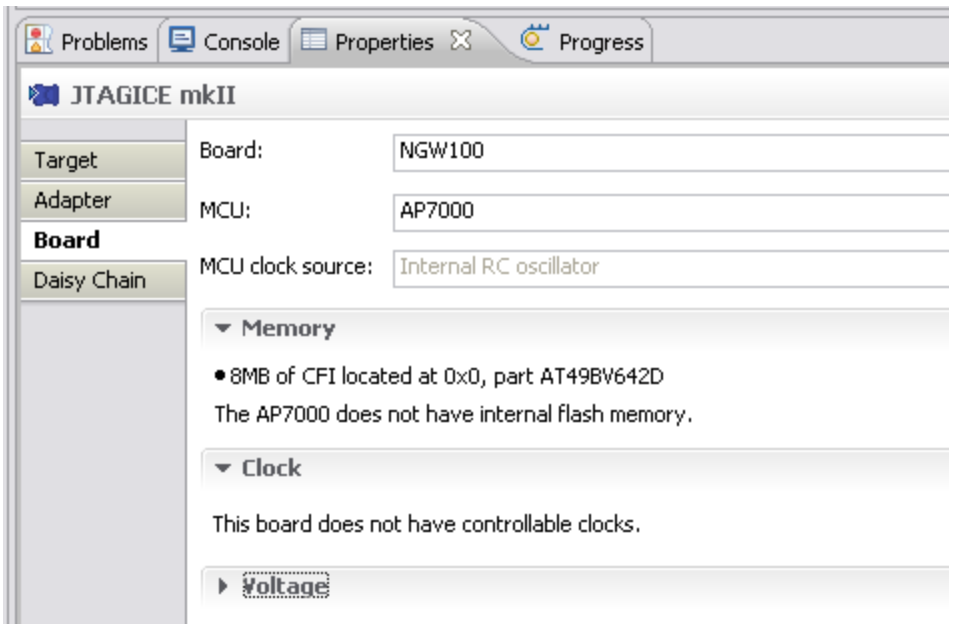


Abbildung 6.7.2-5 AVR32 Studio Board Optionen

Nach dem Einstellen der Target Optionen kann mit einem Rechtklick auf das Target und einem Klick auf „Program“ der Bootloader aktualisiert werden.

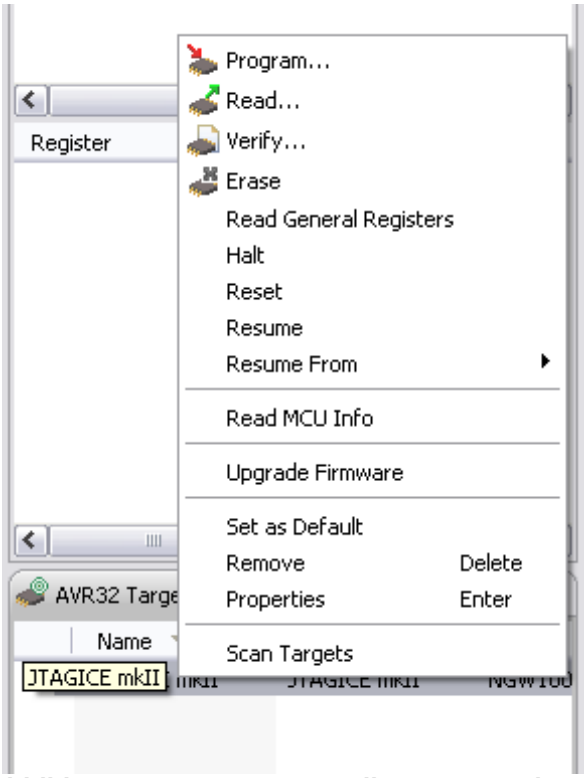


Abbildung 6.7.2-6 AVR32 Studio Programmieren

Nun muss noch der kompilierte Bootloader (/xmedia/buildroot2_3/project_build_avr32/xMedia/u-boot-1.3.4/uboot.bin) von Buildroot auf dem Linux Entwicklungssystem auf den Host kopiert werden (aufgrund von Problemen mit dem USB Treiber in der Virtual Box) und dann mit folgenden Einstellungen auf das Target – das NGW100 – übertragen werden.

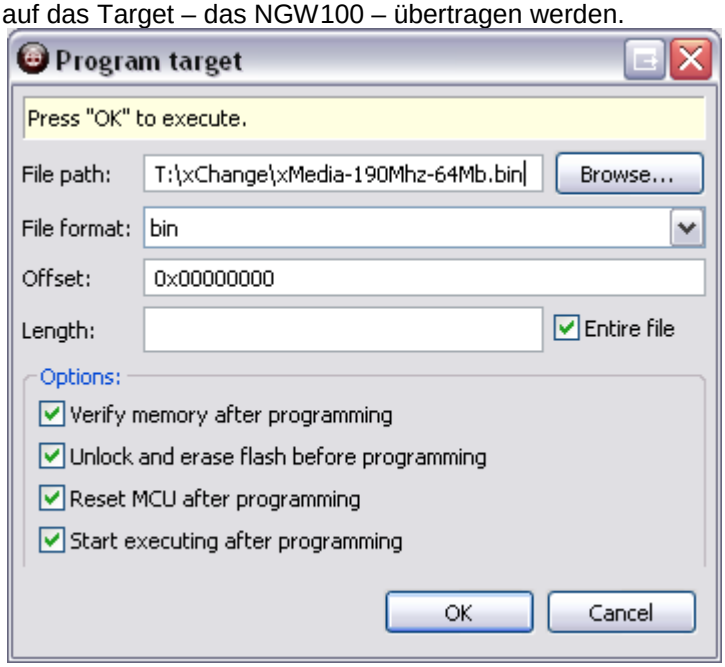


Abbildung 6.7.2-7 AVR32 Studio Programmieren

Nach dem Programmieren des neuen Bootloaders wird das Board automatisch neu gestartet. Nach jeder Änderung am Bootloader muss das Linux Kernel ebenfalls wieder übertragen werden!

Eine detaillierte Beschreibung des Update Prozesses ist auf der Dokumentationsseite http://www.avrfreaks.net/wiki/index.php/Documentation:NGW/Firmware_upgrade zu finden.

6.8. Modifizieren der Linux Kernel

Die Treiber für alle an den AP7000 angebunden Baugruppen sind einer der Knackpunkte des Systems. Der Prozessorhersteller Atmel, gibt vollen Linux Support vor, doch die Treiber für die doch sehr, sehr junge AP-Prozessorserie sind noch nicht ausgereift und weisen teilweise noch schwerwiegende Fehler auf. Darum gestaltete es sich als eines der schwierigsten Unterfangen, alle Treiber lauffähig zu bekommen.

Die folgenden Schritte, die hier in wenigen Seiten zusammengefasst sind, brauchten im Projektverlauf am meisten Zeit, da das Einarbeiten in den Kernel, das finden bzw. schreiben der Patches und das Testen ein sehr schwieriges Unterfangen ist.

6.8.1. Updaten und Konfigurieren des Linux Kernel

Nach dem ersten Kompilieren von Buildroot 2.3 stellt sich heraus, dass Linux 2.6.27 verwendet wird. Auf Kompatibilitätsgründen wird jedoch Linux in der Version 2.6.29 benötigt (verwendet wurde genau 2.6.29-rc3). Dazu muss zunächst der aktualisierte Linux Kernel von http://www.kernel.org (http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.29.tar.gz) heruntergeladen werden und anschließend in den Ordner „/xmedia/buildroot2_3/project_build/xmedia/linux2.6.27“ kopiert werden (es sollten zuvor alle Ordner – nicht die Dateien – in diesem Ordner gelöscht werden).

Danach muss die neue Konfiguration des Kernels noch eingespielt werden, dies geschieht wiederum durch überschreiben der „.config“ Datei – in diesem Fall der Datei im Ordner „/xmedia/buildroot2_3/project_build_avr32/xMedia/linux-2.6.27.6/“ mit der Datei vom beiliegenden Datenträger („linux/linux 2.6.29-rc3“)

6.8.2. Installieren des xMedia Board Supports

Nach dem Aktualisieren auf die Kernel Version 2.6.29 müssen noch, die für das richtige Setup des xMedia Board benötigten Dateien von dem beiliegenden Datenträger (Ordner „/linux/board/“) nach „/xmedia/buildroot2_3/project_build/xmedia/linux-2.6.27.6/arch/avr32/boards/“ kopiert werden.

Anschließend müssen die Dateien „Kconfig“ und „Makefile“ editiert werden.

Hier müssen folgende Einträge erstellt werden:

/xmedia/buildroot2_3/project_build/xmedia/linux-2.6.27.6/arch/avr32/Kconfig

```
…
config BOARD_FAVR_32
    bool "Favr-32 LCD-board"
    select CPU_AT32AP7000

config BOARD_MIMC200
    bool "MIMC200 CPU board"
    select CPU_AT32AP7000

config BOARD_XMEDIA
    bool "xMedia Board"
    select CPU_AT32AP7000
endchoice

source "arch/avr32/boards/atstk1000/Kconfig"
source "arch/avr32/boards/atngw100/Kconfig"
source "arch/avr32/boards/hammerhead/Kconfig"
```

```
source "arch/avr32/boards/favr-32/Kconfig"

choice
    prompt "Boot loader type"
    default LOADER_U_BOOT

config        LOADER_U_BOOT
    bool "U-Boot (or similar) bootloader"
endchoice
...
```

/xmedia/buildroot2_3/project_build/xmedia/linux-2.6.27.6/arch/avr32/Makefile

```
…
core-$(CONFIG_BOARD_ATSTK1000)          += arch/avr32/boards/atstk1000/
core-$(CONFIG_BOARD_ATNGW100)           += arch/avr32/boards/atngw100/
core-$(CONFIG_BOARD_HAMMERHEAD)         += arch/avr32/boards/hammerhead/
core-$(CONFIG_BOARD_XMEDIA)             += arch/avr32/boards/xmedia/
core-$(CONFIG_BOARD_FAVR_32)            += arch/avr32/boards/favr-32/
core-$(CONFIG_BOARD_MIMC200)            += arch/avr32/boards/mimc200/
core-$(CONFIG_LOADER_U_BOOT)            += arch/avr32/boot/u-boot/
…
```

6.8.3. Fehlerbehebung im CIFS Modul

Da der CIFS Treiber, der für das Einbinden von Netzwerkfreigaben zuständig ist, nicht vollständig mit der Prozessorarchitektur kompatibel ist, muss dieser Fehler vor dem kompilieren des Kernels behoben werden. Hierzu hat John Voltz, ein Mitglied er AVR32 Community ein Patch erstellt, welches angewendete werden muss.

Der Patchsatz ist auf dem Datenträger unter „linux/patches/cifs/“ zu finden und wird über folgendes Kommando angewendet:

```
cd /xmedia/buildroot2_3/project_build_avr32/xMedia/linux-2.6.27.6/
patch -p0 < /xchange/Files/linux/patches/cifs/cifs_unaligned.patch
patch -p0 < /xchange/Files/linux/patches/cifs/cifs_unaligned2.patch
```

6.8.4. Fehlerbehebung im Libertas SDIO Treiber

Der Libertas SDIO Treiber ist der Treiber für das verwendete Marvell 88W868 Wireless Modul. Wie beim CIFS Modul gibt es auch hier Kompatibilitätsprobleme mit dem AVR32 Prozessor. Colin Mc Gabe, ein Mitentwickler am Libertas SDIO Treiber hat hierfür bereits ein Patch geschrieben, welches wiederum nur angewendet werden muss.

Der Patchsatz ist auf dem Datenträger unter „linux/patches/wifi/“ zu finden und wird über folgendes Kommando angewendet:

```
cd /xmedia/buildroot2_3/project_build_avr32/xMedia/linux-2.6.27.6/
patch -p1 < /xchange/Files/linux/patches/wifi/wifi.patch
```

6.8.5. Fehlerbeheben im Atmel MCI Treiber

Aufgrund eines noch nicht näher spezifiziertem Fehler, der im MCI (MultMedia Card Interface) Treiber des Prozessors entdeckt wurde, wird das SDIO Wireless Modul nicht richtig initialisiert. Nach langer erfolgloser Absprache mit den Entwicklern des Treibers,

wurde per Zufall eine Lösung gefunden. Der Grund, warum der Treiber funktioniert bleibt ungewiss, jedoch wird vermutet, dass die eingefügte Textausgabe eine gewisse Zeit benötigt, die das Wireless Modul für den Setup benötigt.

Außerdem wurde ein zweiter Fehler entdeckt im Treiber entdeckt der vom Projektteam behoben wurde. Der Fehler wurde an die Kernel Mailing Listen weitergeleitet und die vorgeschlagene Änderung wurde in die folgenden Kernel Versionen aufgenommen.

```
--- a/drivers/mmc/host/atmel-mci.c  2009-01-28 19:49:30.000000000 +0100
+++ b/drivers/mmc/host/atmel-mci.c          2009-05-25 19:54:02.000000000 +0200
@@ -812,7 +812,7 @@ static void atmci_set_ios(struct mmc_hos
                slot->sdc_reg |= MCI_SDCBUS_1BIT;
                break;
        case MMC_BUS_WIDTH_4:
-               slot->sdc_reg = MCI_SDCBUS_4BIT;
+               slot->sdc_reg |= MCI_SDCBUS_4BIT;
                break;
        }

@@ -838,6 +838,9 @@ static void atmci_set_ios(struct mmc_hos
                                clock_min = host->slot[i]->clock;
                }

+               /* Print clock */
+               dev_warn(&mmc->class_dev, "Changing clock speed to %u\n", clock_min);
+
                /* Calculate clock divider */
                clkdiv = DIV_ROUND_UP(host->bus_hz, 2 * clock_min) - 1;
                if (clkdiv > 255) {
```

Das Anwenden des im Ordner „linux/patches/wifi“ beiliegenden Patches gestaltet sich wieder recht einfach.

```
cd /xmedia/buildroot2_3/project_build_avr32/xMedia/linux-2.6.27.6/
patch -p1 < /xchange/Files/linux/patches/wifi/mci.patch
```

6.8.6. Audio Treiber verwendbar machen

Da während der Entwicklung des xMedia Player der AC97 Treiber vom Prozessorhersteller geändert wurde und in der Version 2.6.29 nicht funktionsfähig ist, müssen noch einige Patches angewendet werden, um diesen wieder funktionsfähig zu bekommen.

Der Patchsatz ist auf dem Datenträger unter „linux/patches/sound/“ zu finden und wird über folgendes Kommando angewendet:

```
patch -p1 < /xchange/Files/linux/patches/sound/avr32-dw.patch

patch -p1 < /xchange/Files/linux/patches/sound/avr32-sound-1.patch
patch -p1 < /xchange/Files/linux/patches/sound/avr32-sound-2.patch
patch -p1 < /xchange/Files/linux/patches/sound/avr32-sound-3.patch
patch -p1 < /xchange/Files/linux/patches/sound/avr32-sound-4.patch
patch -p1 < /xchange/Files/linux/patches/sound/avr32-sound-5.patch
patch -p1 < /xchange/Files/linux/patches/sound/avr32-sound-6.patch
patch -p1 < /xchange/Files/linux/patches/sound/avr32-sound-7.patch
```

6.8.7. Kompilieren & Konfigurieren des Kernels

Der Linux Kernel, der in Buildroot integriert ist, ist nicht optimal konfiguriert und muss daher häufig manuell konfiguriert werden. Durch das Laden der „.config“ Datei mit der Projektbezogenen Konfiguration wurde der Kernel bereits neu konfiguriert.

Über den Befehl:

```
cd /xmedia/buildroot2_3/project_build_avr32/xMedia/linux-2.6.27.6
make ARCH=avr32 CROSS_COMPILE=avr32-linux- menuconfig
```

Danach öffnet sich ein Menü wo alle Konfigurationseinstellungen getroffen werden können – die Navigation erfolgt über die Cursortasten. Nach dem Schließen des Menüs wird die Konfigurationsdatei erneut geschrieben bzw. aktualisiert.

Um nun sicherzustellen, dass wirklich alle Teile des Kernels neu kompiliert werden, muss noch ein „clean“ durchgeführt werden, d.h. alle temporären Dateien müssen gelöscht werden.

```
cd /xmedia/buildroot2_3/project_build_avr32/xMedia/linux-2.6.27.6
make ARCH=avr32 CROSS_COMPILE=avr32-linux- clean
```

Nach dem “clean” muss das Kernel neu erstellt werden, die einfachste Methode, dies zu tun, ist das komplette Buildroot Packet neu zu erstellen, dies dauert – sofern alles bereits einmal kompiliert wurde – nicht länger wie das normale kompilieren des Kernels, generiert jedoch ein vollständiges Root – Dateisystem.

```
cd /xmedia/buildroot2_3/
make
```

Nach dem Kompilieren wird ein komplettes Archiv mit dem Root-Dateisystem erstellt, welches den Kernel und alle Module beinhaltet.

6.9. Fehlerbehebung in QTEmbedded

Bereits beim ersten Kompilieren von Buildroot wurde eine Fehler in der Webkit Engine, die in QTEmbedded integriert ist, behoben, doch es gibt noch zwei Fehler die die Nutzung von QT Einschränken welche behoben werden müssen.

6.9.1. Webkit JavaScript Bug beheben

Ein Fehler, wiederum in der Browser Engine Webkit, verhindert das korrekte Ausführen von Javascripts, folgendes Patch behebt dieses Problem:

```
cd /xmedia/buildroot2_3/build_avr32/qt-embedded-linux-opensource-src-4.4.3/
patch -p1 < /xchange/Files/linux/patches/webkit/javascript.patch
```

6.9.2. Farbfehler beheben

Auch im Grafiktreiber (der direkt auf dem Linux Framebuffer aufsetzt) wurde ein Fehler gefunden. Da der Bildschrim am xMedia Player verkehrt herum montiert ist, muss das Bild per Software rotiert werden. Hierzu wird ein spezieller Treiber verwendetet (Transformed:rot180:0). Nun erfolgt jedoch die Erkennung des Farbreihenfolge der Bytes im Linux Framebuffer falsch (BGR – RGB wird vertauscht), so sind die Farben die Farben Rot und Blau vertauscht.

Hier wurde der Fehler nicht behoben, sondern nur Quick'n Dirrty Umgangen, in dem den Farbdaten während des Rotationsvorganges neu zusammengesetzt werden.

```
cd /xmedia/buildroot2_3/build_avr32/qt-embedded-linux-opensource-src-4.4.3/
patch -p1 < /xchange/Files/linux/patches/qt/color.patch
```

6.9.3. Neu Kompilieren

Gleich wie beim Linux Kernel muss zum Erneuten kompilieren erst ein "clean" von QT durchgeführt werden.

```
cd /xmedia/buildroot2_3/build_avr32/qt-embedded-linux-opensource-src-4.4.3/
make clean
```

Nun muss noch die Datei „.compiled" im QT Build-Verzeichnis (/xmedia/buildroot2_3/build_avr32/qt-embedded-linux-opensource-src-4.4.3/) gelöscht werden, nur so wird QT später wirklich neu kompiliert.

Danach ist der einfachste Weg einfach das komplette Buildroot Packet neu zu kompilieren:

```
rm /xmedia/buildroot2_3/build_avr32/qt-embedded-linux-opensource-src-4.4.3/.compiled
cd /xmedia/buildroot2_3/
make
```

Das Kompilieren von QT ist sehr, sehr zeitaufwändig – Änderungen sollten daher nur sehr bedacht vorgenommen werden.

6.10. Fehlerbehebung in MPlayer

Außerdem wurde ein Fehler in dem verwendeten Open Source Musik Player MPlayer gefunden, welcher durch Anwenden folgenden Patches behoben wird:

```
cd /xmedia/buildroot2_3/build_avr32/MPlayer-1.0rc1/
patch -p1 < /xchange/Files/linux/patches/mplayer/vo.patch
```

6.10.1. Neu Kompilieren

Gleich wie beim Linux Kernel muss zum Erneuten kompilieren erst ein “clean” von MPlayer durchgeführt werden.

```
cd /xmedia/buildroot2_3/build_avr32/MPlayer-1.0rc1/
make clean
```

Nun müssen noch die Ausführbaren Dateien im MPlayer Ordner gelöscht werden (die Datei mit dem Namen „mplayer“), damit Buildroot das MPlayer neu kompiliert und installiert.

Danach ist der einfachste Weg einfach das komplette Buildroot Packet neu zu kompilieren:

```
rm /xmedia/buildroot2_3/build_avr32/MPlayer-1.0rc1/mplayer
cd /xmedia/buildroot2_3/
make
```

6.11. Installation auf der Hardware

Das Betriebssystem auf dem Player ist auf zwei Bereiche verteilt, einerseits der Kernel – der mittels uBoot in den internen Flash kopiert wird und andererseits das Dateisystem, dass sich auf der SD Karte befindet.

6.11.1. Formatieren der SD Karte

Die SD Karte für das Betriebsystem muss zunächst partitioniert werden, hier wurde folgendes Layout gewählt:

Größe	Dateisystem	Mountpoint auf der Hardware
100 MB	EXT2	/
128 MB	SWAP	swap
*	FAT32	/home/xmedia/Files/Internal

6.11.2. Der FS Patch Ordner

Nach dem kompilieren des Buildroot Paketes, erhält man ein Archiv mit dem Root-Dateisystem des xMedia Player. Dieses wäre zwar lauffähig aber ist bei weitem nicht optimal konfiguriert.

Darum sollte der fspatch Ordner in „/xmedia/“ erstellt werden. Dort werden nun alle Änderungen Verglichen zum Standard Dateisystem hineinkopiert. D.h. wenn zum Beispiel die Datei „/etc/fstab“ geändert werden soll muss im „/xmedia/fspatch/“ nur die geänderte Datei gespeichert werden.

Der Inhalt des fspatch Ordners ist wiederum dem beiliegenden Datenträger in dem Ordner „xmedia/“ verfügbar.

6.11.3. Der Internal Ordner

Unter „/xmedia/internal“ sollten die Dateien gespeichert werden, die später über den CopyFs Skript auf den internen Speicher (die 3. Partition) kopiert werden soll.

6.11.4. Der CopyFs Skript

Um alle Daten (Dateisystem, fspatch, Software, Webinterface) schnell und einfach auf die SD Karte entpacken und kopieren zu können wurde der CopyFS Skript erstellt, der unter „linux\buildroot2_3“ zu finden ist.

Der Skript erwartet, dass nach dem Einlegen einer SD Karte in das Kartenlesegerät zwei neue Laufwerke erscheinen: „/media/disk“ und „/media/disk-1“.

Nach dem kopieren des Skripts in das „/xmedia/buildroot2_3/“ Verzeichnis und dem Aufrufen mittels:

```
./copyfs.sh
```

Wird zunächst abgefragt ob wirklich alle Dateien gelöscht werden sollen, nach der Eingabe irgendeines Zeichens und dem Drücken der Eingabetaste, werden alle Dateien automatisch kopiert und installiert.

Nach dem Abschluss des Kopiervorgangs wird außerdem der Bootloader und das Root Dateisystem in den „/xchange/“ Ordner kopiert.

Der CopyFS Skript sieht wie folgt aus:

```
cd /media/disk/
echo Remove old data
read x
sudo rm -R ./*
echo Unpack tar

sudo tar xfv /xmedia/buildroot2_3/binaries/xMedia/rootfs.avr32.tar

echo Chmod and PatchFS
sudo chmod 0777 -R ./
sudo cp /xmedia/fspatch/* ./ -R -f

echo Copying xMedia
sudo cp /xmedia/Software/xMedia/xMedia ./home/xMedia/
sudo chmod 0777 -R ./

echo Copying xWeb
sudo cp /xmedia/Software/xWeb/xWeb ./www/cgi-bin -f
sudo cp /xmedia/Software/xWeb/xweb/ ./www/ -R -f

echo Copying Internal Data
sudo cp /xmedia/internal/* /media/disk-1/ -R -f
sudo chmod 0777 -R /media/disk-1/

echo Save binaries
date=`date +"%Y-%m-%d_%H-%M-%S"`
sudo mkdir "/xchange/xmedia-$date/"
sudo cp /xmedia/buildroot2_3/binaries/xMedia/rootfs.avr32.tar "/xchange/xmedia-$date/"
sudo cp /xmedia/buildroot2_3/binaries/xMedia/u-boot.bin "/xchange/xmedia-$date/"
echo Done
```

6.11.5. Kernel auf den Speicher des Players kopieren

Nach dem erfolgreichen Kopieren der des Dateisystems auf die SD Karte muss diese in den Player eingelegt werden, und der Player gestartet werden (Terminalkommunikation mittels seriellem Kabel – 115200 Baud). Im uBoot Menü muss nun der Startvorgang unterbrochen mittels drücken der Leertaste unterbrochen werden.

Mit folgendem Kommandosatz, wird nun das Kernel in den internen Speicher kopiert:

```
protect off 0x20000 0x7EFFFF;erase 0x20000 0x7EFFFF;mmcinit;ext2load mmc 0:1
0x90000000 /boot/uImage;cp.b 0x90000000 0x20000 0200000;
```

Der Kernel wird aus zwei Gründen in den internen Flash kopiert:

1. Der Wireless Treiber arbeitet nicht korrekt, wenn der Kernel auf der SD Karte liegt
2. Das System startet schneller

Nun müssen noch einige Startparameter geändert werden:

```
setenv 'bootcmd' ' bootm 0x20000'
setenv 'bootargs' 'console=ttyS0 root=/dev/mmcblk0p1 rootwait
fbmem=600k@0x10500000'
```

Nach dem Erneuten Anstecken, des Players wird der neue Kernel gebootet und das Root Dateisystem von der SD Karte geladen.

Der xMedia Player sollte nun wie gewünscht in eine Linux Umgebung starten.

7. Die Software

Wie erwähnt wurde zur Entwicklung der Software des Player die Programmiersprache C++ gewählt. Es wurde jedoch mit der QT Bibliothek im Hintergrund gearbeitet, welche beinahe den kompletten Charakter der veralteten Hochsprache C++ ändert und sie zu einer einfach zu benutzenden, modernen Sprache macht.

7.1. QT Creator – Die Entwicklungsumgebung

Um das voll grafische Potential von QT nutzen zu können wurde mit einer Vorabversion des QT Creators gearbeitet, der Version 0.9.1-beta-linux-x86.
Die neue finale Version des QT Creators mit der QT Bibliothek kann von http://www.qtsoftware.com/downloads/sdk-linux-x11-32bit-cpp gratis heruntergeladen werden.

7.1.1. Installieren

Nach dem Download (z.B. auf den Desktop) müssen zunächst die Rechte der herunter geladenen Datei geändert werden

```
sudo chmod 0777 ./Desktop/qt-sdk-linux-x86-opensource-2009.02.bin
```

Und der Installer anschließend gestartet werden:

```
sudo ./Desktop/qt-sdk-linux-x86-opensource-2009.02.bin
```

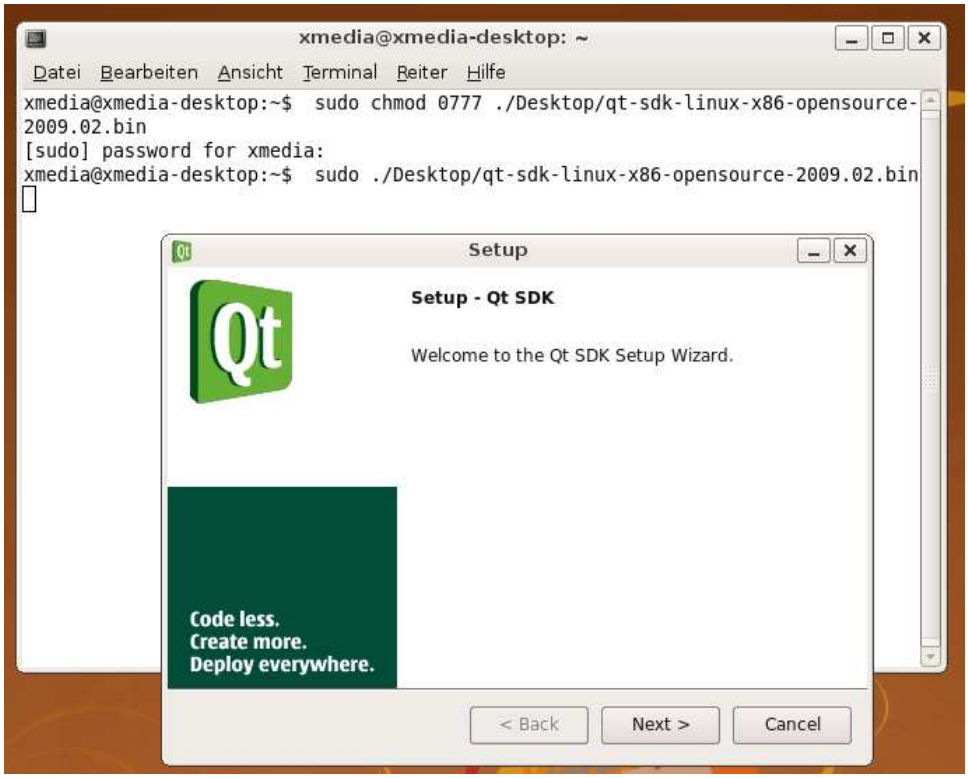


Abbildung 7.1.1-1 QT Installation

Anschließend führt ein einfach aufgebauter Installer durch die komplette Installation. Der gesamte Prozess dauert einige Zeit, da die QT Bibliothek vom Umfang immens ist.

Nach der Installation erscheint der QT Creator Icon am Desktop.

Abbildung 7.1.1-2 QT Creator

Nach dem Klick auf den Icon startet nun das QT Creator IDE und muss nun noch für den Einsatz im Embedded Bereich konfiguriert werden.

7.1.2. Konfigurieren

Damit die Programme die nun mit QT Creator erstellt werden auch auf der AVR32 Architektur lauffähig sind müssen sie Cross-Compiliert werden, d.h. es muss noch eingestellt werden, das QT einen anderen Compiler bzw. eine andere QT Version verwenden soll.

Die geschieht in QT Creator unter „Tools“ → „Options“.

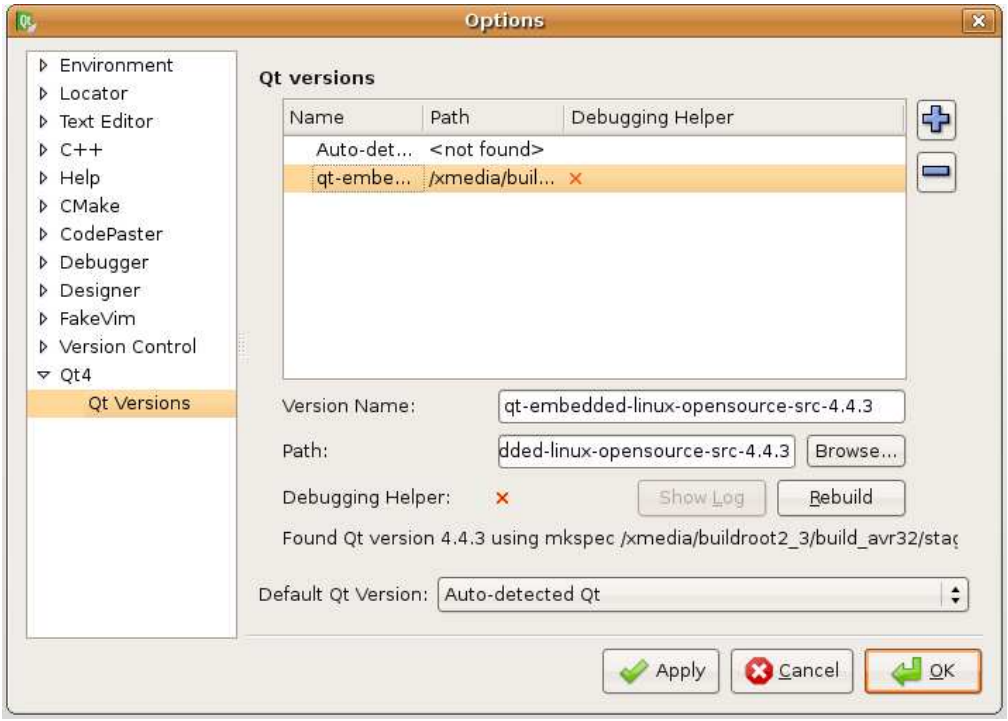


Abbildung 7.1.2-1 QT Optionen

Hier muss nun eine neue QT Version hinzugefügt werden – für den Embedded Bereich die in Buildroot bereits kompilierte Version von QT Embedded (der Pfad ist /xmedia/buildroot2_3/build_avr32/qt-embedded-linux-opensource-src-4.4.3)

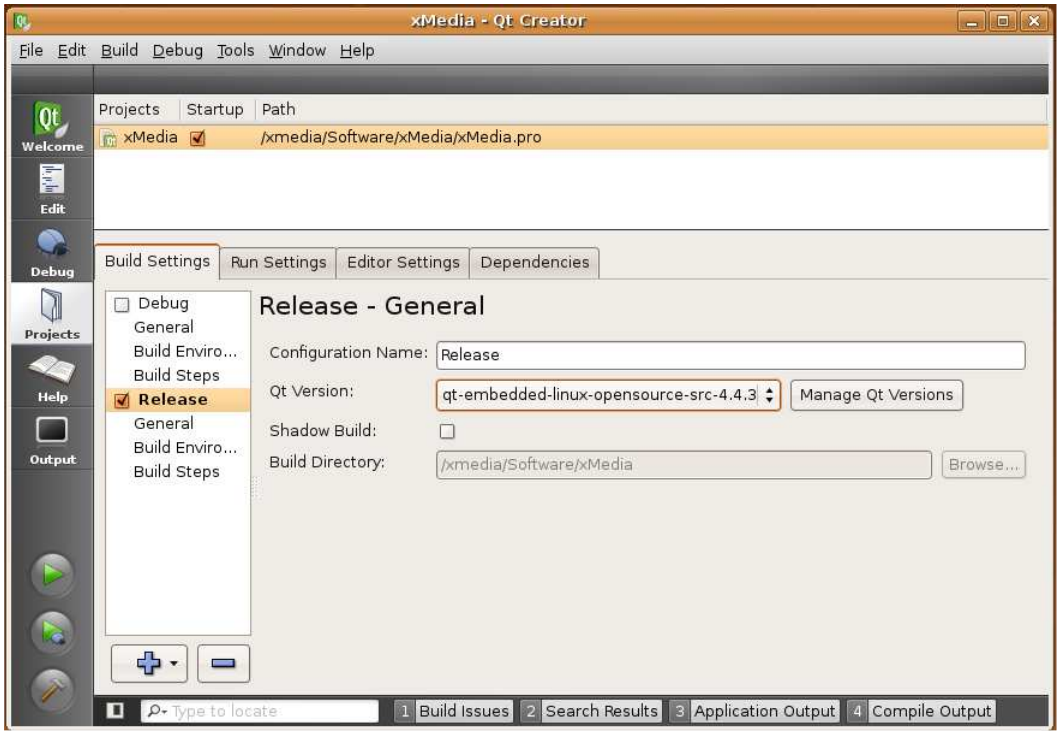


Abbildung 7.1.2-2 QT Version auswählen

Nun kann bei jedem Projekt (hier wurde das xMedia Projekt geöffnet) die QT Version definiert werden, für die die Software erstellt werden soll.

Somit ist es einfach, Software zum Testen für das Linux Entwicklungssystem zu kompilieren und mit einer Einstellung sofort für das Target AVR32 System.

7.1.3. Laden der Projekte

Bestehende Projekte, wie das xMedia und das xWeb Projekt können durch einen simplen Klick auf die „.pro“ Datei geöffnet werden. Die xMedia und xWeb Quellcodes liegen am beiliegenden Datenträger im Ordner „software/“ bei.

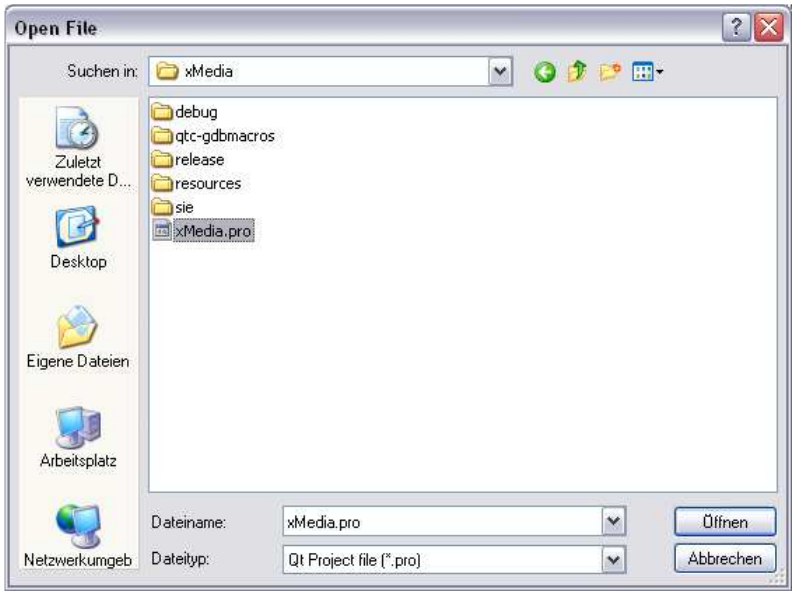


Abbildung 7.1.3-1 QT Creator Projekt öffnen (Windows)

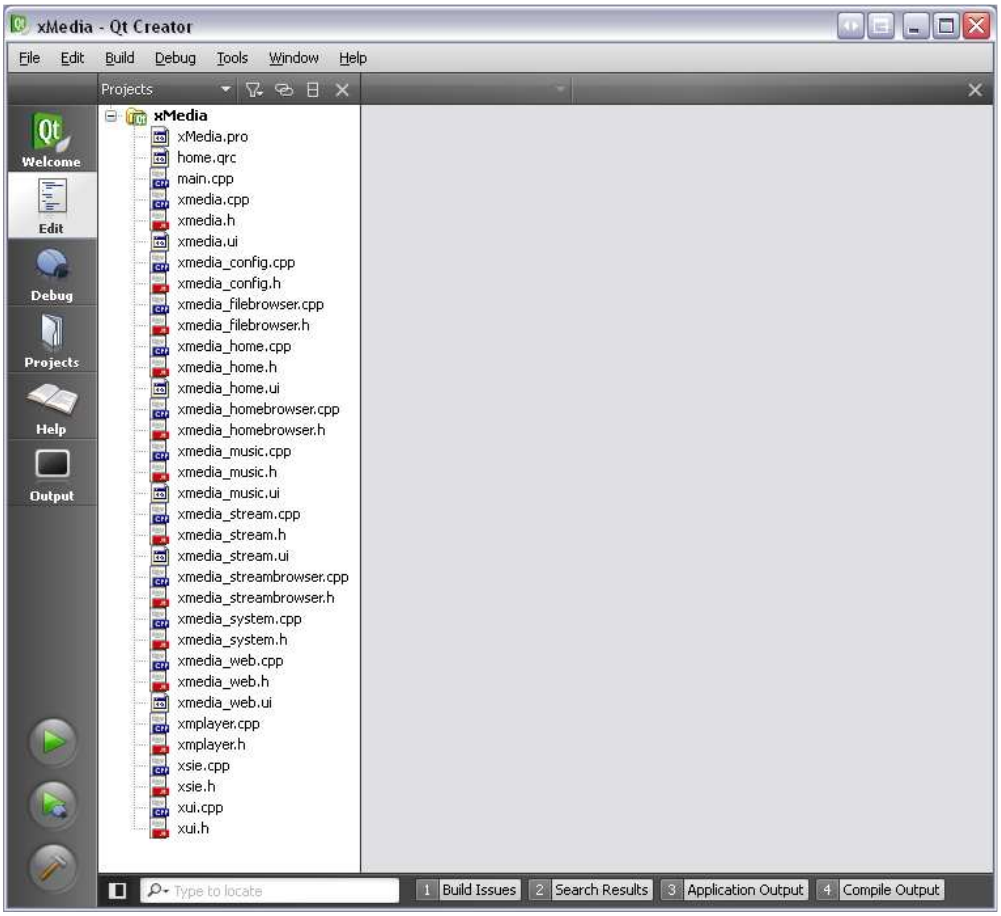


Abbildung 7.1.3-2 QT Creator geöffnetes Projekt(Windows)

7.2. Die xMedia Software

Die xMedia Software wurde wie erwähnt mit C++ und QT erstellt, hier war vor allem das Signal – Slot Konzept von QT sehr nützlich und es konnte wie auch schon in der Firmware somit Event gesteuert gearbeitet werden.

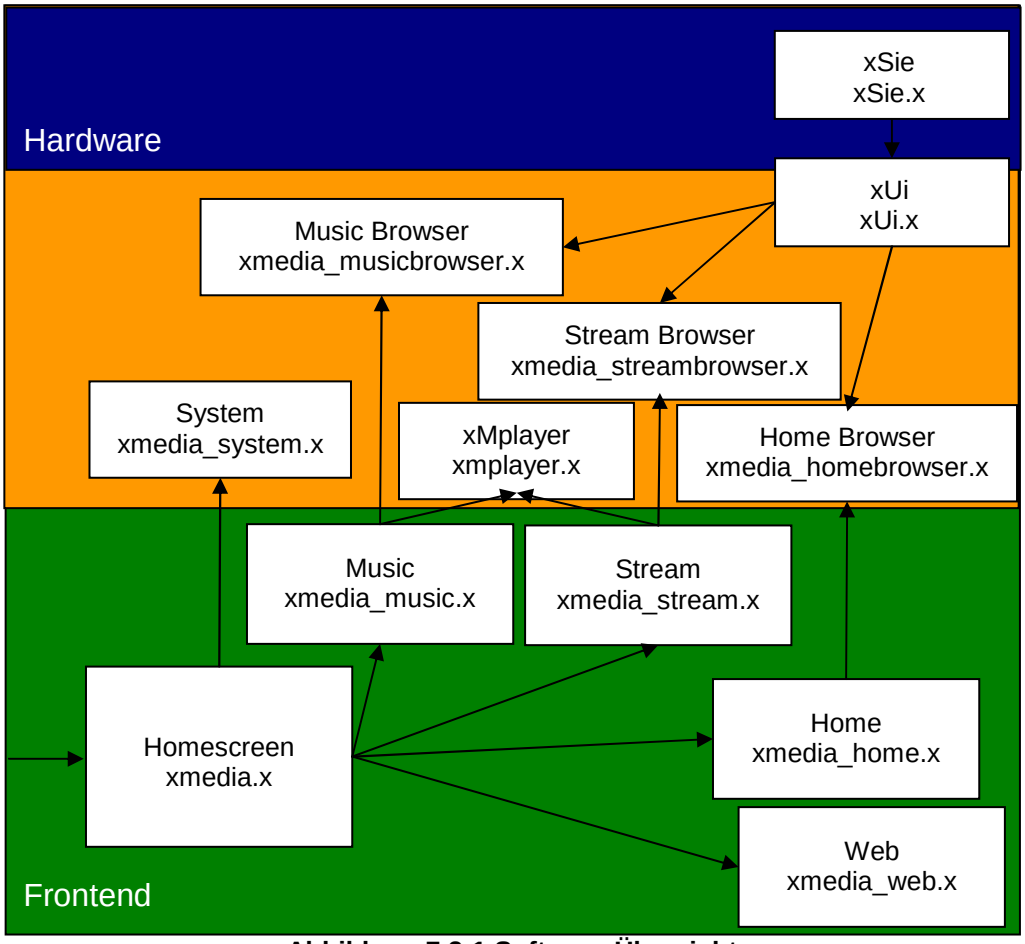


Abbildung 7.2-1 Software Übersicht

7.2.1. Die xSIE Klasse

Die Bereits in der Firmware implementierte xSie wurde nun von C auf C++/Qt portiert und der Treiber für die serielle Schnittstelle wurde gleich eingebunden.

Aufgrund der fehlenden plattformunabhängigen Implementation eines Treibers für die serielle Schnittstellte wurde eine auf Linux basierende Lösung verwendet, welche das Testen auf Windows basierten Systemen erschwert.

```
#define XSIE_USE_DEV "/dev/ttyS1"
#define XSIE_USE_BAUD B38400
```

Die Einstellungen der seriellen Schnittstelle können in der Datei „xsie.h“ vorgenommen werden.

Um mit der Firmware Implementation kompatibel zu bleiben, wurde die Packet Struktur gleich gelassen.

```
typedef struct
{
    unsigned int pid;
    unsigned char command;
    unsigned char length;
    unsigned char data[XSIE_MAX_PACKET_SIZE];
    unsigned char crc;
} xsiePacket;
```

Hier trifft man wiederum die 5 Datenfelder pid, command, length, data und crc an.

Der CRC Check wurde aufgrund der bereits fehlenden Implementation (aus Performancegründen) in der Firmware auch in dieser Implementation nicht integriert.

Der Aufbau der xSie ist wie in der Firmware wieder State-Machine basiert (vor allem der Empfangsalgorithmus mit dem SLIP Verfahren).

7.2.1.1. Funktionen

```
void xSie::submit(unsigned char command, unsigned char length, unsigned char *data);

Diese Funktion sendet ein siePacket  über die SIE.
command          … Befehl der gesendet werden soll
length           … Länge der Nutzdaten
data[length]     … Nutzdaten (Array mit der Länge length)
```

```
void xSie::respond(unsigned char pid, unsigned char command, unsigned char length,
unsigned char *data);

Diese Funktion sendete eine Antwort auf ein empfangenes SIE Paket
Pid              … ID des Paketes auf das geantwortet werden soll
command          … Befehl der gesendet werden soll
length           … Länge der Nutzdaten
data[length]     … Nutzdaten (Array mit der Länge length)
```

```
void xSie::submitPacket(xsiePacket data);

data             … xSie Rohdatenpaket, mit den Daten die gesendet werden sollen
```

7.2.1.2. Signals

Eine gute Anwendung für das Signal-Slot Konzept von QT ist das Empfangen der Daten über von der xSie. Die Anwendung die die Daten von der xSie empfangen will muss nur seinen „slot“ mit dem „signal“ der xSie Klasse verbinden („connect“). Ruft nun die xSie Klasse das Signal auf, so werden die übergebenen Daten automatisch an die verbundene „signal“ Funktion weitergeleitet.

Folgende Slots stehen zur Verfügung:

void **xSie::packetReceived**(xsiePacket);

Dieses Signal wird aufgerufen, wenn ein Paket erfolgreich empfangen wurde.
xSiePacket … Empfangenes Paket als in eine xSiePacket Struktur gepackt

void **xSie::dataError**(xsieError);

Dieses Signal feuert, wenn es einen Fehler beim Empfangen eines Paketes gab.
xsieError … Fehlercode (
XSIE_ILLEGAL_FRAMESTART,
XSIE_ILLEGAL_ESCAPE_SEQUENCE,
XSIE_CRC_CHECK_FAILED)

7.2.2. Die xUi Klasse

Die xUI Klasse hat die Aufgabe alle einkommenden Datenpakete von der xSie zu empfangen und zu verarbeiten bzw. weiterzuschicken. Die Klasse ist wieder nach dem Signal Slot Konzept aufgebaut.

7.2.2.1. Funktionen

void **xui::sendCommand**(unsigned char command, char *data, unsigned char length);

Diese Funktion sendet ein Paket über die SIE.

command	… Befehl der gesendet werden soll
length	… Länge der Nutzdaten
data[length]	… Nutzdaten (Array mit der Länge length)

7.2.2.2. Signals

Die xUi Klasse gibt nun verglichen zur darunterliegenden xSie Klasse mehrere Signale aus, sie teilt die empfangenen Kommands anhand ihrer ID in mehrere Bereiche auf:

- Info Kommando
- Power Management Kommandos
- User Interface Kommandos
- Maus Emulations Kommandos

void **xui::commandInfoReceived**(unsigned char command, unsigned char length, unsigned char *data);
void **xui::commandPowerManagmentReceived**(unsigned char command, unsigned char length, unsigned char *data);
void **xui::commandUserInterfaceReceived**(unsigned char command, unsigned char length, unsigned char *data);
void **xui::commandMouseEmulationReceived**(unsigned char command, unsigned char length, unsigned char *data);

Signale für die jeweiligen Events, alle Signale sind gleich aufgebaut.

command	… Kommando das über die xSie gesendet worden ist
length	… Länge des Nutzdatenarrays
data	… Array mit den Nutzdaten

Außerdem feuert bei einem Übertragungsfehler das „transmissionError“ Signal.

void **xui::transmissionError**(xsieError);

Dieses Signal feuert, wenn es einen Fehler beim Empfangen eines Paketes gab.

xsieError	… Fehlercode (
	XSIE_ILLEGAL_FRAMESTART,
	XSIE_ILLEGAL_ESCAPE_SEQUENCE,
	XSIE_CRC_CHECK_FAILED)

7.2.3. Der MPlayer Klasse

Die xMplayer Klasse ist eine Klasse die einen Wrapper um den im System vorhandenen MPlayer bildet. Der MPlayer ist ein Open Source Produkt das automatisch mit Buildroot auf dem Zielsystem installiert wird.

Das MPlayer liegt als Konsolenanwendung im System vor und in QT wird nun ein neuer Prozess gestartet, der mit dem MPlayer interagiert.

7.2.3.1. Funktionen

Die Hauptfunktion der xMplayer Klasse ist die Steuerung der Musik / Video Wiedergabe.

void **xMplayer::play**(QString filename);

Startet die Wiedergabe einer beliebigen Multimedia Datei
filename … Dateiname / Pfad zur Multimedia Datei

void **xMplayer::setPlay**(int play = -1);

Startet bzw. stoppt die Wiedergabe der aktuellen Multimedia Datei.
play … play = 1 startet die Wiedergabe, play = 0 stoppt die Wiedergabe

xMPlayerProgress **xMplayer::getProgress**();

Gibt eine Struktur des Typs xMPlayerProgress zurück, die Informationen über den Fortschritt der Wiedergabe beinhaltet.

Aufbau der zurückgegeben Struktur:

```
typedef struct
{
        double secondsPlayed; //Wie viele Sekunden wurden bereits abgespielt
        double secondsTotal; //Wie lang ist die Datei (Sekunden)
        double percentPlayed; //Wie viel Prozent der Datei wurden bereits abgespielt
} xMPlayerProgress;
```

xMPlayerMediaType **xMplayer::getMediaType**();

Gibt den Typ der geladenen Multimedia Datei zurück:
XMPLAYER_AUDIO für Musik
XMPLAYER_VIDEO für Videos
XMPLAYER_UNKNOWN wenn der Typ nicht erkannt wurde

7.2.3.2. Signals

Um alle Ereignisse sofort und direkt an die GUI weiterleiten zu können, gibt es einige Singale die beim Ändern der Datei, beim Starten bzw. Stoppen der Wiedergabe oder beim Ändern des Fortschritts der Wiedergabe feuern.

void **xMplayer::fileChanged**(xMPlayerFileInfo);

Dieses Signal feuert wenn die Datei erfolgreich geändert worden ist, die übergebene Struktur hat folgende Form:

```
typedef struct
{
   QString filename;        //Dateiname
   QString artist;          //Interpret (ID3Tag)
   QString album;           //Album (ID3Tag)
   QString title;           //Titel (ID3Tag)
   QString year;            //Erscheinungsjahr (ID3Tag)
   QString comment;         //Kommentar (ID3Tag)
   QString track;           //Track Nummer (ID3Tag)
   QString genre;           //Genre (ID3Tag)
} xMPlayerFileInfo;
```

void **xMplayer::finished**(QString);

Dieses Signal feuert, wenn die Wiedergabe einer Datei beendet wurde.
QString … Dateiname der abgespielten Datei

void **xMplayer::progressChanged**(xMPlayerProgress);

Dieses Signal feuert, wenn sich der Fortschritt der Datei geändert hat (d.h. während der Wiedergabe der Datei), jedoch maximal 1mal pro Sekunde.

xMPlayerProgress …Aufbau der zurückgegeben Struktur:

```
typedef struct
{
        double secondsPlayed; //Wie viele Sekunden wurden bereits abgespielt
        double secondsTotal; //Wie lang ist die Datei (Sekunden)
        double percentPlayed; //Wie viel Prozent der Datei wurden bereits abgespielt
} xMPlayerProgress;
```

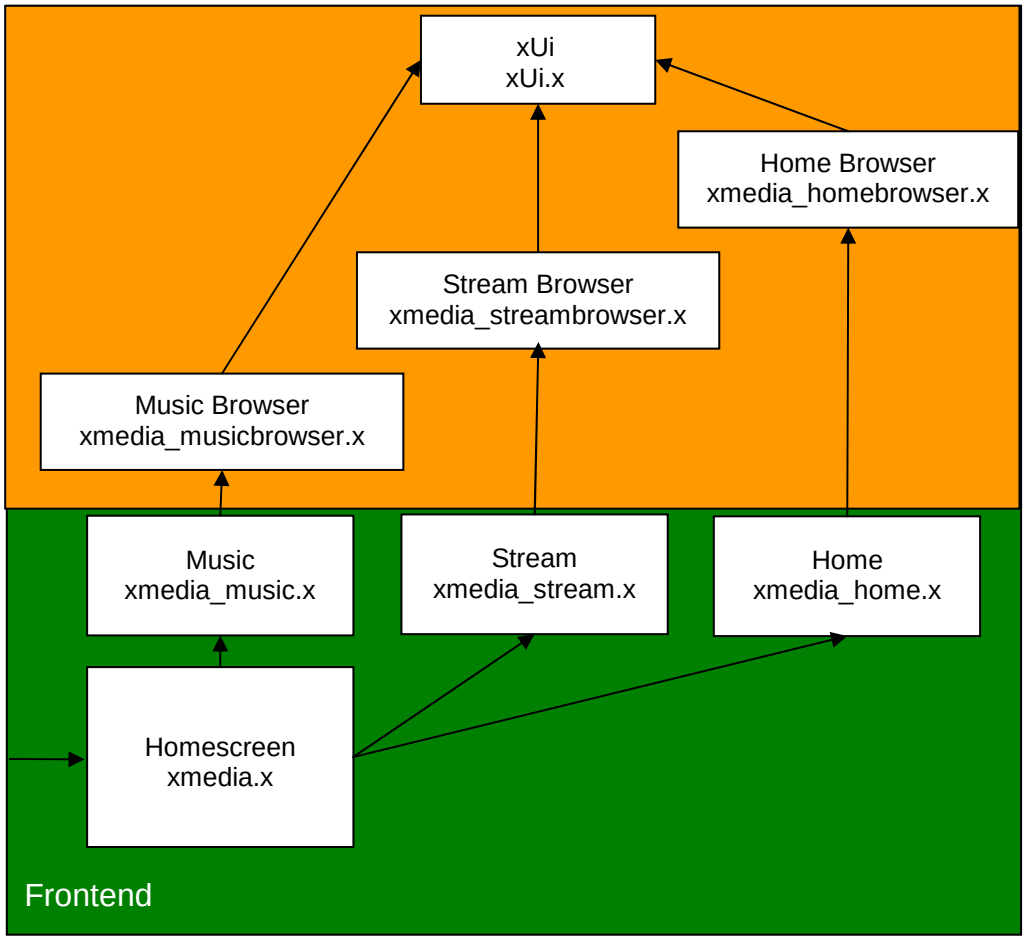
void **xMplayer::stateChanged**(bool);

Dieses Signal feuert, wenn die Wiedergabe gestartet oder gestoppt wird.
Bool … 1 wenn die Wiedergabe läuft, 0 wenn sie gestoppt wurde

7.2.4. Die Benutzeroberfläche

Der QT Creator bietet wirklich gute Möglichkeiten um eine ansprechende grafische Oberfläche zu erstellen. Mit einem Visuellen Editor, der denkbar einfach zu bedienen ist, lassen sich GUI Applikationen einfach per Drag and Drop erstellen und auf gleich ausführen.

Da als User Eingabe jedoch nicht die Maus bzw. die Tastatur verwendet wird, sondern die xUI Klasse, kann nicht normal mit der Benutzeroberfläche gearbeitet werden. Da jedoch die meisten erstellten Fenster im Grunde nur eine entsprechend formatierte Liste enthalten, durch die navigiert wird, wurde bei jeder dieser Anwendungen ein eigener GUI Container mit der entsprechenden Liste erstellt, in der die Benutzereingaben bereits abgefragt werden.



7.2.4.1. xMedia Haupbildschirm

Der xMedia Hauptbildschirm beinhaltet im Grunde nur einige Icons, durch die navigiert werden kann, damit ist es möglich die einzelnen Unterprogramme, wie den Stream Bereich, den Musik Bereich, den Home Bereich oder den Web Bereich aufzurufen.

Die Navigation erfolgt durch das Wheel und die Rechts / Links Kommandos der xUI.

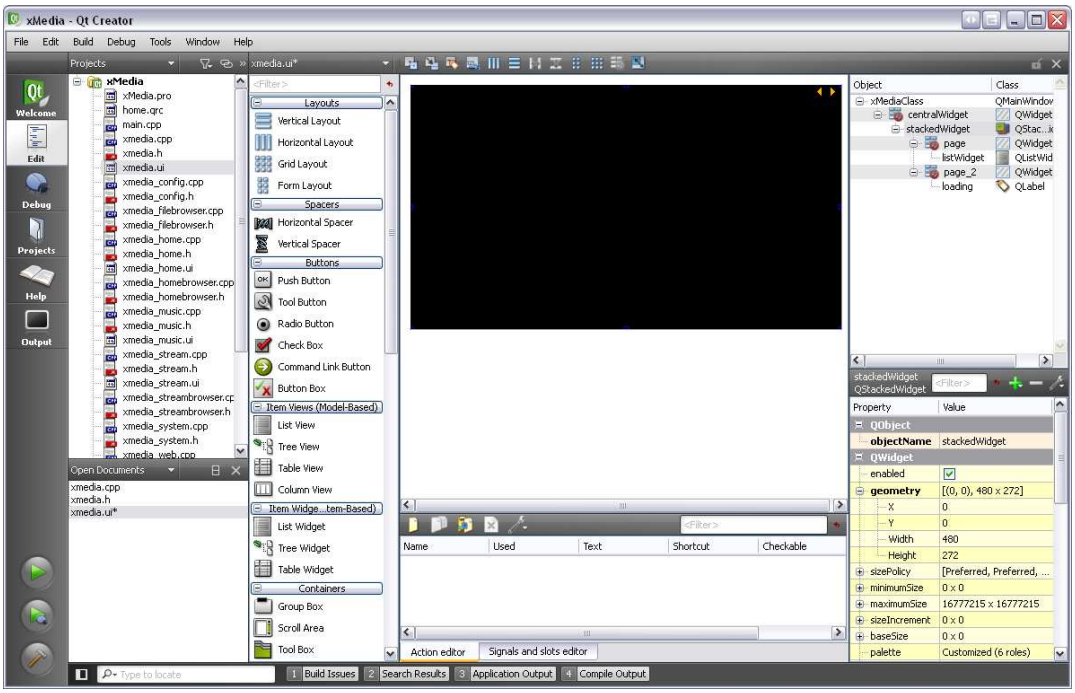


Abbildung 7.2.4.1-1 QTCreator Entwurf - Haupbildschirm

Nach dem Ausführen im Lokalen Testbetrieb, sieht der Bildschirm folgendermaßen aus:

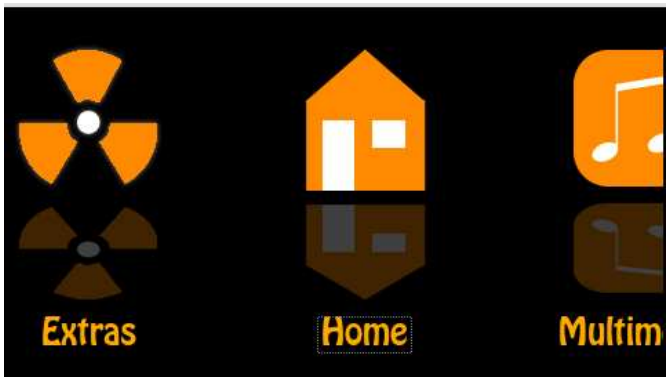


Abbildung 7.2.4.1-2 Hauptbildschirm

7.2.4.2. Der Musikbrowser

Das Musik Unterprogramm ist der zentrale Punkt der GUI. Das Programm setzt sich im Grund aus 3 Komponenten zusammen:

1. Dem Dateibrowser
2. Dem Mediaplayer
3. Der Lautstärkeregelung

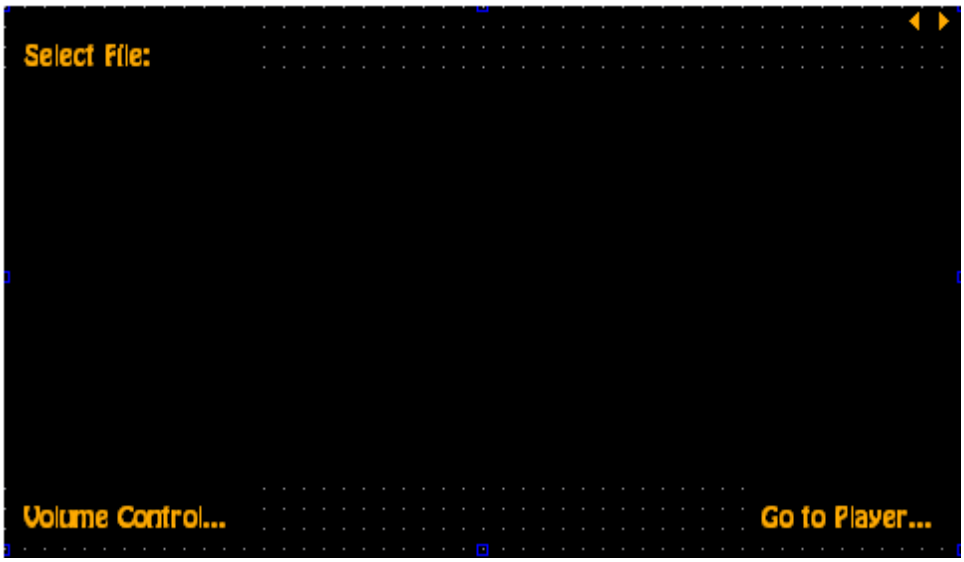


Abbildung 7.2.4.2-1 Dateibrowser (Entwurfsansicht)

Abbildung 7.2.4.2-2 Dateibrowser (lokaler Test)

Hinter dem Dateibrowser steckt die Klass xmedia_filebrowser, eine abgeleitete Klasse des QListWidget. Dadurch, dass diese Klasse von der QT QListWidget Klasse abgeleitet ist, kann ganz normal mit ihr im Design gearbeitet werden, nur der entsprechende Inhalt wird nicht angezeigt.

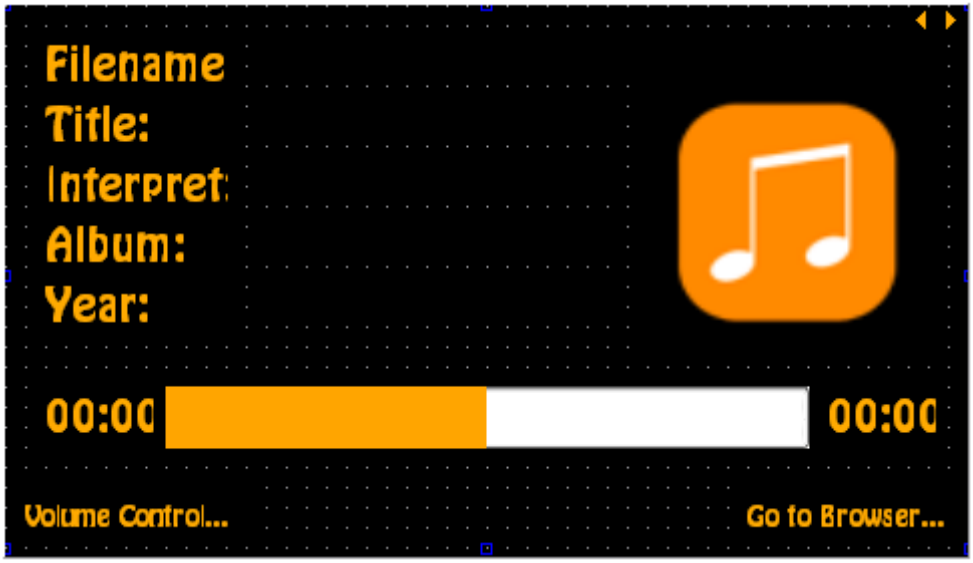


Abbildung 7.2.4.2-3 Media Player (Entwufsansicht)

Die Player GUI ist im Grunde nur eine Leerform mit Feldern, die später mit den entsprechenden Titelinformationen, dem Fortschritt um dem CD Cover gefüllt werden.

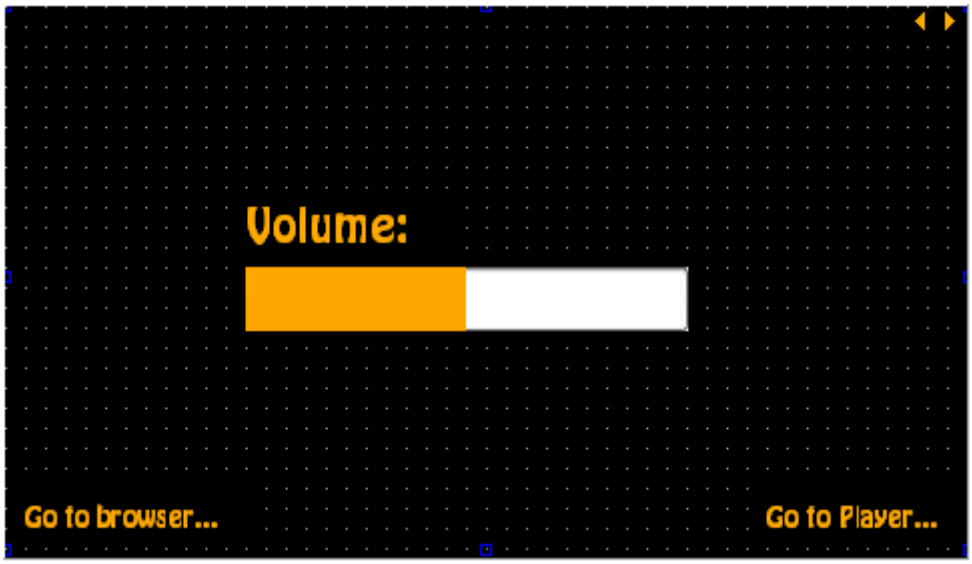


Abbildung 7.2.4.2-4 Lautstärkeregelung (Entwurfsansicht)

7.2.4.3. Der Streambrowser

Der Streambrowser ermöglicht das Navigieren durch alle gespeicherten Stream, er ist im Grunde wie der Dateibrowser aufgebaut, nur dass die hinter dem Streambrowser liegende Klasse xmeda_streambrowser nun nicht auf das Dateisystem zugreift und dieses anzeigt, sondern auf die Konfiguration, die die gespeicherten Streams beinhaltet.

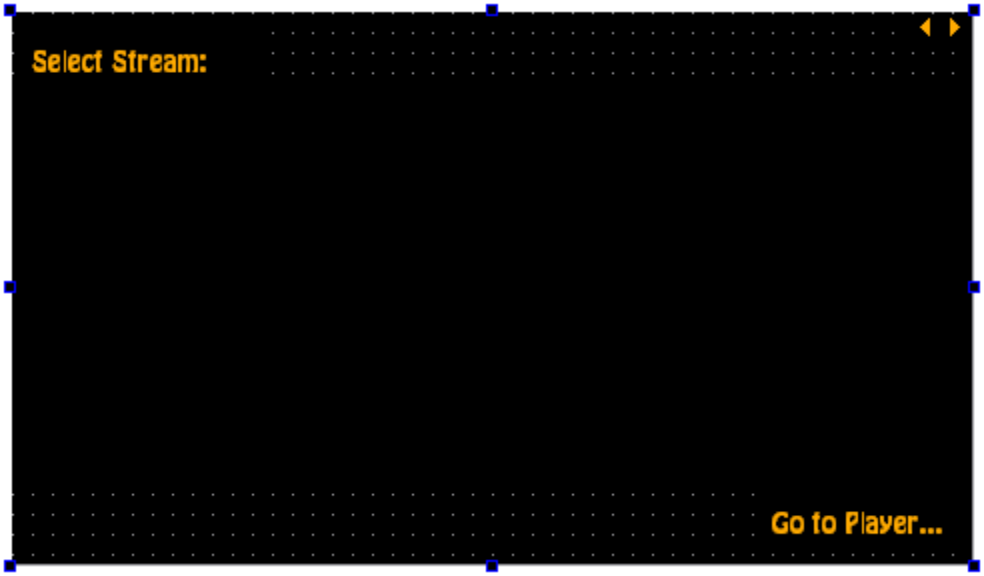


Abbildung 7.2.4.3-1 Streambrowser (Entwicklungsansicht)

Der Media Player ist hier, da er für Streaming-Media ausgelegt ist, auch etwas anders aufgebaut wie beim Dateibrowser.

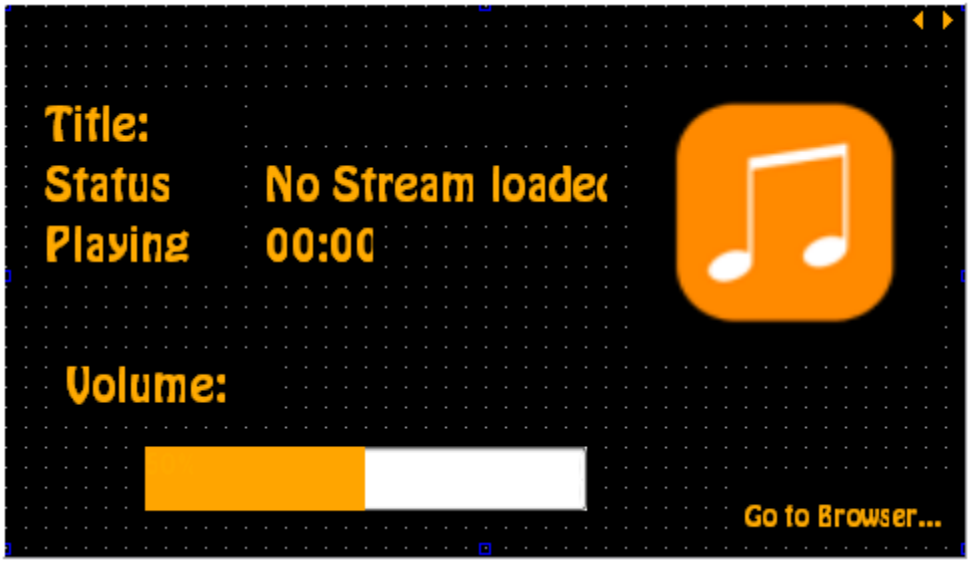


Abbildung 7.2.4.3-2 Streamplayer (Entwicklungsansicht)

Die Lautstärkeregelung ist komplett mit der Lautstärkeregelung im Dateibrowser identisch.

7.2.5. Die Konfiguration

Um die Anwendung konfigurieren zu können, bzw. genauer gesagt über das Webinterface konfigurieren zu können, musst eine gemeinsame Konfigurations-Klasse implementiert werden.

Die Konfiguration sollte möglichst einfach auch manuell zu editieren sein, sollte schnell sein und es ermöglichen das mehrere Instanzen gleichzeitig auf die Konfiguration zugreifen, und diese trotzdem konsistent bleibt. Außerdem sollte es möglich sein komplexe Datenstrukturen in der Konfiguration zu speichern.

Genau die beschriebenen Eigenschaften bietet die QStettings Klasse von QT. Sie ermöglicht das effiziente Schreiben / Lesen und Bearbeiten von „.ini“ Konfigurationsdateien, welche die Daten in Klartext abspeichern.

Beispielsweise sieht eine Konfigurationsdatei folgendermaßen aus:

```
[Media]
Shares\1\Address=//192.168.1.2/Musik
Shares\1\Index=0
Shares\1\Name=Home
Shares\size=1
Stations\1\Address=http://82.150.198.157:8000/welle1.mp3
Stations\1\Index=0
Stations\1\Name=Welle 1
Stations\2\Address=http://server1.digital-webstream.de:29915
Stations\2\Index=1
Stations\2\Name=BeatOne
Stations\3\Address=http://62.26.214.248:80/sunshinelive/livestream.mp3
Stations\3\Index=2
Stations\3\Name=Sunshine Live
Stations\4\Address=http://80.237.157.46:8120/
Stations\4\Index=2
Stations\4\Name=NRJ 104.2
Stations\5\Address=http://gffstream.ic.llnwd.net/stream/gffstream_w12a
Stations\5\Index=2
Stations\5\Name=Bayern 3
Stations\size=5

[Network]
Adapters\1\Address=192.168.1.33
Adapters\1\DHCP=false
Adapters\1\DNS=198.168.0.1
Adapters\1\Gateway=192.168.0.1
Adapters\1\Subnet=255.255.255.0
Adapters\2\Address=192.168.1.33
Adapters\2\DHCP=false
Adapters\2\DNS=198.168.0.1
Adapters\2\Gateway=192.168.0.1
Adapters\2\Subnet=255.255.255.0
Adapters\size=2
```

Das Speichern und Einlesen komplexerer Strukturen, wie der Netzwerkkonfiguration wurde über eigens implementierte Serialisierungsroutinen gelöst, die alle Werte zunächst in einen String umwandeln und diese Werte dann speichern.

7.2.5.1. Funktionen

void **save**();

Speichert die Konfigurationsdatei ab

void **reload**();

Lädt die Konfigurationsdatei (neu)

void **setParameter**(QString name, QString value);

Setzt / Ändert einen allgemeinen Parameter.
name … Name des zu setzenden Parameters
value … Wert auf den der Parameter gesetzt werden soll

QString **getParameter**(QString name);

Fragt einen allgemeinen Parameter ab. Der Wert wird als String zurückgegeben.
Name … Name der Parameters

7.2.5.2. Variablen

Zum Zugriff auf Datenarrays bzw. Strukturen stehen drei Membervariablen vom Typ QList zur Verfügung, die die entsprechenden Konfigurationen beinhalten.

QList<xMediaConfigNetworks> networks;

Diese Liste beinhaltet die Netzwerkkonfiguration für alle Netzwerkschnittstellen. Die Struktur die in der Liste gespeichert wird hat folgenden Aufbau:

```
typedef struct
{
   bool dhcp;              //DHCP Server verwenden
   QString address;        //IP Adresse
   QString subnet;         //Subnetz Maske
   QString gateway;        //Standardgateway
   QString dns;            //Standard DNS Server
} xMediaConfigNetworks;
```

QList<xMediaConfigStations> stations;

Diese Liste beinhaltet alle Webradios. Die Struktur die in der Liste gespeichert wird hat folgenden Aufbau:

```
typedef struct
{
   int index;              //Index des Webradios, für die Auslistung von Bedeutung
   QString address;        //URL zum Stream
   QString name;           //Angezeigter Name des Webradios
} xMediaConfigStations;
```

QList<xMediaConfigShares> shares;

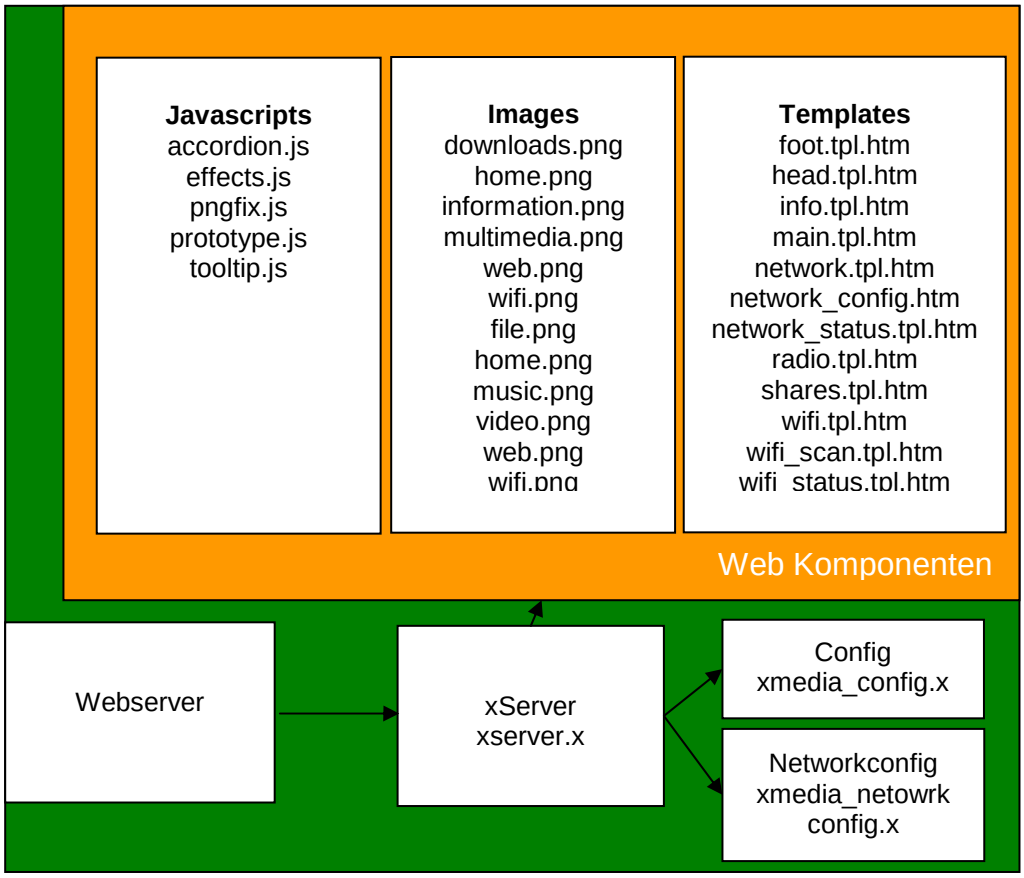
Diese Liste beinhaltet alle Netzwerkfreigaben. Die Struktur die in der Liste gespeichert wird hat folgenden Aufbau:

```
typedef struct
{
    int index;              //Numerischer Index zur Identifizierung des Datensatzes
    QString address;        //Adresse der Netzwerkfreigabe
    QString name;           //Angezeigter Name der Netzwerkfreigabe
} xMediaConfigShares;
```

7.3. Die xWeb Webinterface

Das Webinterface soll es ermöglichen, Einstellungen an der Konfiguration vorzunehmen, die Netzwerkadapter zu konfigurieren, Drahtlosnetzwerke zu suchen und einzustellen, Webradios einzutragen und Netzwerkfreigaben festzulegen.

Der mittels Buildroot installierte Webserver hat keine Unterstützung für die im Web häufig verwendete Programmiersprache PHP, daher blieb nur eine nativ auf dem System ausführbare Sprache wie C++ übrig. Dies kam uns eigentlich recht gelegen, da wir so den Code von der xMedia Software teilweise wiederverwenden konnten. Außerdem war das Einbinden von C++ / Qt Programmen durch das CGI relativ einfach – es musste im Grunde nur eine Konsolenapplikation erstellt werden.



7.3.1. Der xServer

Der xServer ist das einzige CGI Programm des Webinterfaces und kombiniert alle Funktionen die benötigt werden.

Ein CGI Skript ist im Grunde genommen ein Kommandozeilenprogramm das vom Webserver gestartet wird, wenn die entsprechende URL aufgerufen wird. Im Vergleich zu einem PHP Skript hat man C++ basierten Programmen vollen Hardwarezugriff was hier von Vorteil ist, da zum Beispiel direkt auf den Wireless Lan Adapter zugegriffen werden kann.

Die Parameter, die per GET, d.h. über die Adressleiste des Webservers übergeben werden, werden über die Umgebungsvariable „QUERY_STRING“ an das Programm übergeben und können mittels String Manipulation (es gibt in QT keine fertigen Funktionen zum Parsen solcher Wertepaare, da QT nicht für CGI Skripts ausgelegt ist).

Die ganze Programmaktion findet nun in dem xServer Programm statt:

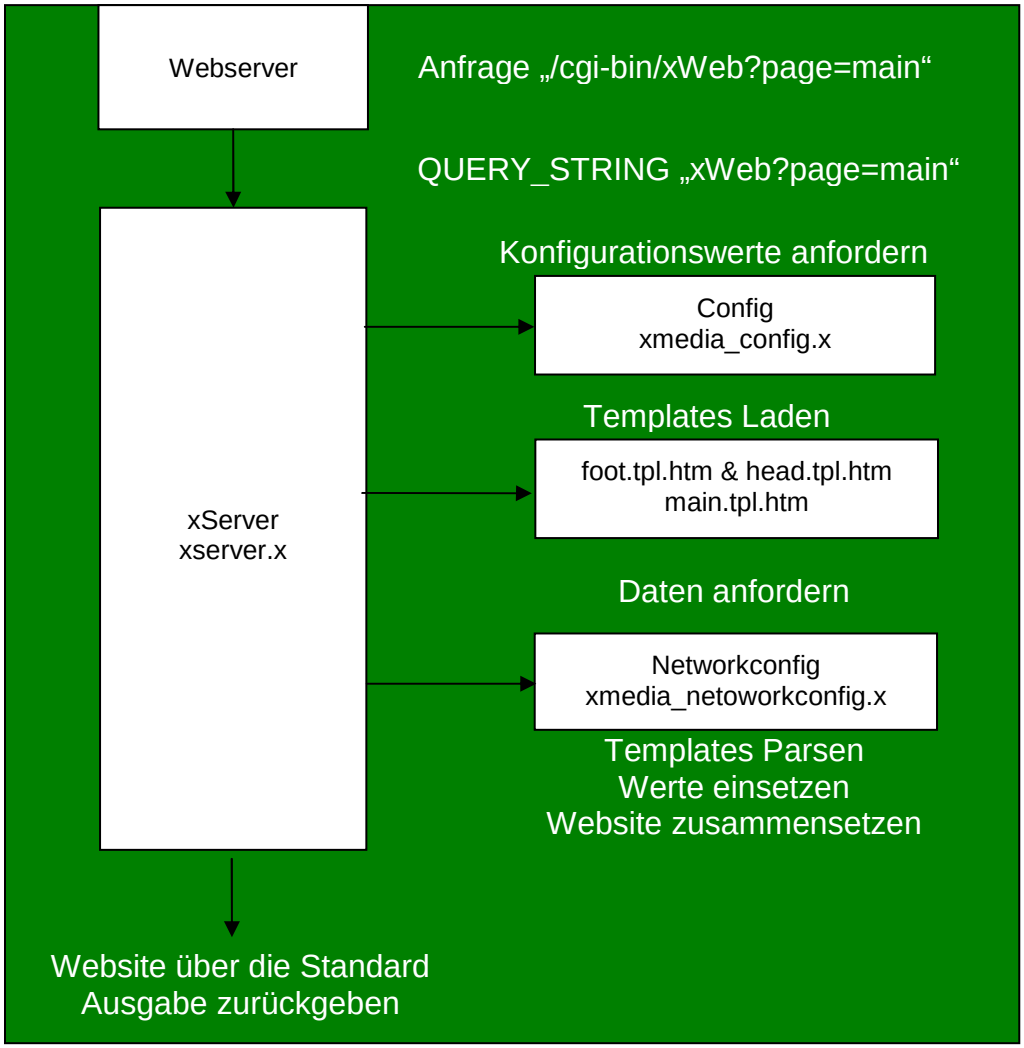


Abbildung 7.3.1-1 xServer Programmablauf

7.3.2. Die Netzwerk Konfiguration

Die Netzwerkkonfigurations-Klasse bietet nun Zugriff auf die Einstellungen der Netzwerkadapter und die Drahtlosnetzwerk Einstellungen. Die Klasse greift direkt auf die Kommandozeilentools „ifconfig“, „iwconfig“ und „route“ zu.

7.3.2.1. Funktionen

QList<xmedia_networkConfigWifiStatus>
xmedia_networkConfig::wifiAvailableNetworks();

Sucht alle in der Umgebung verfügbaren Drahtlosnetzwerke und gibt die Liste zurück. Die zurückgegeben Datenstruktur ist eine Liste die folgende Datenstruktur beinhaltet:

```
typedef struct{
   QString essid;              //Name des Drahtlosnetzwerkes
   QString frequency;          //Frequenz auf der das Netzwerk sendet
   QString accessPoint;        //Kennung des Zugriffspunkts
   int linkQuality;            //Signalqualität
   QString linkSignalLevel;    //Signalstärke
   QString linkNoiseLevel;     //Signalrauschstärke
   QString bitRate;            //Datenrate
   QString txPower;            //Sendeleistung
   QString secKey;             // Netzwerkschlüssel
   QString secMode;            //Verschlüsselungsmodus
} xmedia_networkConfigWifiStatus;
```

void **xmedia_networkConfig::wifiConfigureNetwork**(QString interface, QString essid, QString key, QString datarate);

Konfiguriert die Wireless Lan Schnittstelle und stellt sie auf ein bestimmtes Netzwerk ein.

interface	… Kennung der Netzwerkkarte
essid	… Name des Drahtlosnetzwerkes
key	… Netzwerkschlüssel
datarate	… Übertragungsrate

xmedia_networkConfigWifiStatus **xmedia_networkConfig::wifiNetworkStatus**();

Gibt den Status der aktuellen Verbindung des Drahtlosnetzwerkadapter zurück.

QList<xmedia_networkConfigNetworkStatus>
xmedia_networkConfig::networkStatus();

Gibt den Netzwerkstatus aller verfügbaren Netzwerkadapter zurück. Die zurückgegeben Datenstruktur ist eine Liste die folgende Datenstruktur beinhaltet:

```
typedef struct{
   QString name;          //Name des Netzwerkadapters
   QString mac;           //MAC Adresse des Adapters
   QString ip;            //IP Adresse
   QString subnet;        //Subnetzmaske
   QString gateway;       //Standard Gateway
   long bytesRX;          //Anzahl der empfangenen Datenbytes
   long bytesTX;          //Anzahl der gesendeten Datenbytes
} xmedia_networkConfigNetworkStatus;
```

void **xmedia_networkConfig::networkConfigure**(QString interface, bool useDhcp, QString ip, QString subnet, QString gateway, QString dns);

Konfiguriert die Netzwerkeinstellungen eines Drahtlosnetzwerkadapters.

interface	…Name des Netzwerkadapters
useDhcp	…Netzwerkeinstellungen vom DHCP Server beziehen (alle weiteren Einstellungen werden ignoriert)
ip	…IP Adresse
subnet	…Subnetzmaske
gateway	…Standard Gateway
dns	…Standard DNS Server

8. Das Gehäuse

8.1. Allgemein

Der erste Eindruck zählt!“ – Diesen Satz hört man immer wieder und bemerkt ebenso oft, dass diese Feststellung nicht aus der Luft gegriffen ist. Warum sollte diese, schon seit vielen Jahren bekannte, Feststellung nicht auch auf ein Gehäuse übertragbar sein.
Inspiriert durch diese Feststellung legten wir unser Augenmerk besonders auf das Gehäusedesign und auf die verwendeten Materialien. Um dem Gehäuse eine gewisse Dynamik zu verleihen wurde es mit speziellen Rundungen ausgestattet. Auch auf exklusive Materialien wurde Wert gelegt. Hierfür beschränkten wir uns auf edles Nussholz mit eingelegten Plexiglaselementen. Die edle Optik, hervorgerufen durch das dunkle Nussholz, verleiht unserem Projekt einen eigenen Charakter.
Das Design selbst unterstreicht die Funktionalität, Moderne und Innovation unseres Projekts. Auch bei der Gehäusekonstruktion beruhten unsere Entwicklungen auf möglichst viele Schrauben zu verzichten. Dies geschah aus 2 Gründen: 1. um die mechanische Beanspruchung zu minimieren und 2. Um das Gesamtkunstwerk nicht mit störenden Schrauben zu verunstalten.

„Der erste Eindruck zählt - Der letzte Eindruck bleibt“

8.2. Materialien

8.2.1. Gehäuse-Materialien

Bei der Auswahl der Materialien wurde auf Robustheit, gutes Aussehen und exzellente Bearbeitungseigenschaften wert gelegt. Wir entschieden uns für edles Nussholz. Da die Wandstärke besonders dünn sein sollte, um eine angemessene Größe zu erreichen, musste das Material besonders robust sein und durfte bei der Bearbeitung nicht splittern oder einreißen. Da Nussholz diese Eigenschaften aufweist, entschieden wir uns für dieses edle Material.

Abbildung 8.2.1-1 Nussholzmaserung

Nuss

Das eher harte Nussholz zeigt ein höchst dekoratives Erscheinungsbild, das sich durch zonenweise Streifen oder dunklere Maserung auszeichnet.

Eigenschaften

Nussholz gilt als mittelschwer bis schwer (Darrdichte 640 kg/m³), in der Härte gibt es in manchen Normenwerken irreführend hohe Werte, sie liegt bei 30 N/mm². Das Holz lässt sich gut, wenn auch nur langsam trocknen. Es ist sehr gut zu bearbeiten, zu beizen und polieren. Beim Verleimen können durch Alkalien in den Leimen Gerbsäureflecken entstehen. Bei Kontakt mit Eisen entsteht eine blauschwarze Färbung, und es kommt zu ausgeprägter Korrosion.

8.2.2. Display-Material

Als Displayschutz wurde ein Plexiglaselement verwendet. Plexiglas ist annähernd kratzfest und bittet die nötige Stabilität um angemessenen Schutz für das Display zu gewährleisten. Um keine eingeschränkte Sicht vorzufinden, wurde klares, nicht getöntes Plexiglas verwendet.

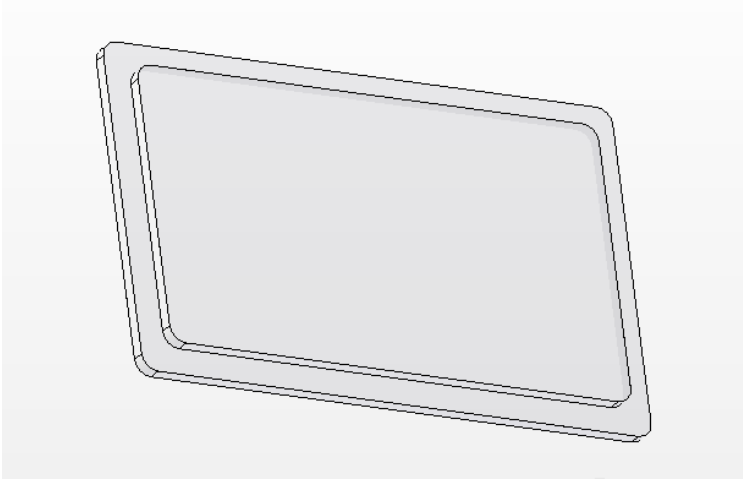


Abbildung 8.2.2-1 Plexiglasdisplay

8.3. Aufbau

Beim Aufbau wurde auf minimalen Schraubeneinsatz wert gelegt, jedoch sollten die Teile fest miteinander verbunden sein. Auch für die Befestigung der Platine musste eine Lösung gefunden werden.
Das Gehäuse besteht im Grunde aus zwei Grundelementen, Schlagen genannt. Diese Schalen wurden mit einer Nut versehen, um jegliches Verrücken der beiden Elemente zu eliminieren. Um nun die beiden Platten fest miteinander zu verbinden wurden an der oberen und unteren Kante, Schraubenelemente eingesetzt. Die Befestigung der Platine erfolgt über weitere Schraubenelemente. Die Platine wird mittels Muttern und Schrauben mit der unteren Schale des Gehäuses befestigt. Dies bedeutet es wurde auch auf eine Sicherung wert gelegt, um die Platine zu entlasten, da das Gehäuse selbst befestigt werden kann und die Platine separat.
Bei der Befestigung des Plexiglaselementes wurde wiederum mit einer Nut gearbeitet. Die Nut des Plexiglases bietet das exakte Gegenstück zur Nut des Gehäuses. Es wurde auf einen planen Übergang zwischen Plexiglas und Holz wert gelegt. Hierfür mussten die beiden Elemente exakt bearbeitet werden.
Um die diversen Anschlüsse herauszuführen, mussten die Anschlüsse exakt ausgefräst werden.

Abbildung 8.3-1 Gehäuse Explosionsdarstellung

8.3.1. Unterteil:

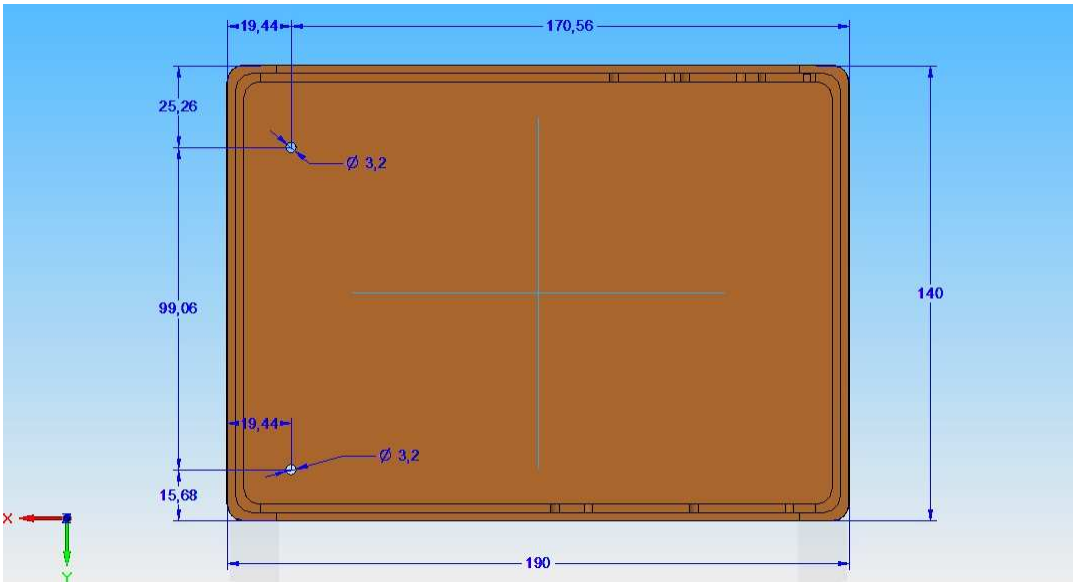


Abbildung 8.3.1-1: Unterteil-Draufsicht

Bei der Gehäuse-Konstruktion wurde besonders auf platzsparendes Design wert gelegt. Der äußere Umfang des Gehäuses beträgt 140*190 mm. Bei einer Wandstärke von 5 mm. Um die untere mit der oberen Schalenhälfte zu vereinen wurden 2 Befestigungsbohrungen vorgesehen. Diese Befestigungsbohrungen dienen nicht nur zur Vereinigung der beiden Gehäuse-Hälften, sondern auch zur Befestigung der Platine.

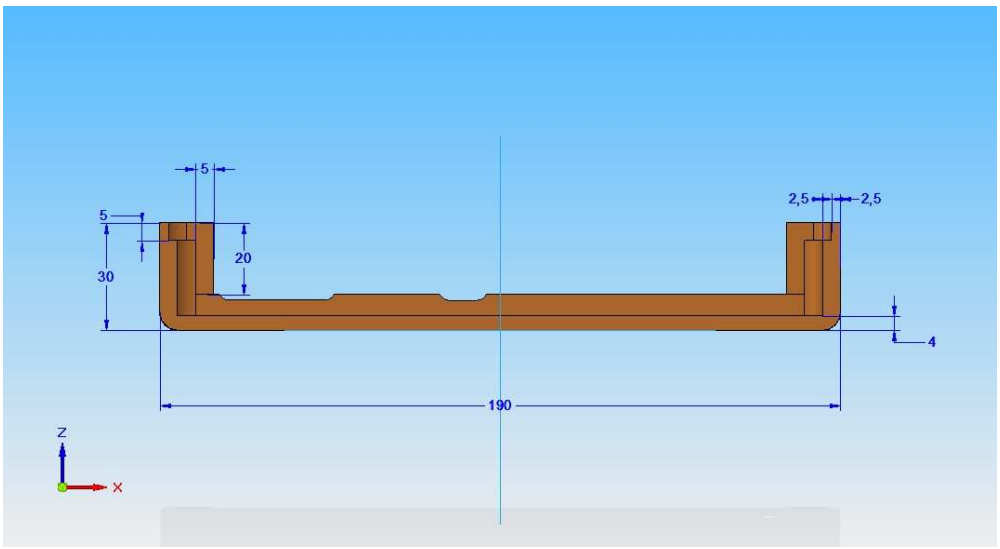


Abbildung 8.3.1-2: Schnitt- Seitenansicht

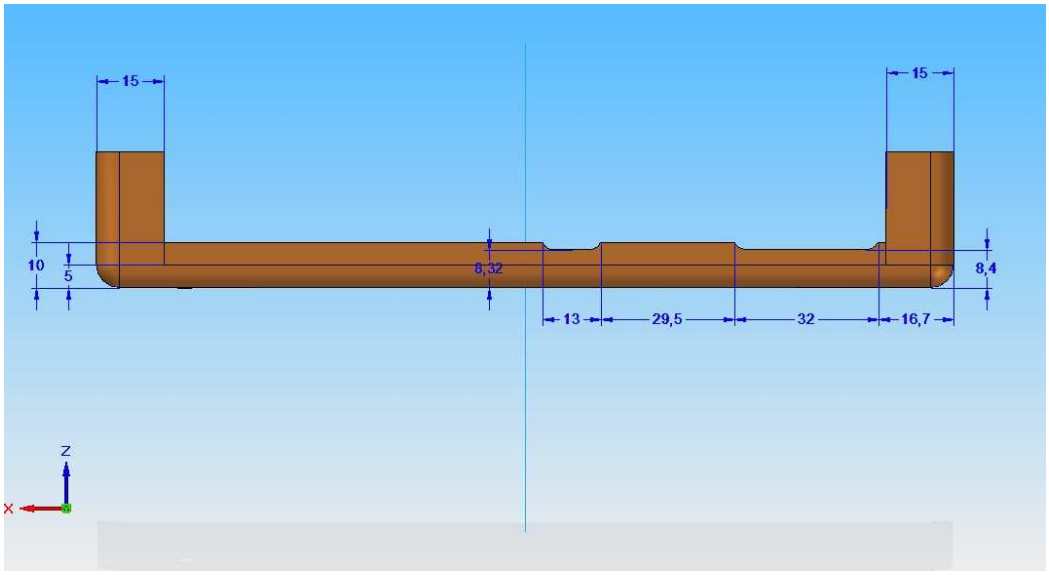


Abbildung 8.3.1-3: Seitenansicht

Auch auf die Höhe des Gehäuses sollte geachtet werden. Dafür wurden die Gehäuse-Hälften auf eine Höhe von 30 mm reduziert.

8.3.2. Seitenteile 1:

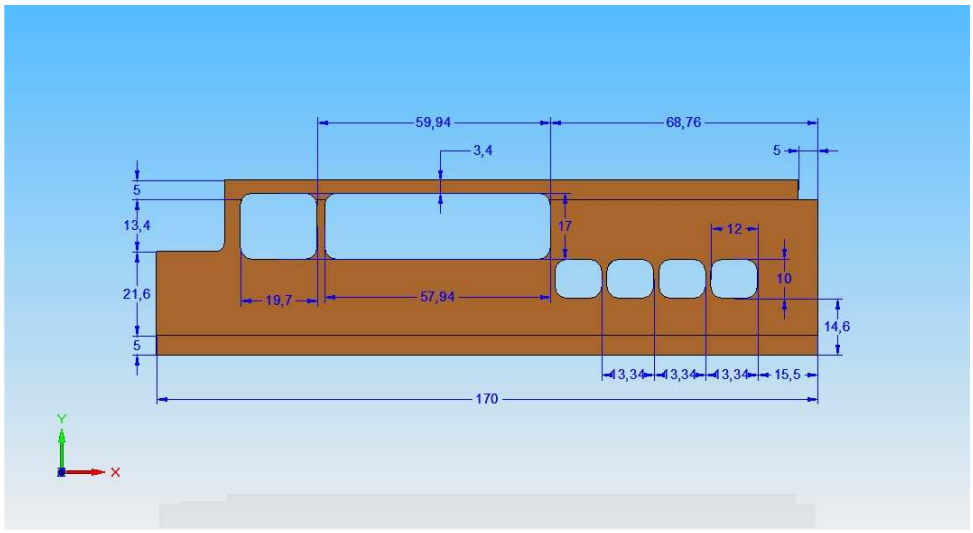


Abbildung 8.3.2-1: Seitenteil 1-Ausschnitt

Bei den Seitenteilen mussten die Anschlüsse nach außen geführt werden. Auch hierfür wurde auf besondere Eleganz und Genauigkeit wert gelegt. Genauigkeit deswegen, da die Anschlüsse ohne Probleme zu erreichen sein mussten. Um die Seitenteile mit den beiden Hälften zu verschmelzen, wurde eine Nut vorgesehen, die von der oberen und unteren Hälfte ergänzt wird.

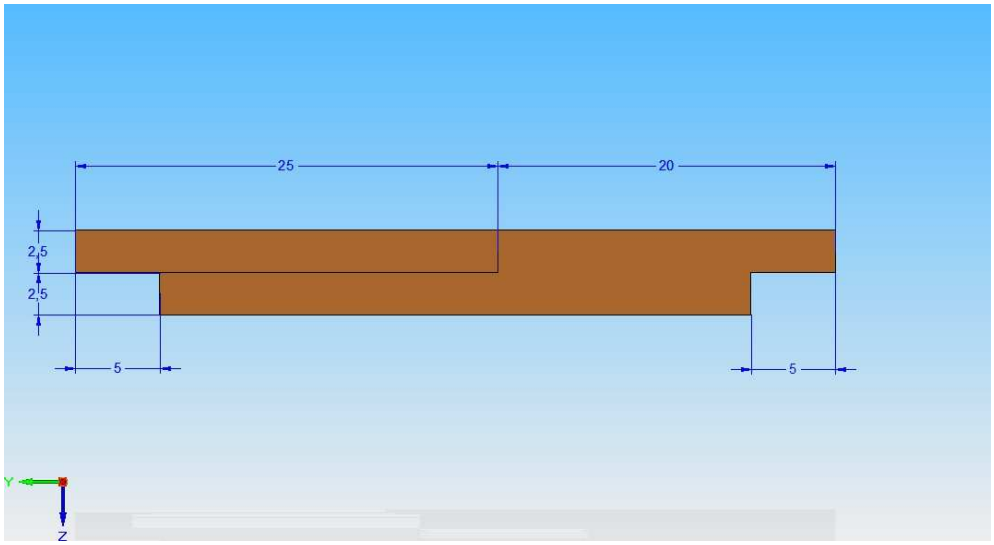


Abbildung 8.3.2-2: Seitenteil 1

Auch bei der Tiefe wurden auf Robustheit sowie Platzersparnis wert gelegt.

8.3.3. Seitenteil 2:

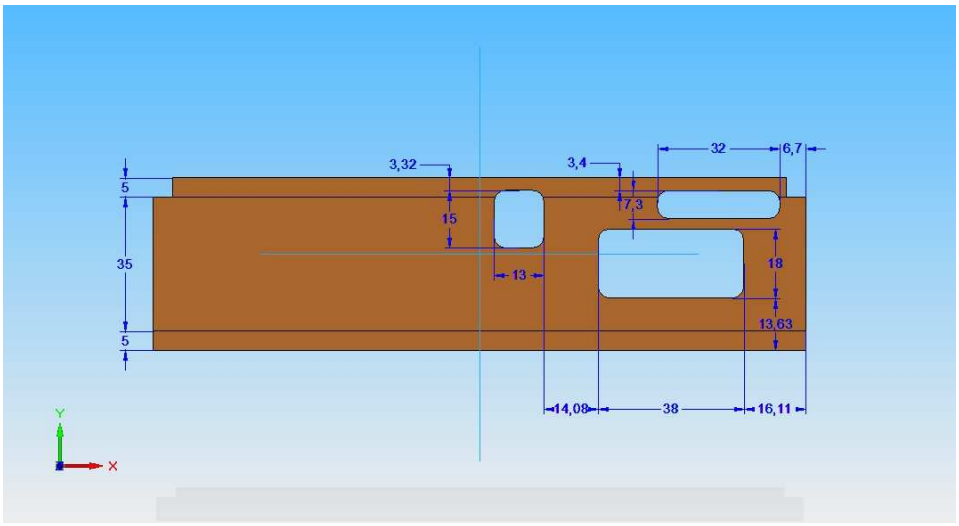


Abbildung 8.3.3-1: Seitenteil 2-Ausschnitt

Auch für die Komplementärseite wurden die Anschlüsse herausgeführt. Hierbei handelt es sich um einen SD-Karten Slot und eine VGA-Schnittstelle, sowie einen Anschluss für die Energieversorgung.

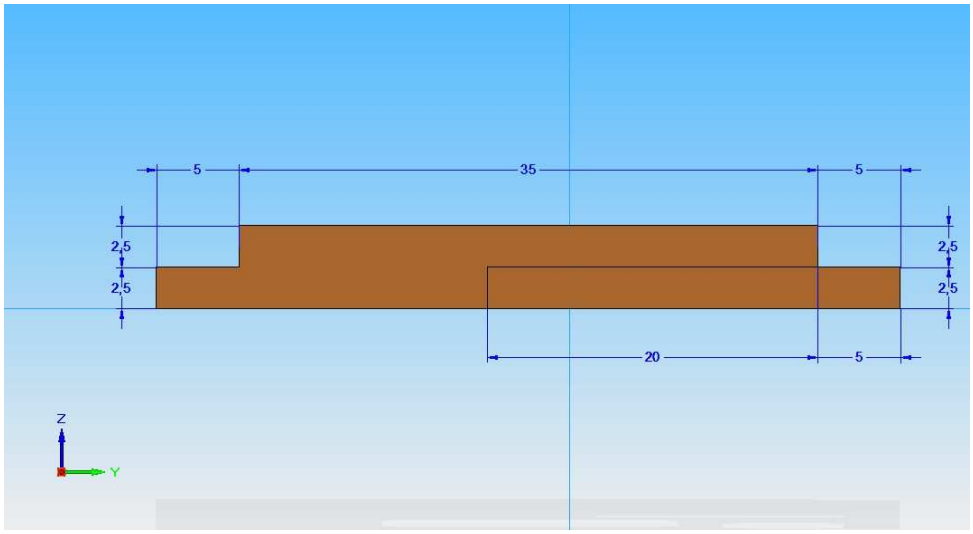


Abbildung 8.3.3-2: Seitenteil 2

Wie bei der der Komplementärseite wurde auch hier eine spezielle Nut zur Verschmelzung der Seitenteile mit den beiden Hälften vorgesehen.

8.3.4. Oberteil:

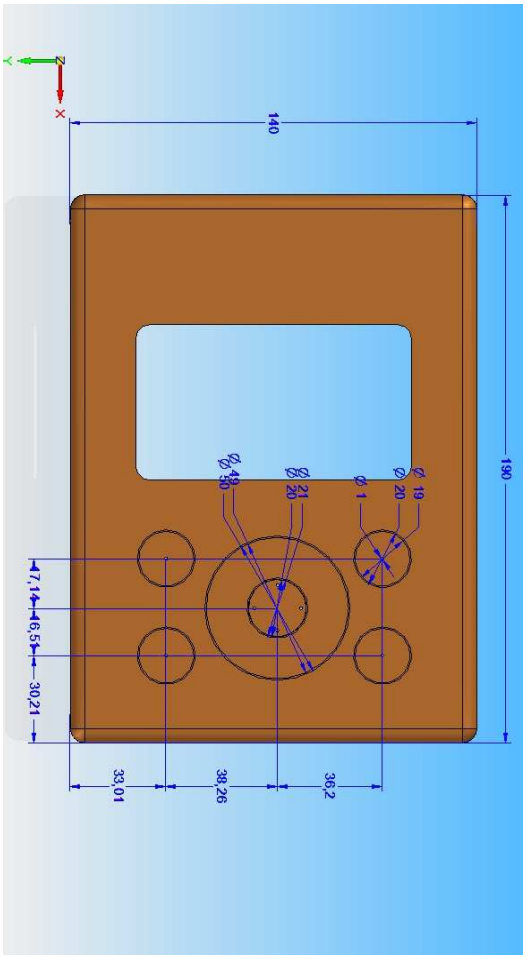


Abbildung 8.3.4-1: Oberteil

Die obere Gehäuse-Hälfte stellt den Mittelpunkt des Gehäuse-Design dar. Für das Display wurde ein genau angepasster Ausschnitt getätigt, der genau auf die Maße des Displays ausgelegt wurde. Um die kapazitiven Tasten zu kennzeichnen, die durch Holz

jeweilige Berührungen problemlos erkennen, wurden leichte Kreis-Ausfräsungen vorgenommen. Auch für die Leds wurde eine kleine aber feine 1mm Bohrung durchgeführt. Dies bringt einen gewissen Style-Effekt mit sich.

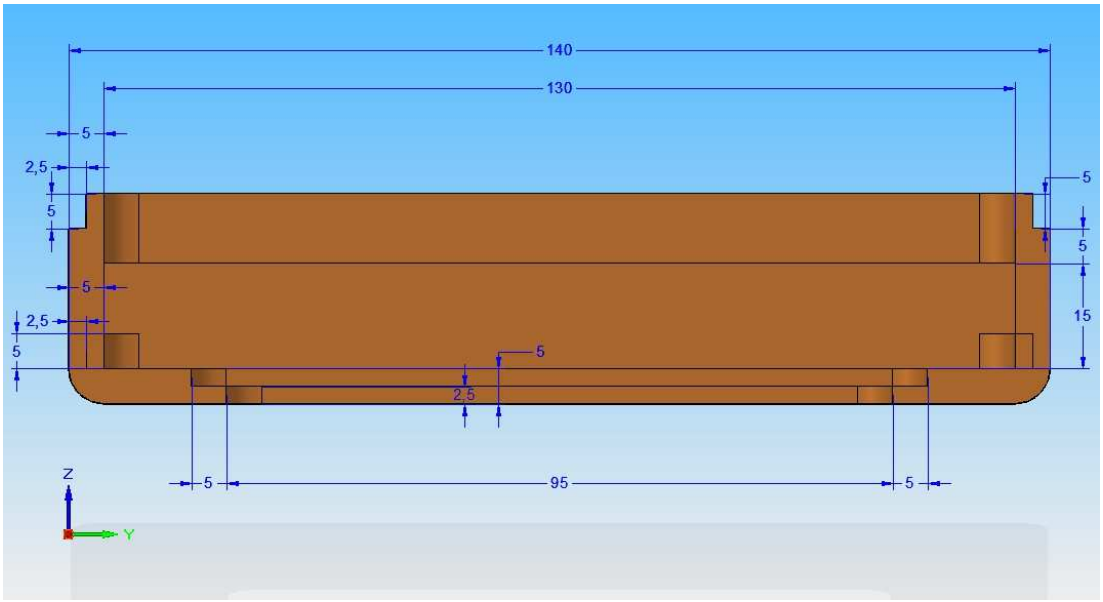


Abbildung 8.3.4-2: Oberteil-Schnitt

Um die Platine einen gewissen Halt zugeben, wurde kleiner Holzblock mit einer Höhe von 15 mm eingefügt. Die Platine wird auf diesen Block mit 2 Schrauben befestigt und somit stabilisiert.

8.4. Zusammenarbeit im Team

Wie in jedem Projekt ist einer der wichtigsten Aspekte die Zusammenarbeit der einzelnen Projektmitglieder. Wir harmonierten gut miteinander und versuchten alles, wie besprochen, so gut wie möglich in die Tat umzusetzen. Die umfangreiche Arbeit am Projekt erforderte auch einen sehr großen Zeitaufwand. Darum war es wichtig, dass beim Arbeiten ein gewisser Spaßfaktor vorhanden war, der jedoch niemanden von der Arbeit abhielt.
Bei den täglichen Teambesprechungen hatten wir immer wieder unsere Ziele für den Tag besprochen und die jeweiligen anderen Projektmitglieder über die genaueren Arbeitsschritte, die durchzuführen waren, informiert.
So bekam jeder einen Überblick über den Aufgabenbereich des anderen. Wir würden das Arbeiten am Projekt auch als „Paralleles Lernen“ bezeichnen, da jeder einen Spezialbereich über hat und gleichzeitig mit den anderen über ihr jeweiliges Thema mitlernt.
Bevor Beendigung eines Arbeitstages wurden die erreichten Ziele des Tages besprochen und es wurde auf die nächsten Ziele näher eingegangen. Durch diese Besprechungen konnte sich jeder individuell auf neue Situationen vorbereiten.

Die Distanz zwischen den Wohnorten war jedoch ein großer Nachteil für die Zusammenarbeit. Dies machte es sehr kompliziert an Wochenenden oder in den Ferien gemeinsam am Projekt zu arbeiten oder die Fortschritte des Einzelnen den Anderen zu präsentieren. An solchen Tagen konnte nur online oder via Telefon kommuniziert werden. Auftretende Probleme konnten so nur auf Umwegen beseitigt werden.

Da unser Projekt eine Eigeninnovation ist und ohne eine Firma durchgeführt wurde, war es unmöglich Firmen dazu zu begeistern uns bei dem kostenaufwendigen Projekt finanziell zu unterstützen.

Interessant am Teamprojekt war, dass man immer wieder auf neue Herausforderungen stieß. Neue Problemlösungen mussten gefunden werden und so konnten wir unserer Kreativität freien Lauf lassen.
Das selbstständige Arbeiten war auch sehr positiv im Hinblick auf die Zukunft, da man auch später im Berufsleben immer wieder auf Hindernisse trifft, welche man aus eigener Kraft und mit neuen Ideen überwinden muss.
Für die Zukunft ist auch mitzunehmen, dass wir bereits ein Projekt im Team realisiert haben und so eventuell in späteren Gruppenarbeiten einen Vorteil, auf Grund der gemachten Erfahrungen, aus diesem Projekt ziehen.
Obwohl wir uns bereits seit 5 Jahren kennen und uns dachten viel über einander zu wissen, lernten wir immer wieder neue Stärken und Schwächen des Anderen kennen.
Durch die Teamarbeit und die gegenseitige Hilfestellung konnte auch die eine oder andere Schwäche ausgebessert werden.

Hinter diesem Projekt steckt ein immenser Zeitaufwand. An vielen Tagen wurde auch bis in die Nachtstunden gearbeitet, um das nächste Ziel zu erreichen.
Neben der gesamten Hard- und Softwareentwicklung gab es noch viele andere Bereiche, mit denen man sich zeitintensiv beschäftigen musste. Zu diesen Aufgaben zählten unter anderem die Gehäusekonstruktion, Bestellungen von Bauteilen, Leiterplattenfertigung, Homepage und vor allem Dokumentation.

9. Anhang

Aufgrund der hohen Anzahl an Stromlaufplänen, Layouts und anderen Fertigungsunterlagen sind diese nur in digitaler Form auf dem beiliegenden Datenträger verfügbar.

9.1. Hardwaredokumentation

Die Hardwaredokumentation ist in die drei Projektteile eingeteilt:

1. Der xMedia Player (Ordner „xMedia Player“)
   - “Datasheets” – Datenblätter der Komponenten des Players
   - „Dokumentation“ – Dokumentation der Hardware
   - „Firmware“ – Firmware für den Boardcontroller, den xDC und ein Testprogramm
   - „Gehäuse“ – Solid Edge Dateien des Gehäuses
   - „Hardware“ – Eagle Stromlauf und Layoutdateien
   - „Software“ – Linux Software

2. Die xRemote (Ordner „xRemote“)
   - “Datasheets” – Datenblätter der Komponenten des Players
   - „Dokumentation“ – Dokumentation der Hardware
   - „Firmware“ – Firmware für den Boardcontroller, den xDC und ein Testprogramm
   - „Hardware“ – Eagle Stromlauf und Layoutdateien

3. Die xDimension (Ordner „xDimension“)
   - “Informationen” – Zusatzinformationen und Kalkulationen
   - „Dokumentation“ – Dokumentation der Hardware
   - „Hardware“ – Eagle Stromlauf und Layoutdateien

9.2. Bilder

Im Ordner „/Bilder/“ auf dem beiliegenden Datenträger finden Sie einige Bilder des xMedia und xRemote Prototypen.

9.3. Videos

Der Ordner „/Videos/“ beinhaltet einige kurze Demonstrationsvideos von den Prototypen / Finalversion aller Projektteile.

„Demo“
- xmedia.wmv – Einleitung, Hardwareerklärung, Funktionsdemonstration
- xremote.wmv – Hardwareerklärung, Funktionsdemonstration (mit xMedia Player)
- xdi – xDimension Funktionsdemonstration

9.4. Marketing

Der Ordner „Marketing“ beinhaltet alle Logo, Grafiken, Flyer und Plakate, die während der Projektphase erstellt wurden.

9.5. Linux

Der Ordner Linux beinhaltet alle Inhalte die zum Aufsetzen und zum Konfigurieren des Entwicklungssystems nötig sind.

9.6. Wettbewerbe

Dieser Ordner beinhaltet alle Dateien die im Zuge von Wettbewerbsteilnahmen erstellt wurden.

10. Abbildungsverzeichnis