Masterarbeit

Design und Implementierung einer Software-Architektur für Multimedia-Anwendungen auf einer mobilen Hardware-Plattform

Peter Randeu

Institut für Technische Informatik
Technische Universität Graz
Vorstand: O. Univ.-Prof. Dipl.-Ing. Dr. techn. Reinhold Weiß

Begutachter: Ass.Prof. Dipl.-Ing. Dr. techn. Christian Steger
Betreuer: Ass.Prof. Dipl.-Ing. Dr. techn. Christian Steger

Graz, im Mai 2011

Kurzfassung

Diese Arbeit beschreibt das Design und die Implementierung einer Anwendung für ein Embedded System zur Aufzeichnung und Echtzeitübertragung von Audio- und Videodaten. Sie wurde bei der Grazer Firma Spintower durchgeführt.

Das Embedded System basiert auf einem System-on-Chip (SoC) mit einer ARM-CPU und einer Video Processing Unit, die H.264-Videos in Echtzeit kodieren und dekodieren kann. Ziel der Arbeit ist es, eine Anwendung zu entwickeln, die Video- und Audiosignale in Echtzeit aufzeichnet und über eine Netzwerkverbindung zu einem Server überträgt. Auch die Signale des integrierten GPS-Empfängers sollen aufgezeichnet und übertragen werden. Teil der Anwendung ist zudem eine grafische Benutzeroberfläche, die am Display des Embedded System angezeigt und über dessen Touchscreen bedient werden kann.

Diese Arbeit beginnt mit der Recherche zu verwandten Arbeiten. Auf einen Vergleich verschiedener SoC, die als Hardwarebasis für diese Anwendung in Frage kommen, folgt eine Analyse der Funktionen kommerzieller Produkte, die zur mobilen Echtzeitübertragung von Video- und Audiosignalen eingesetzt werden. Schließlich werden einige Forschungsprojekte vorgestellt, die sich mit der Aufzeichnung, Übertragung und Wiedergabe von Video- und Audiosignalen auf Embedded Systems beschäftigen.

Das Design-Kapitel beschreibt den Entwurf der Anwendung. Es werden die Funktionen der Hardware des Embedded System analysiert und Überlegungen zu deren Nutzung durch die Software angestellt. Zusammen mit den Ergebnissen der Analyse der Anforderungen des Benutzers an die Anwendung, wird daraus ein objektorientiertes Design der Anwendung erstellt. Diese unterteilt sich in vier Hauptkomponenten: eine Bibliothek zur Nutzung der Video- und Audiohardware, eine Klasse zur Auswertung der GPS-Signale, eine Klasse zur Netzwerkkommunikation und die grafische Benutzeroberfläche.

Die Implementierung der Anwendung erfolgt in C++, unter Verwendung der Qt Klassen-Bibliothek. Der Zugriff auf die Video- und Audiohardware erfolgt mit Hilfe des Multimedia-Frameworks GStreamer.

Abschließend folgt eine Analyse der erzielten Resultate. Die Qualität der Übertragung eines Videos bei unterschiedlichen Verbindungeigenschaften wird untersucht. Der Stromverbrauch des Embedded Systems für verschiedene Betriebszustände wird gemessen und ausgewertet.

Abstract

This master's thesis describes the design and the implementation of a multimedia software architecture for an embedded system. It was developed for the company Spintower in the city of Graz.

The embedded system is based on a system on chip (SoC) that consists of an ARM CPU and a video processing unit (VPU). The VPU can encode and decode H.264 videos in real-time. Goal of this work is to develop a software application for the embedded system, that records audio and video signals and transmits them to a server in real-time. The signals of the integrated GPS receiver are also to be transmitted to the server. Part of the application is a graphical user interface (GUI); it is controlled by the user through the touchscreen of the embedded system.

The work starts with a research on related topics. Properties of different SoC that may be used as hardware platforms for the application are compared. An overview of the functionality of different commercial video recording and transmitting products is given. Research papers on mobile video recording and transmitting and on GUI design for mobile systems are presented.

The design chapter starts with an analysis on how the functionality of the hardware can be accessed by the software. Together with the results of an evaluation of the user requirements, an object oriented design of the application is derived. The design is split up into four major parts: a library to access the functionality of the video and audio hardware, a class to handle the GPS device, a class for communication with the server, and the GUI.

The implementation of the application is done in C++ using the Qt library. Video and audio hardware is accessed through the GStreamer library.

Finally an overview of the results achieved is given. The quality of video transmissions over different communication paths using different transmission protocols is compared. The power consumption of the embedded system is measured and evaluated for different states of operation.

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz,am ....

(Unterschrift)

Danksagung

Diese Masterarbeit wurde in den Jahren 2010 und 2011 bei der Firma Spintower und am Institut für Technische Informatik an der Technischen Universität Graz durchgeführt.

Ich danke meinem Chef bei der Firma Spintower, Herrn Dipl.-Ing. Mario Schwaiger, und Herrn Ass.Prof. Dipl.-Ing. Dr. techn. Christian Steger vom Institut für Technische Informatik an der TU Graz, die es mir ermöglicht haben, diese Arbeit durchführen zu können.

Großen Dank meiner Familie und meiner Freundin Johanna, die mich immer unterstützt haben. Ganz besonders möchte ich mich bei meinen Eltern bedanken, die mir dieses Studium ermöglicht haben.

Vielen Dank!

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

1.1. Motivation

Diese Masterarbeit wurde im Rahmen meiner Tätigkeit bei der Grazer Firma Spintower erstellt. Die Firma Spintower beschäftigt sich mit der Entwicklung von stromsparenden Hardwaresystemen und der dazupassenden Software. Die Palette der entwickelten Systeme reicht von mobilen Devices zur Übertragung von Ton- und Videodaten, über energieautarke Kiosksysteme, bis hin zu besonders Kompakten Datenloggern zur Aufzeichnung und Übertragung verschiedenster Sensordaten.

Eine Entwicklung der Firma ist der Mobile Field Assistant (MFA, siehe Abbildung 1.1). Der MFA ist ein stromsparendes, kompaktes, am Körper tragbares Gerät. Er soll Mitarbeitern im Außeneinsatz die drahtlos Kommunikation mit einer Zentrale ermöglichen. Dabei können verschiedenste, vom MFA aufgezeichnete Sensordaten übertragen werden, etwa über das Kamera-Interface des MFA aufgezeichnete Videobilder, über das Mikrophon des MFA aufgenommene Tonsignale oder über den GPS-Sensor des MFA aufgezeichnete Positionsdaten. Die Bedienung des MFA erfolgt über den Touchscreen des integrierten Bildschirms.

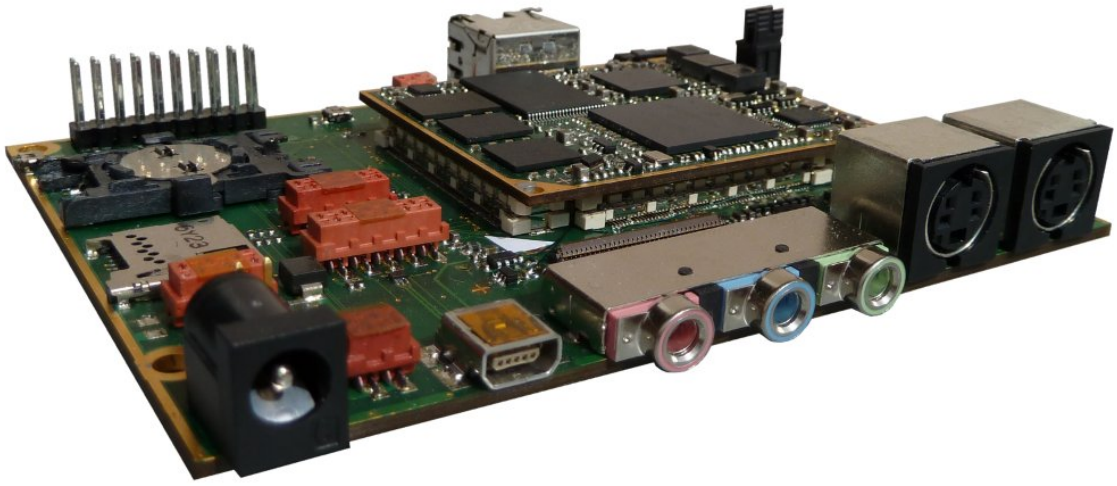


Abbildung 1.1.: Die Hardware des Mobile Field Assistant [Spintower].

1.2. Zielsetzung

Ziel der Masterarbeit ist es, eine Software-Anwendung für den MFA zu erstellen. Die Anwendung soll Ton- und Videodaten aufzeichnen und gleichzeitig live über eine Netzwerkverbindung an einen Server senden können. Auch die Daten des GPS-Sensors sollen aufgezeichnet und an den Server übertragen werden können. Zudem soll eine grafische Benutzeroberfläche erstellt werden, über die der Anwender die Funktionen des MFA steuern kann. Die Bedienung der Benutzeroberfläche soll über den Touchscreen des MFA erfolgen und möglichst intuitiv sein.

1.3. Gliederung

Die Masterarbeit beginnt mit einer kurzen, einleitenden Beschreibung der Arbeit. Es folgte eine Analyse des State of the Art. Die Eigenschaften des MFA werden mit zu Projekstart und aktuell verfügbaren Hardwaresystemen verglichen. Forschungsprojekte, die sich dem Thema Liveübertragung von Ton- und Videodaten widmen, werden zusammengefasst. Die Funktionen ähnlicher, kommerzieller Produkte werden analysiert um eine Übersicht über den aktuellen Stand der Technik zu erlangen.

Im Design-Kapitel werden die von der Anwendung zu erfüllenden Ziele identifiziert. Darauf folgt eine Analyse der Eigenschaften der zur Verfügung stehenden Hardware. Aus den Zielen und den Hardwareeigenschaften wird das Design der Software-Architektur erstellt.

Die Implementierung der Software-Architektur wird im Kaptiel Implementierung beschrieben. Darauf folgt eine Zusammenfassung der erreichten Ziele, Einschränkungen und Verbesserungsmöglichkeiten.

Schlußendlich folgt ein kurzes Résumé über die Durchführung der Masterarbeit.

2. State of the Art

2.1. Embedded Hardware Plattformen

Aufgrund des immer größer werdenden Marktes für Smartphones und Tablet PCs, ist die technische Entwicklung im Bereich mobiler Hardwareplattformen in den letzten Jahren rasant vorangeschritten. Zahlreiche Hersteller bieten System-On-Chip (SoC) oder Application Processors an, die einerseits eine hohe Rechenleistung bieten, andererseits aber dennoch sehr energieeffizient arbeiten. Zumeist bestehen diese SoC aus einer ARM-CPU, die um zahlreiche Co-Prozessoren, Digitale Signal Prozessoren (DSP) und I/O-Controller ergänzt wird. Die DSPs übernehmen Spezialaufgaben wie die Kodierung und Dekodierung von Videos oder die Ver- und Entschlüsselung von Daten. Die ARM-CPU führt das Betriebssystem und die darauf laufenden Programme aus. Diese evaluieren die Eingaben des Benutzers und setzen sie in Kommandos zur Steuerung der DSPs und Schnittstellen um. SoC sind modular aufgebaut. Nicht benötigte Komponenten können temporär deaktiviert werden. Zum Beispiel kann der Videodecoder abgeschaltet werden, wenn kein Video abgespielt wird. Dadurch, und mit Hilfe weiterer Stromspartechniken [1], wie dynamischer Spannungsregulierung oder durch Ausnutzung verschiedener CPU-Schlafzustände, ist es möglich sehr energieeffiziente Systeme zu erstellen.

Eine Hardwareplattform, die als Basis für dieses Projekt in Frage kommt, muß zumindest folgende Anforderungen erfüllen:

- Geringer Stromverbrauch
- Schnittstelle für eine externe Kamera
- Hardwareunterstützung für die Kompression und Dekompression von Videodaten
- Unterstützung des H.264-Videocodec
- Unterstützung des Linux Betriebssystems

Zudem musste die gewählte Hardware in absehbarer Zeit nach Start des Projektes (2007-2008) am Markt verfügbar sein.

Unter Berücksichtigung dieser Anforderungen, schränkt sich die große Zahl an SoC auf eine überschaubare Menge ein. Im Folgenden werden die in Frage kommenden Systeme der drei großen Chiphersteller Texas Instruments, Samsung und Freescale (zuvor Motorola) einander gegenübergestellt. Danach folgt ein Vergleich aktueller Systeme, die zu Projektstart noch nicht verfügbar waren. Dies soll den technischen Fortschritt auf diesem Gebiet verdeutlichen und ist, im Hinblick auf die künftige Durchführung weiterer, ähnlicher Projekte von Interesse.

2.1.1. Zu Projekbeginn verfügbare Systeme

Zum Zeitpunkt des Projektstarts war die Anzahl an für das Projekt in Frage kommenden SoC noch relativ gering. Von Texas Instruments kommen drei Systeme in Frage, von Freescale und Samsung jeweils eines. Folgende Systeme werden verglichen:

- i.MX27, Freescale
- TMS320DM6441, Texas Instruments
- TMS320DM6446, Texas Instruments
- TMS320DM6467, Texas Instruments
- S3C6410, Samsung

Systembewertung

Die Bewertung der SoC erfolgt anhand des Vergleichs verschiedener Systemeigenschaften. Weist ein SoC bei einer Systemeigenschaft große Mängel gegenüber den anderen SoC auf, wird dies mit -4 Punkten bewertet, mittel schwere Mängel werden mit -2 Punkten bewertet, kleine Mängel mit -1 Punkt. Große Mängel gefährden die Durchführbarkeit des Projekts stark. Mittlere Mängel beeinträchtigen die Durchführbarkeit des Projekts nicht, der Aufwand um sie auszugleichen ist allerdings nicht vernachlässigbar. Kleine Mängel weist eine Systemeigenschaft dann auf, wenn sie zwar im Vergleich zur selben Eigenschaft der anderen SoC schlechter ist, aber die Anforderungen zur Durchführung des Projekts dennoch erfüllt.

Die Eigenschaften werden in Form von Tabellen gegenübergestellt und bewertet. In den Tabellen werden große Mängel mit einem roten Zellenhintergrund dargestellt, mittlere Mängel mit einem orangefarbenen und kleine Mängel mit einem gelben.

Folgende Systemeigenschaften werden verglichen:

- Verfügbarkeitsdatum
- Preis
- CPU-Typ und -Frequenz
- Kernspannung der CPU
- Fertigungstechnologie
- Instruktions- und Datencache
- Verfügbarkeit eines DSP
- Anzahl an I^2C-, USB-, UART-, MMC/SD-Schnittstellen
- Verfügbarkeit eines Verschlüsselungs-Co-Prozessors
- Maximale Bildauflösung der H.264-Kodierer/Dekodierer
- Bildauflösung der Kameraschnittstelle

- Schnittstelle zur Videoausgabe
- Verfügbarkeit eines 3D-Grafikbeschleunigers

Vergleich von Verfügbarkeitsdatum, Preis und CPU-Eigenschaften

In Tabelle 2.1 sind Datum der Verfügbarkeit, Preis und die Eigenschaften der CPUs der SoC angeführt. Das Datum beschreibt jenes Jahr, in dem das SoC vom Hersteller für den Verkauf angekündigt wurde. Hierbei ist zu beachten, daß dies nicht zwangsläufig jenes Jahr ist, seit dem das SoC in großen Zahlen tatsächlich am Markt verfügbar ist. Für dieses Projekt sind jene Systeme interessant, die in den Jahren 2007 oder 2008 angekündigt wurden. Die Preise der SoC wurden, falls nicht anders angegeben, am 30.9.2010 von den Webseiten der Hersteller übernommen. Natürlich ist ein möglichst kleiner Preis wünschenswert. Die CPU-Eigenschaften der SoC unterscheiden sich vor allem in Frequenz, Fertigungstechnologie und der Größe des Datencache. Die Kernspannung der CPU hängt vorallem von der Fertigungstechnologie und der CPU-Frequenz ab. Da die Verarbeitung der Videodaten vom H.264-Kodierer durchgeführt wird, muß die ARM-CPU gerade schnell genug sein um Audiodaten zu kodieren und zu dekodieren, eine reaktive Benutzeroberfläche darzustellen, und Sensordaten, zum Beispiel die eines GPS-Empfängers, aufzunehmen und für den Transport über eine Netzwerkverbindung vorzubereiten.

	Verfügbar [Jahr]	Preis [$]	CPU	CPU Frequenz [Mhz]	Kern-Spannung [V]	Fertigungs-Technologie [nm]	Instruktions Cache [kB]	Daten Cache [kB]
TMS320 DM6441	2007	30.35	ARM9	256 202.5	1.2 1.05	90	16	8
TMS320 DM6446	2006	35.63	ARM9	405 297 256	1.3 1.2 1.2	90	16	8
TMS320 DM6467	2007 (1GHz 2010)	83.53	ARM9	364.5 297	1.2 1.05	90	16	8
i.mx27	2007	14.86	ARM9	400	1.2-1.5	90	16	16
S3C6410	2008		ARM11	800 667 533	1.3 1.2 1.1	65	16	16

Tabelle 2.1.: Die CPUs der SoC im Vergleich

Aus den Eigenschaften der einzelnen SoC ergibt sich folgende Bewertung:

- Verfügbar: Das späte Verfügbarkeitsdatum des S3C6410 ist dessen größter Nachteil.
- Preis: Die Preise der TI SoC sind im Vergleich zu dem des Freescale SoC relativ hoch. Vorallem der TMS320DM6467 ist aufgrund seiner umfangreichen Ausstattung an Videoverarbeitungseinheiten (siehe Tabelle 2.3) sehr teuer.
- CPU: Der modernere S3C6410 verfügt mit dem ARM11 auch über die modernste CPU im Vergleich.

- CPU-Frequenz: Nur der ARM9 des TMS320DM6441 weist mit max. 256 MHz eine sehr geringe Taktfrequenz auf.
- Kernspannung: Diese bewegt sich bei allen Kandidaten, abhängig von der Taktfrequenz, in einem ähnlichen Bereich.
- Instruktions-Cache: Die CPUs aller SoC verfügen über 16 KB an Instruktions-Cache
- Daten-Cache: Die SoC von TI verfügen mit 8 KB nur über halb so viel Daten-Cache, wie ihre Konkurrenten.

Aus dem Vergleich der CPU-Eigenschaften würde der modernere S3C6410 als klarer Sieger hervorgehen. Aufgrund des späten Verfügbarkeitsdatums ist allerdings dem i.MX27 der Vorzug zu geben.

Vergleich von Co-Prozessoren und Interface-Controllern

In Tabelle 2.2 sind die DSPs, Schnittstellen und Verschlüsselungs-Co-Prozessoren der SoC angegeben. Die Spalte DSP zeigt, ob das SoC über einen programmierbaren DSP verfügt. Dieser ist für Videoverarbeitung, etwa eine Gesichtserkennung [2], sehr nützlich. Die Spalten I2C, USB, UART, MMC/SD geben an, wieviele Schnittstellen des jeweiligen Typs die Systeme ansprechen können. Besonders MMC/SD-Card-Interfaces sind sehr nützlich, da man das System über Speicherkarte individuell konfigurieren kann. Zudem dienen sie als Speicherort für aufgezeichnete Video- und Audiodaten. Auch das Betriebssystem, kann auf der Karte gespeichert, und von dieser gestartet werden. In der Spalte Sicherheit sind Co-Prozessoren zur Ver- und Entschlüsselung von Daten angeführt.

	DSP Frequenz [MHz]	I2C	USB	UART	MMC/SD	Sicherheit
TMS320 DM6441	TMS320C64x+ 513 405	1	1	3	1	
TMS320 DM6446	TMS320C64x+ 810 594 513	1	1	3	1	
TMS320 DM6467	TMS320C64x+ 729 594	1	1	3	0	
i.mx27		2	2	6	3	SAHARA2 AES, DES, 3DES MD5, SHA-1/224/256 RNG
S3C6410		2	2	4	3	Crypto Accelerator AES, DES, 3DES MD5, SHA-1 RNG

Tabelle 2.2.: Die Schnittstellen der SoC im Vergleich

Aus dem Vergleich der Eigenschaften in Tabelle 2.2 ergibt sich folgende Bewertung der Systeme:

- DSP: Die SoC von TI verfügen über eigene DSPs. Diese sind bei den Systemen von Freescale und Samsung nicht vorhanden.
- I2C, USB, UART, MMC/SD: Die Anzahl an Schnittstellen ist bei den SoC von TI gering. Der TMS320DM6467 besitzt keine MMC-Schnittstelle.
- Sicherheit: Nur der i.MX27 und der S3C6410 verfügen über eigene Co-Prozessoren zur Ver- und Entschlüsselung von Daten.

Auch aus dem Vergleich der Co-Prozessoren und Interface-Controller gehen der i.MX27 und der S3C6410 als Sieger hervor. Deren größter Nachteil ist das Fehlen eines programmierbaren DSP, der allerdings nicht unbedingt für die Durchführung des Projekts notwendig ist. Die SoC von TI verfügen über weniger Schnittstellen, dem TMS320DM6467 fehlt zudem die Schnittstelle für eine Speicherkarte. Außerdem fehlt den Systemen von TI ein Crypto-Co-Prozessor, weshalb die rechenintensive Ver- und Entschlüsselung von Daten vom DSP oder der ARM-CPU durchgeführt werden muß.

Vergleich der Videoeigenschaften

Tabelle 2.3 zeigt die Videoeigenschaften der SoC. In den Spalten „H.264 Kodieren" und „H.264 Dekodieren" ist die maximale Bildauflösung angegeben, die vom H.264-Co-Prozessor bei einer Bildwiederholrate von 30 Bildern pro Sekunde verarbeitet werden kann. Hierbei steht „D1" für eine Auflösung von horizontal 720 und vertikal 576 Pixeln, 1080p für horizontal 1920 und vertikal 1080 Pixel. Die Spalte „Kameraschnittstelle" listet die unterstützen Formate der Kameraschnittstellen auf. Die Kürzel BT.601, BT.656 und BT.1120 stehen für die von der International Telecommunication Union (ITU) festgelegten Empfehlungen [3]. In „Video Ausgabe" findet man die verfügbaren Videoausgabe-Schnittstellen. Ob die SoC über Beschleuniger für 3D-Grafik verfügen und welche 3D-Standards unterstützt werden, ist der Spalte „3D" zu entnehmen.

Aus dem Vergleich der Eigenschaften in Tabelle 2.3 ergibt sich folgende Bewertung der Systeme:

- H.264 Kodieren: Alle System-On-Chip erfüllen die Mindestanforderung von D1. Der TMS320DM6467 unterstützt auch High Definition (HD) Video.
- H.264 Dekodieren: Auch hier erfüllen alles Systeme die Mindestanforderung.
- Kameraschnittstelle und Videoausgabe: Abgesehen vom TMS320DM6467 bieten alle SoC eine LCD-Schnittstelle.
- 3D: Nur der S3C6410 verfügt als einziges SoC über einen 3D-Grafikbeschleuniger.

Der TMS320Dm6467 weist, dank der Unterstützung für HD-Video, überragende Videoeigenschaften im Vergleich zu den anderen SoC auf. Diesen Vorteil kann er aber nicht wirklich ausspielen, da HD-Video für dieses Projekt nicht von Bedeutung ist. Die Mindestanforderungen an die Videoeigenschaften werden von allen SoC erfüllt.

	H.264 Kodieren [max. Auflösung]	H.264 Dekodieren [max. Auflösung]	Kamera-Schnittstelle	Video Ausgabe	3D
TMS320 DM6441	D1	D1	BT.601/656	NTSC/PAL S-Video BT.601/656 LCD	
TMS320 DM6446	D1	D1	BT.601/656	NTSC/PAL S-Video BT.601/656 LCD	
TMS320 DM6467	1080p	1080p	2 x BT.656 oder 1 x BT.1120	2 x BT.656 oder 1 x BT.1120	
i.mx27	D1	D1	BT.601/656	LCD 800x600	
S3C6410	D1	D1	BT.601/656	NTSC/PAL BT.601/656 LCD 1024x768	OpenGL ES

Tabelle 2.3.: Die Videoeigenschaften der SoC im Vergleich

Gesamtwertung

Der Vergleich der Systeme ergibt das in Tabelle 2.4 gezeigte Gesamtergebnis. Das SoC i.MX27 der Firma Freescale eignet sich am besten und wurde als Hardwarebasis für das Projekt gewählt.

Platzierung	System	Kleine Mängel	Mittlere Mängel	Große Mängel	Wertung*
1	i.mx27	5	1	0	-7
2	S3C6410	2	1	1	-8
3	TMS320 DM6446	11	1	0	-13
4	TMS320 DM6441	12	1	0	-14
5	TMS320 DM6467	7	1	2	-17
*Punkteverteilung: -1 Punkt pro kleinem Mangel, -2 Punkte pro mittlerem Mangel, -4 Punkte pro großem Mangel					

Tabelle 2.4.: Bewertung der Systeme

2.1.2. Aktuelle Systeme

Seit Projektstart hat sich die Technologie im Bereich der SoC rasant weiterentwickelt. Insbesonders neue Prozessoren der Firma ARM finden bei der Entwicklung neuer Systemen oft Verwendung. Die Unterstützung von HD-Video bei der Aufzeichnung und Wiedergabe von Videos ist nun Standard. Es folgt eine tabellarische Übersicht aktueller Systeme.

Besonders interessant ist der S5PC110 der Firma Samsung, der unter anderem im Apple iPhone 4, im iPad und im Samsung Galaxy S Smartphone Verwendung finden soll.

Vergleich der CPU-Eigenschaften

Wie in Tabelle 2.5 ersichtlich, wird in aktuellen SoC meist eine ARM Cortex-A8 CPU eingesetzt. Nur TI verwendet beim TMS320DM365 die ältere ARM9 CPU. Die Größe von Instruktions- und Daten-Cache hat sich im Vergleich zur Vorgängergeneration großteils von 16 KB auf 32 KB verdoppelt. Der Fertigungsprozess ist von 90 nm auf 65 nm oder 45 nm umgestellt worden. Die Frequenz der CPUs hat sich auf 800 MHz bis 1 GHz gesteigert, die Kernspannung ist unverändert.

	Verfügbar [Jahr]	Preis [$]	CPU	CPU Frequenz [Mhz]	Kern-Spannung [V]	Fertigungs-Technologie [nm]	Instruktions Cache [kB]	Daten Cache [kB]
DM3725 DM3730	2010	22.65 25.6	Cortex-A8	1000 800	0.9-1.2	45	32	32
OMAP3525 OMAP3530	2010	37.5 41.7	Cortex-A8	720	0.8-1.35	65	16	16
TMS320 DM365	2009	25.05	ARM9	300 270 216	1.35 1.2 1.2	65	16	8
I.mx513 I.mx515	2009		Cortex-A8	800	1.15	65	32	32
S5PC100	2009		Cortex-A8	667	1.2		32	32
S5PC110	2009		Cortex-A8	1000 800	1.25 1.2	45	32	32

Tabelle 2.5.: Die CPUs aktueller SoC im Vergleich

Vergleich von Co-Prozessoren und Interface-Controllern

Einen Vergleich der Anzahl an Schnittstellen und verfügbarer Co-Prozessoren ist in Tabelle 2.6 enthalten. Man kann feststellen, daß die aktuelle Generation von SoC mehr Schnittstellen eines jeweiligen Typs bietet, als die Vorgängergeneration. Die SoC von TI enthalten nach wie vor einen programmierbaren DSP.

Vergleich der Videoeigenschaften

Die meisten der aktuellen SoC bieten Unterstützung für HD-Video, wie aus Tabelle 2.7 hervorgeht. Dies trifft besonders auf die Ausgabeformate der Videoschnittstellen zu, bei den Kameraschnittstellen hat sich HD-Video noch nicht voll durchgesetzt. Nahezu alle SoC besitzen nun auch einen 3D-Grafikbeschleuniger, manche auch einen 2D-Vektorgrafik-Beschleuniger.

	DSP Frequenz [MHz]	I2C	USB	UART	MMC/SD	Sicherheit
DM3725 DM3730	TMS320C64x+ 800 600	4	4	4	3	
OMAP3525 OMAP3530	TMS320C64x+ 520	3	2	3	3	
TMS320 DM365		1	1	2	2	
I.mx513 I.mx515		3	3	3	4	SAHARA Lite AES, DES, 3DES, RC4 MD5, SHA-1/224/256 RNG
S5PC100		2	2	4	3	Crypto Engine
S5PC110		3	2	4	4	Crypto Engine

Tabelle 2.6.: Die Schnittstellen aktueller SoC im Vergleich

	H.264 Kodieren [max. Auflösung]	H.264 Dekodieren [max. Auflösung]	Kamera-Schnittstelle	Video Ausgabe	3D
DM3725 DM3730	D1	720p	BT.601/656	NTSC/PAL S-Video BT.656 LCD 1080i 720p	DM3730 POWERVR SGX OpenGL ES
OMAP3525 OMAP3530	D1	D1	BT.601/656	NTSC/PAL S-Video BT.656 LCD 1080i 720p	OMAP3530 POWERVR SGX OpenGL ES
TMS320 DM365	720p	720p	BT.601/656/1120	NTSC/PAL BT.601/656 1080i analog LCD 1080i 720p	
I.mx513 I.mx515	D1	720p	2 x BT.656/1120	Composite S-Video Component 1080i, 720p LCD 1280x800	I.MX515 OpenGL ES OpenVG
S5PC100	720p	720p	BT.601/656	NTSC/PAL HDMI LCD 1024x768	OpenGL ES OpenVG
S5PC110	1080p	1080p	BT.601/656	NTSC/PAL HDMI LCD 1280x1024	OpenGL ES OpenVG

Tabelle 2.7.: Die Videoeigenschaften aktueller SoC im Vergleich

2.2. Forschungsprojekte

2.2.1. Entwicklung eines WLAN-Videoübertragungssystems

In ihrer Arbeit [4] beschreiben Bin Fan, Lianfeng Shen und Tiecheng Song vom National Mobile Communications Research Laboratory Southeast der University Nanjing, China, das Design und die Implementierung eines Systems zur Übertragung von Videos über WLAN.

Als besonders wichtige Eigenschaft einer Videoübertragung wird deren Echtzeitfähigkeit identifiziert. Viele Einsatzgebiete, wie zum Beispiel militärische Anwendungen, erfordern eine geringe End-zu-End Latenz der Datenübertragung. Praktisch ist eine kleine Latenz aber aufgrund beschränkter Bandbreite, langsamer Kodierung oder Problemen bei der Steuerung der Datenübertragung nur schwer zu erreichen, siehe [5], [6], [7].

Das von Fan und dessen Kollegen untersuchte Szenario besteht aus zwei Embedded Linux Systemen, die jeweils eine Empfangs- und Sendeeinheit besitzen. Eines der Systeme verfügt über eine Kamera, deren Bilder kodiert über WLAN zum zweiten System gesendet werden. Das zweite System dekodiert die empfangenen Videobilder und stellt sie auf einem LCD dar. Zur Übertragung der Videodaten wird das Real-time Transport Protocol (RTP) verwendet. Die Ursachen einer zu hohe Latenz der Videoübertragung in einem solchen Szenario sind mannigfaltig. Um ein Bild von der Kamera des Senders bis auf den Bildschirm des Empfängers zu übertragen, sind viele verschiedene Arbeitsschritte notwendig, von denen jeder einen Teil zur Gesamtlatenz beiträgt. Desto kleiner die einzelnen Teillatenzen sind, umso kleiner wird auch die Gesamtlatenz der Videoübertragung sein.

Die Hardware, die Fan und seine Kollegen für deren Untersuchungen nutzen, basiert, wie der MFA, auf dem i.MX27 der Firma Freescale. Die Eigenschaften des i.MX27 sind in Abschnitt 2.1.1 näher erläutert. In Abbildung 2.1 ist der von Fan verwendete Hardwareaufbau ersichtlich.

Nach dem Start des Systems, initialisiert der i.MX27 die Systemkonfiguration, erstellt die WLAN-Verbindung und startet die Videoaufnahme. Die Videodaten werden vom MPEG-4-Application-Specific Integrated Circuit (ASIC) in das MPEG-4 Format kodiert. Jedes kodierte Bild wird vom i.MX27 in ein RTP-Paket gepackt und vom 802.11g-Funkmodul versendet. Beim Empfänger findet der umgekehrte Vorgang statt, der mit der Darstellung der Videobilder auf dem LCD endet.

Die von Fan et al. erstellte Softwarearchitektur ist in Abbildung 2.2 dargestellt. Auf oberster Ebene läuft die Anwendung, die aus den Videodaten RTP- und Real-time Transport Control-Protocol (RTCP)-Pakete erstellt. RTCP-Pakete werden zwischen Sender und Empfänger ausgetauscht. Sie enthalten Informationen zur Teilnehmeridentifizierung, Datenstromsynchronisation und Quality of Service (QoS). Die RTP- und RTCP-Pakete werden in UDP-Pakete verpackt und an den WLAN-Treiber übergeben. Der WLAN-Treiber ist Teil des Linux-Kernels, der die Schnittstelle zwischen Software und Hardware bildet.

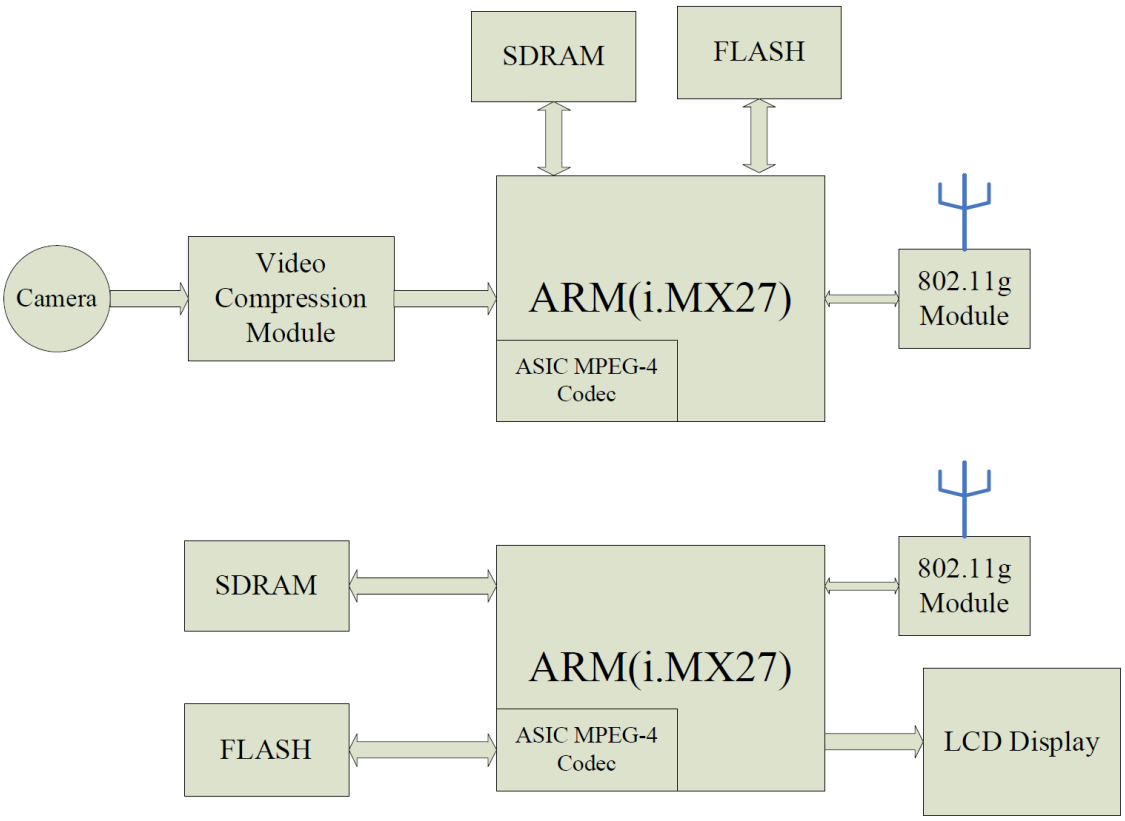


Abbildung 2.1.: Hardwareaufbau in Fans Untersuchungen [4, S. 685]

Ziel von Fan et al. ist es, die Latenz der Videoübertragung möglichst zu minimieren. Durch Analyse des Zeitverhaltens der einzelnen, für die Videoübertragung notwendigen, Arbeitsschritte, konnten drei Maßnahmen zur Reduzierung der Gesamtverzögerung gefunden werden.

1. Optimierung des MPEG-4 Codec:
   Um eine möglichst hohe Datenkompression zu erreichen, verwendet der MPEG-4 Algorithmus, neben I-Bilder und P-Bilder, sogenannte B-Bilder. Ein B-Bild ist ein Einzelbild eines Videos, das durch die Differenz von vorhergehenden und nachfolgenden Einzelbildern beschrieben wird. Bevor ein B-Bild dekodiert werden kann, müssen erst alle nachfolgenden Bilder, auf die sich das B-Bild bezieht, empfangen worden sein. Die Verwendung von B-Bilder erhöht die Latenz um mindestens das Zweifache im Vergleich zur Kodierung ohne B-Bildern. Auch beim H.264-Algorithmus führt die Verwendung von B-Bildern zu wenigstens einer Verdopplung der Latenz [8]. Im MPEG-4-Standard ist ein Verfahren zur Kodierung und Dekodierung von Videos ohne B-Bilder definiert. Es wird als MPEG-4 Simple Profile (MPEG-4 SP) bezeichnet [9]. Der MPEG-4 Videokodierer des i.MX27 unterstützt dieses Profil und muß mit diesem konfiguriert werden, um die Latenz der Videoübertragung zu minimieren.

   Zudem schlagen Fan et al. vor, die Anzahl der Intra-Bilder (I-Bilder) auf eines pro

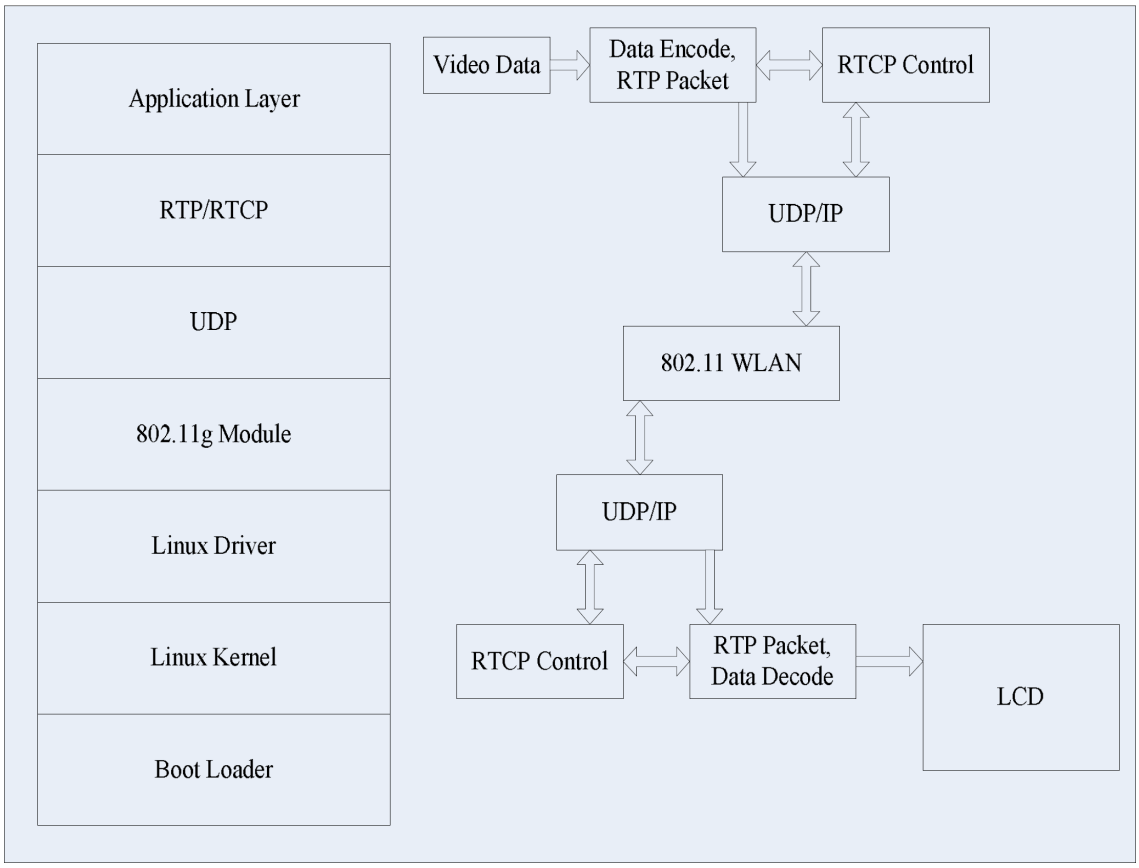


Abbildung 2.2.: Fans Softwarearchitektur [4, S. 685]

Bildergruppe (engl. Group of Pictures, GOP) zu beschränken. I-Bilder können, im Gegensatz zu den zuvor erwähnten B-Bildern, unabhängig von vorhergehenden oder nachfolgenden Bildern dekodiert werden. Sie beschreiben ein vollständiges Einzelbild und sind daher größer als B- oder P-Bilder. Eine Reduzierung der I-Bilder reduziert somit auch die für die Videoübertragung notwendige Bandbreite.

2. Wahl eines geeigneten Transportprotokolls:
   Aufgrund der Eigenschaften von TCP, beim Aufbau einer Verbindung einen 3-Wege-Handshake durchzuführen, sowie Datenverluste automatisch zu erkennen und zu korrigieren, kann es beim Verbindungsaufbau und beim Empfang von Datenpaketen zu Verzögerungen kommen. Daher verwenden Fan et al. statt TCP das für die Videoübertragung besser geeignete RTP auf Basis von UDP.

3. Maximierung der Bandbreite:
   Eine weitere Maßnahme zur Reduzierung der Latenz ist, die verfügbare Bandbreite vollständig auszunutzen. Fan et al. erreichen dies in ihrem System durch Verwendung des 802.11g WLAN-Standards anstatt des langsameren 802.11b.

2.2.2. Implementierung eines Embedded H.264 Live Video Streaming Systems

In ihrer Arbeit [10] beschreiben Vun und Ansary die Implementierung eines Streaming Systems zur Liveübertragung von H.264-kodierten Videos. Basis ihres Systems ist ein DM6446 DVEVM Evaluierungsboard- der Firma Texas Instruments (siehe Abbildung 2.3). In Abschnitt 2.1.1 sind die Eigenschaften des DM6446 zusammengefasst.

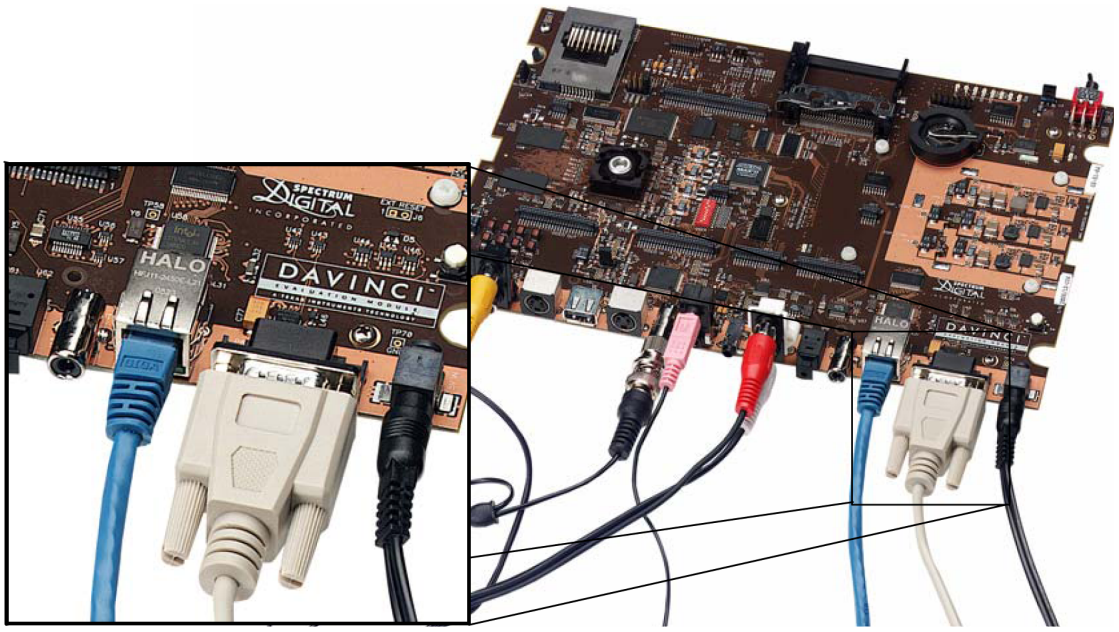


Abbildung 2.3.: DaVinci Video Evaluation Module, [11, S. 2-5]

Zunächst erläutern Vun und Ansary die Videostreaming Protokolle, die in ihrem System zum Einsatz kommen. Diese sind das Real-Time Streaming Protocol (RTSP), RTP, sowie das Session Description Protocol (SDP).

RTSP: Das Real-Time Streaming Protocol [12] dient zur Steuerung des Medienservers. Es ermöglicht einem Client, eine Verbindung zu einem Medienserver aufzubauen, und von diesem Videodaten anzufordern. Die Kommunikation zwischen Client und Server erfolgt, ähnlich dem Hyper Text Transfer Protokol (HTTP), über Befehle in Textform. Die Übertragung der Mediendaten selbst erfolgt nicht per RTSP, hierfür verwenden Vun und Ansary das RTP.

RTP: Das Real-time Transport Protocol [13] beschreibt den Aufbau eines Netzwerkpaketes, das Multimediadaten enthält. Jedes RTP-Paket beginnt mit einem Header. Dieser enthält Informationen über das zur Kodierung der Daten verwendete Format, einen Zeitstempel sowie eine fortlaufende Nummer. Diese ermöglicht es dem Empfänger die Reihenfolge der Datenpakete zu rekonstruieren, auch wenn zum Transport der RTP-Pakete ein

Protokoll verwendet wurde, das keine Garantien über die Empfangsreihenfolge der Pakete abgibt (zum Beispiel UDP).

SDP: Das Session Description Protocol [14] beschreibt, wie Empfänger und Sender eine Sitzung zum Austausch von Multimediadaten aufbauen. Eine Sitzung besteht aus einer Menge von Sendern und Empfängern, sowie einem oder mehreren Datenströmen, die von Sender zu Empfänger übertragen werden. Das SDP beschränkt sich allein auf die Beschreibung des Verbindungsaufbaus und der Medienformate. Die Mediendaten selbst werden, wie zuvor beschrieben, per RTP übertragen. Wie die Beschreibung einer Sitzung zum Empfänger übertragen wird, ist nicht vorgegeben. Die Beschreibung kann beispielsweise als MIME-kodierter Anhang einer E-Mail verschickt werden, per HTTP von einem Web-Server empfangen werden, oder, wie in der Arbeit von Vun und Ansary, als Teil des RTSP übertragen werden.

Der von Vun und Ansary verwendete RTSP-Server ist der Live555 Media Server. Dieser implementiert das RTSP, RTP sowie SDP, und ist mit gängigen Medienspielern wie zum Beispiel QuickTime und VLC kompatibel. Der Originalversion des Servers weist allerdings zwei Einschränkungen auf:

- Keine Unterstützung für H.264.
- Keine ausführbare Version für die DaVinci-Plattform.

Glücklicherweise ist der Quelltext des Servers frei zugänglich und beliebig erweiterbar. Für Vun und Ansary war es daher problemlos möglich, den Server um die fehlende H.264 Unterstützung zu erweitern und eine auf der DaVinci-Plattform ausführbare Version des Servers zu kompilieren.

In Abbildung 2.4 ist das Design der Video-Streaming-Anwendung dargestellt. Die Anwendung besteht aus zwei Prozessen, die von der ARM-CPU ausgeführt werden. Das „Video encoding Program" (VeP) führt die Kodierung des Videos durch. Der Live555 Server macht das kodierte Video Medienspielern per RTSP zugänglich. Die Kommunikation zwischen den Prozessen erfolgt über einen UDP-Socket.

Das VeP basiert auf einem Demo-Programm für das Evaluation Board. Es steuert den DSP des DM6446, der die eigentliche Kodierung der Videobilder in das H.264-Format durchführt. Der Datenaustausch zwischen DSP und VeP erfolgt über einen gemeinsam genutzten Speicherbereich (shared memory). In diesem legt der DSP fertig kodierte Videobilder ab, aus dem sie das VeP ausliest und über den UDP-Socket an den Live555 Media Server überträgt. Der Server kapselt die kodierten Videodaten in RTP-Pakete und liefert sie über eine Netzwerkverbindung an Medienspieler aus.

Die Aufteilung der Anwendung in zwei getrennte Prozesse, die über einen UDP-Socket mit einander kommunizieren müssen, begründen Vun und Ansary damit, den Live555-Server und das Demo-Programm so nur wenig verändern zu müssen. Eine effizientere Version der Anwendung könnte alle Funktionen in einem Prozess zusammenfassen, wodurch der zum Datenaustausch notwendige Umweg über den Socket wegfallen würde.

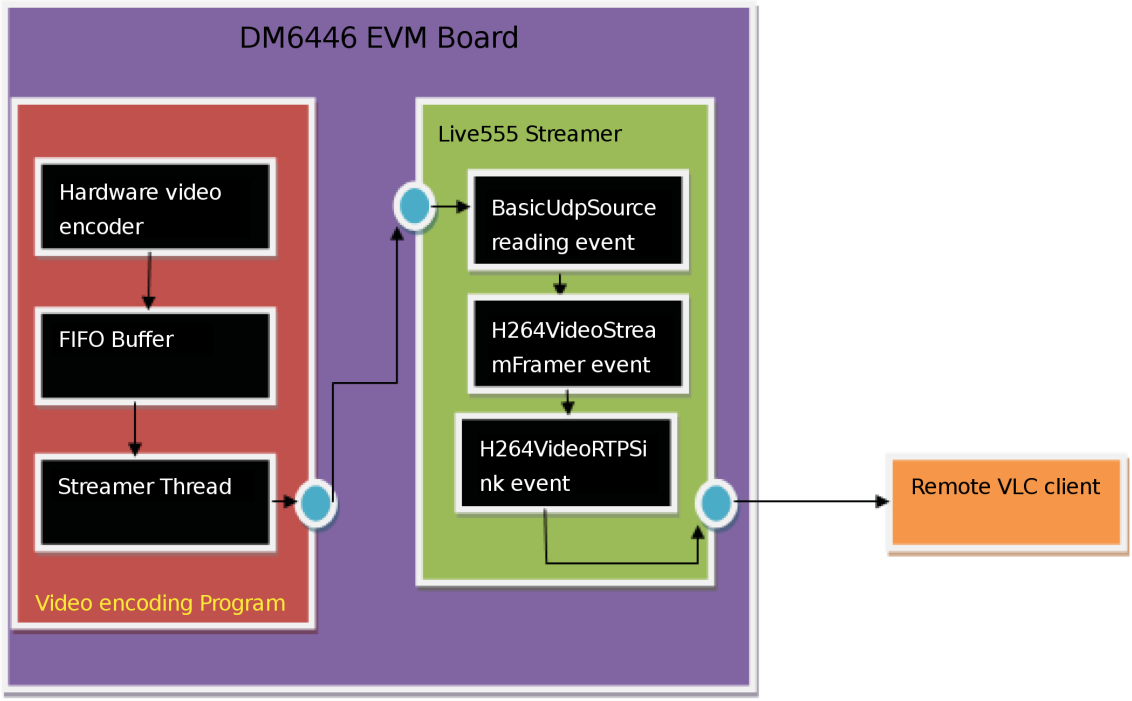


Abbildung 2.4.: Kommunikation zwischen H.264-Kodierer und Live555-Server [10, S. 4]

2.2.3. Embedded Videoüberwachungssystem

In ihrer Arbeit [15] beschreiben Gao et al. die Implementierung eines auf H.264 basierenden Videoüberwachungssystems. Den Kern der von ihnen verwendete Hardware bilden zwei Blackfin-Prozessoren, der ADSP-BF561 und der ADSP-BF533, die, zusammen mit SDRAM, Kamera, Display, Video-Encoder/Decodern und weiteren Komponenten, einen Funktionsumfang entsprechend dem Freescale i.MX27 bieten. Hinzu kommt noch eine PTZ-Steuerung für die bewegliche Kamera. Der BF561 übernimmt die Kodierung der H.264-Videos, während der BF533 für die Netzwerkübertragung, die PTZ-Steuerung und das Basissystem verantwortlich ist.

Laut Gao et al. würde der Einsatz von TCP zur Übertragung der Videodaten einen großen Overhead erzeugen. Daher, und da sich der Verlust einzelner Datenpakete nur sehr kurzfristig auf die Qualität der Videobilder auswirkt, wird UDP zur Übertragung der Videodaten verwendet. Um die Geschwindigkeit des H.264-Encoders zu beschleunigen, wird die Auflösung der Kamerabilder vor der Kompression von 720 mal 576 Pixeln auf 176 mal 144 Pixel reduziert. Um die Verarbeitung der Bilddaten weiter zu beschleunigen, wird zum Datentransfer DMA verwendet. Schließlich wurde auch noch der H.264-Encoder selbst optimiert. Zwar ist der Blackfin-Prozessor nicht superskalar, dennoch ist es mit einigen Einschränkung möglich, drei Instruktionen parallel auszuführen. Die Gesamtlaufzeit der drei parallelen Instruktionen entspricht jener der langsamsten der drei. Zudem verfügt der BF561 über spezielle Videoinstruktionen, die bei der H.264-Kodierung effizient eingesetzt werden können. Eine weitere nützliche Eigenschaft des Prozessors sind Hardware-Loops.

Diese ermöglichen die Ausführung von Schleifen ohne Mehraufwand durch das Inkrementieren der Zählervariable und der Überprüfung der Abbruchbedingung. Der Quellcode des H.264-Encoders wurde zudem durch die Verwendung von Assembler-Anweisungen optimiert. Für ein Testvideo konnten Gao et al. die Bildwiederholrate von 1,76 Bilder pro Sekunde (engl. Frames per Second, fps) auf 21,6 fps steigern.

2.2.4. Embedded Video Player

In ihrer Arbeit [16] beschreiben Zu-jue et al. das Design und die Implementierung eines Video Players für ein Embedded System. Die Basis des Embedded System bildet ein Intel PXA255 ARM Prozessor, auf dem ein Linux-Kernel läuft. Das Systemdesign ist in Abbildung 2.5 dargestellt.

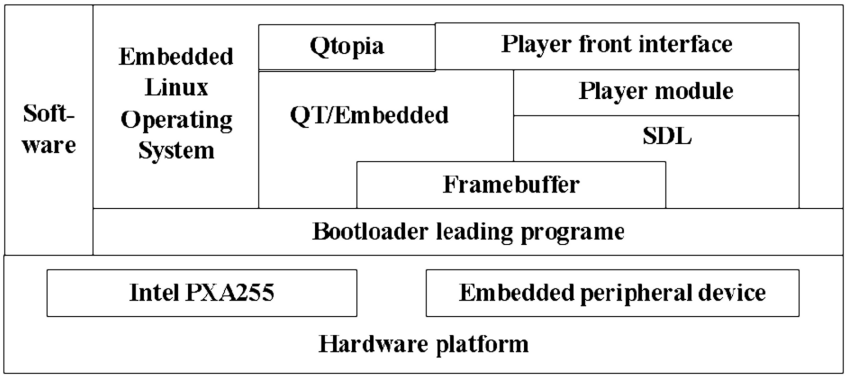


Abbildung 2.5.: Das Design des Video Players [16, S. 2]

Als großes Problem bestehender Video Player für Embedded Systems wird die oft sehr geringe Anzahl an unterstützen Videoformaten identifiziert. Dieser Einschränkung begegnen Zu-jue et al. durch den Einsatz der FFmpeg-Bibliothek [1], die Unterstützung für die Dekodierung zahlreicher Video- und Audioformate bietet.

Für die Implementierung der grafischen Benutzeroberfläche des Players wird, nach Evaluierung verschiedener Bibliotheken (Microwindows, MiniGui und Qt), Qt verwendet, da es die Programmierung des Players sehr vereinfacht und dessen Zuverlässigkeit und Stabilität erhöht. Qt läuft direkt auf dem Framebuffer und benötigt keinen X-Server.

Das Player-Modul ist für die Darstellung der Videos verantwortlich. Die Videos werden von der FFmpeg-Bibliothek dekodiert und mit der SDL-Bibliothek am LCD des Embedded System dargestellt. Das Player-Modul bietet Funktionen wie Play, Pause, und Stop, die von der Benutzeroberfläche aus aufgerufen werden. Um sich gegenseitig nicht zu beeinflussen, laufen Benutzeroberfläche und Player-Modul in eigenen Threads.

Laut Zu-ju et al. kann ihr Player die meisten gängigen Videoformate ausreichend schnell wiedergeben. Konkrete Angaben über maximale Auflösung und Bitrate der Videos werden leider nicht gemacht.

[1] http://ffmpeg.org

2.3. Kommerzielle Videorekorder und -transmitter

Dieser Abschnitt liefert einen Überblick über am Markt verfügbare Systeme zur Aufzeichnung und Übertragung von Video- und Audiosignalen. Die dabei identifizierten Systemeigenschaften sollen in das Design der MFA-Anwendung einfließen.

2.3.1. Pixavi Xcore

Die Firma Pixavi bietet mit dem Xcore (siehe Abbildung 2.6) ein Produkte an, das für den Einsatz als tragbares Videokommunikationssysteme konzipiert wurde. Die Einsatzgebiete für den Xcore sind laut Pixavi:

Abbildung 2.6.: Der Xcore ST6000 von Pixavi [17].

- Videokonferenzen
- Teamarbeit
- Inspektions- und Wartungsarbeiten
- Webcasting

- Taktische Kommunikation
- Dokumentation
- Technische Hilfestellung
- Überwachung

Das Gerät verfügt über folgende Schnittstellen:

- HD-Video Komponenten Ein- und Ausgang
- Ethernet
- USB-2.0
- Spannungsausgang (7 V) zur Versorgung einer externen Kamera
- Anschluß für eine WIFI-Antenne
- Audio-Ein/Ausgang

Zudem verfügt das Gerät über eine Tastatur, ein LCD mit einer Auflösung von horizontal 640 und vertikal 480 Pixeln und Touchscreen, sowie ein integriertes Mikrophon. Die Kommunikation erfolgt über ein WLAN nach dem 802.11g- oder 802.11n-Standard. An Videoformaten werden H.263 und H.264 unterstützt, Videos können lokal gespeichert (64 GB) werden, Live-Streams werden per RTP beziehungsweise RTSP übertragen. Die Laufzeit des Xcore soll in vollem Betrieb etwa 6 Stunden betragen.

Einschränkungen

Größtes Manko des Pixavi Xcore ist dessen fehlende UMTS-Anbindung. Videos können nur am Gerät gespeichert, oder über ein WLAN-Verbindung übertragen werden. Die Übertragung des Videos über größere Strecken soll laut Pixavi mit Hilfe des Xkit Laptops erfolgen. Dieser kann die vom Xcaster über WLAN gesendeten Videodaten empfangen und per 3G-Modem weiterleiten.

Eine weitere Einschränkung des Xcaster ist das fehlen eines GPS-Empfängers. Eine Positionsbestimmung ist daher nicht möglich. Auch eine Schnittstelle zur Anbindung externer Sensoren (etwa Temperatur, Luftdruck oder Herzfrequenz) ist nicht vorhanden. Der Preis des XCore ST6000 ist mit 6250 US-Dollar (Anfrage per E-Mail an Pixavi am 13.10.2010) sehr hoch.

Abbildung 2.7.: Der m.AVR H.264 von TS-Market [18].

2.3.2. m.AVR H.264

Der m.AVR H.264 (siehe Abbildung 2.7) ist ein besonders kompakter Video- und Audiorekorder. Das Metallgehäuse ist nur 70 mal 82 mal 15 Millimeter groß und wiegt lediglich 110 Gramm. Einsatzzweck des m.AVR H.264 ist laut Hersteller die professionelle Aufzeichnung von Video- und Audiodaten.

Das Gerät verfügt über folgende Schnittstellen:

- 4 SD-Karteneinschübe
- Videoeingang für eine externe Kamera
- 5 V oder 12 V Spannungsausgang zur Versorgung einer externen Kamera
- Anschluß für einen externen GPS-Empfänger

Die Auflösung der internen Videokamera beträgt horizontal 640 und vertikal 480 Pixel. Über einen Spannungsausgang kann eine externe Kamera mit einer 5 V oder 12 V Spannung versorgt werden. Videos speichert der m.AVR mit einer Auflösung von bis zu 720 mal 576 Pixeln ab. Die Videodaten werden H.264-kodiert auf einer der SD-Karten gespeichert. Es stehen vier SD-Karteneinschübe zur Verfügung, durch die der Speicherplatz auf bis zu 128 GB erweitert werden kann. Das Gerät verfügt über einen Anschluß für einen externen GPS-Empfänger. Die Positionsdaten legt der m.AVR als Texteinblendung über das Kamerabild, oder speichert sie in einer Datei auf der SD-Karte. In vollem Betrieb kann der m.AVR, laut Hersteller, für bis zu 8 Stunden vom integrierten Lithium-Ionen-Akku mit Strom versorgt werden.

Einschränkungen

Der m.AVR verfügt über keinerlei Netzwerkschnittstellen. Videos können nur lokal auf den SD-Karten gespeichert werden, ein Live-Streaming ist somit nicht möglich. Abgesehen vom GPS-Sensor, ist die Anbindung weiterer externen Sensoren nicht möglich. Der Preis des m.AVR beträgt 550 US-Dollar [2].

2.3.3. LiveU LU-30

Der LU-30 (siehe Abbildung 2.8) der Firma LiveU dient zur Liveübertragung von Video- und Audiosignalen. Einsatzgebiete des HD60 sind laut LiveU:

- Live-Berichte von Kultur- oder Sportveranstaltung
- Übertragung von Nachrichten
- Mobile Berichterstattung von Motorrädern, Booten oder Hubschrauben
- Überwachung/Sicherheit

Abbildung 2.8.: Der LU-30 von LiveU [19].

[2] `http://merkurydnepr.com/product/professional-miniature-audio-video-recorder-mavr-h264x4.html`, (20.10.2010)

Das Gerät verfügt über folgende Schnittstellen:

- Digitale Videoeingänge (USB, Firewire, SDI)
- Analoger Composite Videoeingang
- Mikrophoneingang
- Ethernet

Der LU-30 kann Videosignale einer externen Kamera bis zu einer Auflösung von 720 mal 576 Pixeln in das H.264-Format kodieren. Die Übertragung der Daten erfolgt entweder über 3G Mobilfunk, WiMAX, WLAN, LAN oder per Satelliten-Uplink. Um eine stabile Datenverbindung über die Mobilfunkverbindung zu gewährleisten, können bis zu 7 3G-Modems in das Gerät eingebaut werden. Der Datenverkehr wird, je nach Verfügbarkeit der Verbindungen, auf die einzelnen Modems aufgeteilt. Die Konfiguration des Gerätes erfolgt entweder über eine Netzwerkverbindung oder lokal über den LCD-Touchscreen. Das LCD kann auch eine Vorschau des Videobildes anzeigen.

Einschränkungen

Das Gerät verfügt über keine Möglichkeit, Sensordaten aufzunehmen. Die Abmessungen sind mit 25 cm mal 36 cm mal 11.5 cm sehr groß, ebenso wie das Gewicht von 5 kg. Daher muß der LU-30 in einer großen Tasche oder einem Rucksack transportiert werden. Es ist nicht möglich den LU-30 käuflich zu erwerben, das Gerät wird von Distributoren vermietet. Die Firma Livestream bietet den LU-30 für eine monatliche Miete von 2500 US-Dollar an, inklusive Datenvolumen zur Übertragung von 30 Stunden Video pro Monat.

2.3.4. mini 3G DVS

Der mini 3G DVS der Marke Plustek ist ein am Körper tragbares Gerät zur lokalen Aufzeichnung oder Liveübertragung von Video- und Audiodaten. Es basiert auf einem nicht näher spezifizierten TI DaVinci SoC.

Das Gerät verfügt über folgende Schnittstellen:

- Komponenten Videoeingang
- Mikrophoneingang
- Audioausgang
- USB-Schnittstelle
- μSIM-Karteneinschub
- MicroSD-Karteneinschub

Der mini 3G DVS kann Videos einer externen Kamera mit einer Auflösung von bis zu 640 mal 480 Pixeln und einer Bildwiederholrate von 25 Bildern pro Sekunde in das H.264-Format kodieren. Audiosignale kodiert er in das AAC-Format. Die Multimediadaten werden live per 3G Mobilfunk übertragen, oder lokal auf einer bis zu 32 GB großen MicroSD-Karte gespeichert. Über den Mikrophoneingang und den Audioausgang ist eine bidirektionale Audiokommunikation zwischen dem mini 3G DVS und dem Empfänger möglich. Der 2,4 Zoll große TFT-Bildschirm zeigt eine Vorschau des Kamerabildes oder dient zur Darstellung eines zuvor aufgezeichneten Videos. Die Positionsdaten des integrierten GPS-Empfängers können in das Videobild eingeblendet oder an einen Server übertragen werden. Die Abmessungen des Gehäuses betragen 90 mm mal 60 mm mal 25 mm. Der integrierte Akku versorgt den mini 3G DVS in vollem Betrieb für bis zu 3 Stunden mit Strom. Der Preis bei Direktimport aus China soll 560 US-Dollar pro Stück betragen.

Einschränkungen

- Der Bildschirm ist sehr klein.
- Das Gerät verfügt über keinen Touchscreen.
- Die Akkulaufzeit ist mit maximal 3 Stunden sehr kurz.
- Es können keine externen Sensoren an das Gerät angebunden werden.

2.3.5. Zusammenfassung der Produkteigenschaften

Die Eigenschaften der betrachteten Produkte sind in Tabelle 2.8 zusammengefasst.

Teil des Vergleichs sind jene Eigenschaften, die auch die in diesem Projekt entwickelte Hardware aufweist. Man erkennt, daß keines der Produkte über alle Eigenschaften verfügt. Entweder fehlt eine Mobilfunkanbindung, oder eine Spannungsversorgung für eine externe Kamera. Auch über einen GPS-Empfänger oder einen SD-Karteneinschub verfügen nicht alle Geräte. Der Pixavi Xcore ist am ehesten mit dem MFA zu vergleichen. Allerdings fehlt ihm das wichtige 3G-Modem, ein SD-Karteneinschub, ein GPS-Empfänger und die Möglichkeit auf externe Sensoren zuzugreifen. Zudem ist der Preis des Xcore sehr hoch.

Trotz intensiver Suche konnten keine weiteren Produkte gefunden werden, deren Hardware an die Funktionalität und Kompaktheit des MFA heranreichen würden. Es zeigt sich, daß der MFA, als Hardwarebasis für dieses Projekt, über eine einzigartige Kombination an Funktionen verfügt und somit eine kleine Nische im großen Markt der mobilen Videokommunikation besetzt.

	MFA	Pixavi Xcore	m.AVR H.264x4	LiveU LU30	mini 3G DVS
3G	+	-	-	+	+
WLAN	+	+	-	+	-
Ethernet	+	+	-	+	-
USB	+	+	-	+	+
Audioeingang	+	+	+	+	+
Videoeingang	+	+	+	+	+
Spannungsversorgung für externe Kamera	+	+	+	-	-
Bildschirm	groß	groß	klein	groß	klein
Touchscreen	+	+	-	+	-
SD-Karteneinschub	+	-	+	-	+
GPS	+	-	+	-	+
externe Sensoren	+	-	-	-	-
Größe [mm]	150x100x30	140x102x25	70x82x15	280x370x750	96x63x22
Gewicht [g]	300	300	110	5000	300
Preis [US-Dollar]	+	6250	550	2500 pro Monat	560
+/grüner Hintergrund: Eigenschaft vorhanden/positiv -/oranger Hintergrund: Eigenschaft nicht vorhanden/negativ					

Tabelle 2.8.: Die Eigenschaften der Produkte im Überblick

3. Design der Software-Architektur

3.1. Zielsetzung

Ziel diese Projekts ist es, eine Softwareanwendung zu entwickeln, die es ermöglicht den MFA als mobiles Speicher- und Übertragungssystem für Audio- und Videosignale zu nutzen. Die Signale werden, vom Anwender gesteuert, über Kamera und Mikrophon des MFA aufgenommen, komprimiert und am MFA gespeichert oder live über eine Funkverbindung zu einem Empfänger übertragen. Die Signale sollen den Empfänger mit möglichst kleiner Latenz erreichen. Zudem soll der Empfänger in der Lage sein, Nachrichten an den MFA zu senden. Mögliche praktische Einsatzgebiete für ein solches Gerät sind:

- Unterstützung bei der Fernwartung in Industrieanlagen oder Reinräumen.
- Liveübertragung von Sportereignissen wie Radrennen oder Langlauf, bei denen der Sportler den MFA aufgrund des geringen Gewichts bei sich tragen kann.
- Im Bereich des Videojournalismus, wo mit Hilfe des MFA eine technisch einfach zu realisierende Liveberichterstattung von interessanten Ereignissen möglich wird.
- Koordinierung von Teameinsätzen aller Art.

Aus den genannten Einsatzgebieten ergeben sich folgende, grundlegende Anforderungen an die Funktionalität des MFA:

- Aufzeichnung von Video- und Tondaten
- Wiedergabe von Video- und Tondaten
- Tonübertragung zwischen MFA und Empfänger
- Videoübertragung zwischen MFA und Empfänger
- Übertragung der GPS-Positionsdaten
- Intuitive Bedienung des MFA

In Abbildung 3.1 sind die beteiligten Komponenten auf höchster Abstraktionsebene dargestellt. Der MFA kann über die Kamera und das Mikrophon Video- und Tonsignale aufzeichnen. Über den integrierten GPS-Empfänger sammelt er Positionsdaten. Alle aufgezeichneten Daten werden zum MFA-Server übertragen.

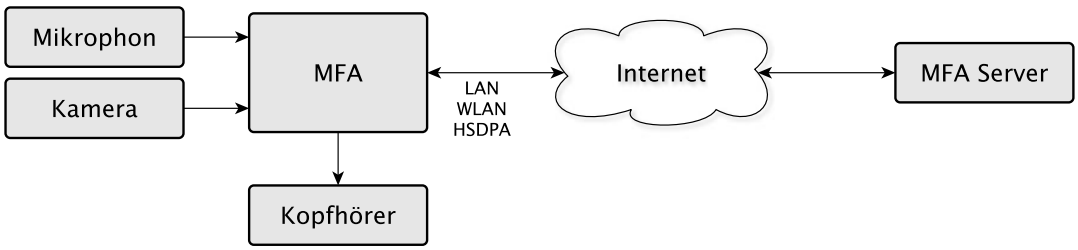


Abbildung 3.1.: Kommunikation zwischen dem MFA und dem Empfänger.

3.2. Analyse der Hardware

3.2.1. Die Hardwarekomponenten des MFA

Sämtliche Funktionen, die von der Anwendung am MFA ausgeführt werden können, beruhen auf den Fähigkeiten der vorhandenen Hardware. Sie bildet die Grundlage für den Entwurf der Software. Ein Blockdiagramm der wichtigsten Hardwarekomponenten ist in Abbildung 3.2 dargestellt. Eine kurze Analyse der Komponenten soll verdeutlichen, in wie weit die gewünschte Funktionalität der Anwendung realisiert werden kann und mit welchen Einschränkungen durch die Hardware gerechnet werden muß.

i.MX27 Application Processor

Der Application Processor bildet den Kern der MFA-Hardware. Er verfügt über eine mit 400 MHz getaktete ARM9 CPU, 128 MB SDRAM, sowie zahlreiche Controller und Schnittstellen, über die externe Komponenten angebunden werden. Im Anhang in Abbildung A.1 ist ein Blockdiagramm des i.MX27 dargestellt.

Der Audio Controller des i.MX27 ermöglicht es, Tonsignale aufzuzeichnen oder wiederzugeben. Aufnahmen können mit Abtastraten von 8 kHz oder 16 kHz durchgeführt werden. Die Wiedergabe von mit bis zu 48 kHz abgetasteten Tonsignale ist möglich. Leider besitzt der MFA keinen Hardware-Chip zur Weiterverarbeitung der Tonsignale. Die Verarbeitung, zum Beispiel eine Kodierung in das MP3-Format, muss daher in Software erfolgen und führt zu einer nicht vernachlässigbaren Belastung der CPU.

Die Video Processing Unit (VPU) ist für die Kodierung und Dekodierung der Videosignale zuständig. Sie unterstützt die Formate H.264, H.263 und MPEG-4 bei einer Bildauflösung von bis zu 720 mal 576 Pixeln und einer Bildwiederholrate von bis zu 25 Bildern pro Sekunde.

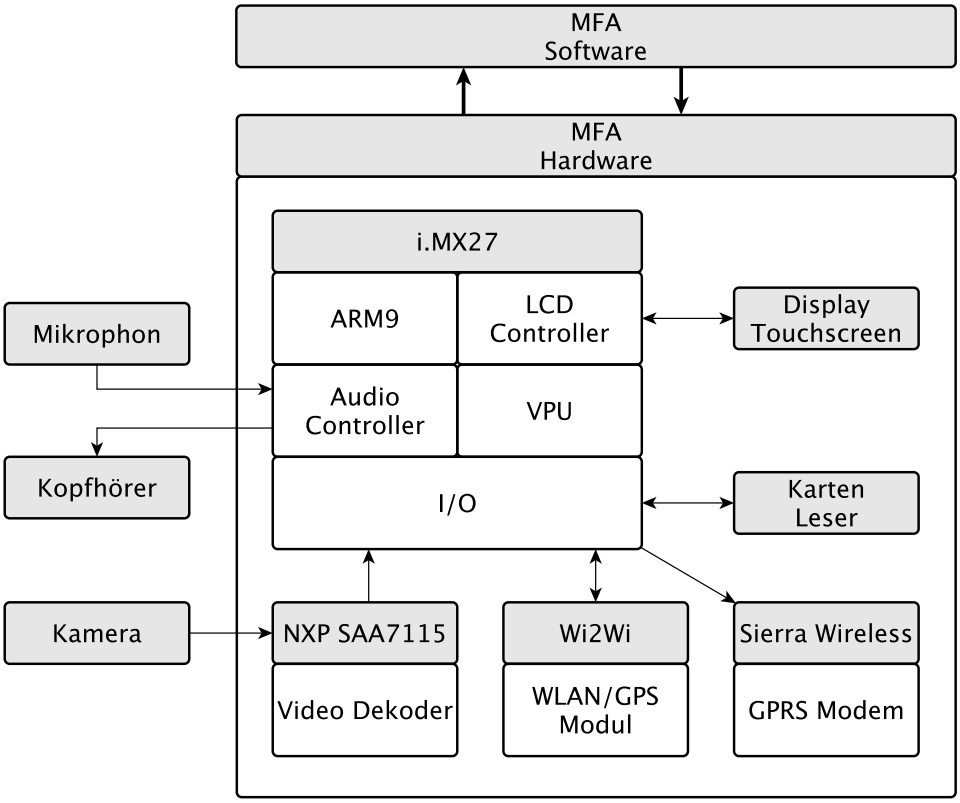


Abbildung 3.2.: Die Hardwareblöcke des MFA.

Display

Das Active Matrix Organic Light Emitting Diode (AMOLED) Display des MFA hat eine Auflösung von 480 mal 272 Pixeln und ist mit einem Touchscreen ausgestattet. Die Stromaufnahme des Display ist stark von der Farbe des dargestellten Inhalts abhängig. Für weiße Flächen ist die Stromaufnahme maximal, während sie für dunkler werdende Inhalte immer weiter abnimmt. Bei der Erstellung der grafischen Oberfläche der Anwendung ist daher darauf zu achten, die Darstellung großer, heller Flächen möglichst zu vermeiden, um die Stromaufnahme des Displays zu minimieren. Der LCD-Controller, an den das Display angeschlossen ist, würde eine maximale Auflösung von 800 mal 600 Pixeln unterstützen.

Kartenleser

Der Kartenleser des MFA nimmt MicroSD-Karten auf. Karten mit einer Kapazität von bis zu 16 GB wurden erfolgreich getestet.

UMTS-Modem

Das UMTS-Modem von Sierra Wireless ist im PCI Express MiniCard Format ausgeführt und über USB 2.0 an den Application Processor angebunden. Es unterstützt folgende Standards zur Datenübertragung: HSUPA, HSDPA, EDGE, GPRS und GSM.

WLAN/GPS Modul

Das im PCI Express MiniCard Format ausgeführte Wi2Wi-Modul enthält sowohl ein Funkmodul als auch einen GPS-Empfänger. Das Funkmodul ist über Secure Digital Input Output (SDIO) an den Application Processor angebunden und unterstützt Datenübertragung nach den IEEE-Standards 802.11 b und g, sowie die Datenverschlüsselungsmethoden WEP und WPA.

Der GPS-Empfänger ist über eine UART an den Application Processor angebunden. Die Kommunikation mit dem GPS-Empfänger erfolgt nach dem National Marine Electronics Association (NMEA) 0183 Protokoll.

Videodekoder

Der NXP SAA7115 ermöglicht es, die PAL-, NTSC- oder SECAM-Signale einer externen Kamera zu digitalisieren. Die Erkennung das Signaltyps erfolgt automatisch. Die digitalisierten Daten können von der VPU weiterverarbeitet werden.

3.2.2. Einschränkungen durch die Hardware

Die Eigenschaften der Hardwareblöcke führen zu folgenden Vorbedingungen und Einschränkungen, auf die beim Entwurf der Software Rücksicht genommen werden muß.

- Die CPU ist 400 MHz schnell.
- Es stehen 128 MB SDRAM zur Verfügung.
- Die Auflösung von Videos ist bei einer Bildwiederholrate von 30 Bildern pro Sekunde, sowohl bei der Aufnahme, als auch bei der Wiedergabe, auf 720x576 Pixel beschränkt.
- Das Display ist 480 mal 272 Pixel groß.
- Die Stromaufnahme des Displays ist von der Helligkeit des dargestellten Inhalts abhängig. Die Darstellung heller Inhalte benötigt mehr Strom als die dunkler.
- Es stehen 16 GB nicht flüchtiger Speicher zur Verfügung.
- Die Datenübertragung kann per UMTS-Modem oder per WLAN-Modul erfolgen.
- Die Datenrate bei Nutzung des UMTS-Modems ist auf die Angaben im HSUPA-Protokoll (Uplink) und HSDPA-Protokoll (Downlink), sowie auf die vom Netzbetreiber tatsächlich ermöglichten und praktisch erreichbaren Werte beschränkt.

- Die Datenrate bei Nutzung des WLAN-Moduls ist auf die Angaben im IEEE 802.11g Standard und die praktisch tatsächlich erreichbaren Werte beschränkt.
- Tonsignale können mit Abtastraten von 8 kHz oder 16 kHz aufgezeichnet werden.
- Die Wiedergabe von mit bis zu 48 kHz abgetasteten Tonsignale ist möglich.
- Sowohl die Kodierung als auch die Dekodierung der Tonsignale muß von der ARM-CPU durchgeführt werden.

3.2.3. Hardware-Software Schnittstelle

Die Schnittstelle zwischen der MFA-Hardware und Software bildet ein Linux-Kernel. Für alle Hardwarekomponenten existieren Linux-Kernelmodule durch die eine Nutzung der Hardware durch die Software möglich wird. Für die in Abbildung 3.2 dargestellten Hardwaremodule folgen nun Überlegungen, wie die MFA-Anwendung die Module ansprechen kann. Dabei sollte ein direkter Aufruf von Kernelfunktionen vermieden werden. Die Kommunikation mit der Hardware soll durch den Einsatz bereits bestehender Systembibliotheken möglichst weit abstrahiert werden, wodurch eine schnellere, weniger fehleranfällige Entwicklung der Anwendung ermöglicht wird.

VPU

Um die Videodekodierer und -kodierer des i.MX27 nutzen zu können, stellt der Hersteller Freescale eine eigene Bibliothek bereit. Diese ermöglicht eine systemnahe, relativ komplexe Programmierung der Hardware. Aufbauend auf dieser Bibliothek, hat Freescale Plugins für das Multimedia Framework GStreamer [1] entwickelt. Über die Plugins können Aufnahme, Wiedergabe und Weiterverarbeitung der Videodaten wesentlich einfacher und schneller realisiert werden. Zudem existieren GStreamer-Plugins zur Nutzung der Audiohardware des i.MX27.

Display

Das Display kann über das Kernelmodul „mx2fb" angesprochen werden. Eine Besonderheit des Bildspeichers (engl. Framebuffer) des i.MX27 ist, daß er in zwei, sich überlagernde Bereich (fb0 und fb1) geteilt ist, die unabhängig von einander aktiviert und beschrieben werden können. Auf fb1 wird der Vordergrund dargestellt, der sämtliche Inhalte von fb0 verdeckt. Um dennoch Inhalte auf fb0 anzeigen zu könne, kann für fb1 eine Farbe als transparent definiert werden (Chroma Key). Außerdem kann für fb1 ein globaler Translu-zenzwert definiert werden.

[1] `http://www.gstreamer.net` (4.5.2011)

Um eine grafische Benutzeroberfläche anzuzeigen, eignet sich die direkte Programmierung des Framebuffers nur schlecht, da keinerlei Funktionen zur Darstellung von grafischen Elementen wie Linien, Bilder oder Fenster vorhanden sind. Stattdessen ist es sinnvoll, eine Bibliothek zu verwenden die solche Funktionen zur Verfügung stellt. Der Framebuffer-Treiber des i.MX27 implementiert das Kernelinterface für Standard-Framebuffer und ist daher zu den unter Linux gängigen Grafikbibliotheken kompatibel. Rein prinzipiell wäre es daher problemlos möglich, einen X-Server auf dem MFA zu betreiben. Um die recht begrenzten Ressourcen des MFA besser nutzen zu können, ist es allerdings sinnvoll den Einsatz eines X-Servers zu vermeiden. Für Linux existieren eine Reihe von Grafikbibliotheken, die direkt auf dem Framebuffer aufbauen und keinen X-Server benötigen:

- SDL: Der Simple Direct Media Layer [2] ist eine Bibliothek die vorrangig für die Programmierung von Spielen konzipiert wurde. Sie bietet Funktionen, die für die schnelle Darstellung von Grafiken optimiert sind. Zudem existieren eine Vielzahl an Erweiterungen, etwa zum Laden von Bildern in verschiedensten Formaten, oder zur Verwendung von TrueType Schriftarten. Die Programmierung erfolgt in C. Vorteile von SDL sind die einfache, gut dokumentiert API und der geringe Ressourcenbedarf. Der größte Nachteil ist, daß die Programmierung auf einer sehr tiefen Ebene stattfindet. Grafische Elemente zur Darstellung einer Benutzeroberfläche sind nicht vorhanden und müssen erst selbst programmiert werden.
- GTK+: Der Gnome Tool Kit [3] ist die für die Programmierung der Gnome Desktopumgebung verwendete Grafikbibliothek. Sie bietet sämtliche Funktionen, die notwendig sind um grafische Benutzeroberflächen zu erstellen. Die Bibliothek ist in C programmiert kann aber in vielen anderen Programmiersprachen, wie etwa C++ oder Python, verwendet werden. Vorteile von GTK+ sind der große Funktionsumfang und die Vielzahl an bereits vorhandenen grafischen Elementen. Nachteil ist die komplexe, anfangs schwer überschaubare API.
- Qt: Qt [4] ist eine in C++ geschriebene Klassenbibliothek von Nokia zur Erstellung grafischer Benutzeroberflächen. Sie wird vom Linux-Desktop KDE verwendet, kommt aber auch auf Embedded Systems, wie zum Beispiel Symbian-Mobiltelefonen von Nokia, zum Einsatz. Vorteile von Qt sind der großer Funktionsumfang, die übersichtliche, einfach zu erlernende API und die sehr gute Dokumentation. Einziger Nachteil ist der im Vergleich zu den anderen Bibliotheken etwas größere Ressourcenbedarf.

Nach einer Abwägung der Vor- und Nachteile der drei Bibliotheken scheint sich Qt am Besten für die Erstellung der grafischen Oberfläche des MFA zu eignen.

Kartenleser

Sobald eine SD-Karte in den Kartenleser eingelegt wird, kann über das Dateisystem auf deren Inhalt zugegriffen werden. Von Seiten der Software ist keine spezielle Konfiguration notwendig.

[2] `http://www.libsdl.org` (4.5.2011)
[3] `http://www.gtk.org` (4.5.2011)
[4] `http://qt.nokia.com` (4.5.2011)

UMTS-Modem

Das Modem wird durch Laden des entsprechenden Kernelmoduls aktiviert und kann über USB als serielles Device angesprochen werden. Es unterstützt den AT-Befehlssatz. Der PPP Daemon [5] vereinfacht die Konfiguration des Modems und ermöglicht die Einwahl in das Netz des Mobilfunkbetreibers.

WLAN/GPS-Modul

Das WLAN-Modul ist nach Laden des Kernelmoduls verfügbar und kann mit den unter Linux üblichen Mitteln konfiguriert und in Betrieb genommen werden. Auch ein Einsatz in WEP oder WPA verschlüsselten Netzen ist problemlos möglich.

Das GPS-Modul benötigt kein eigenes Kernelmodul, es ist sofort über die serielle Schnittstelle ansprechbar. Die Kommunikation mit dem Modul erfolgt nach dem NMEA-Protokoll. Die Programmierung eines Parsers für das relativ komplexe Protokoll kann durch Verwendung des GPS Daemon gpsd [6], oder der Qt-Bibliothek Qt Mobility [7] vermieden werden.

Videodekoder

Der Videodekoder SAA7115 ist sofort nach Laden des Kernelmoduls einsatzbereit. Der Zugriff auf die Kamerabilder erfolgt entweder direkt durch die Programmierung der entsprechenden Kernelfunktionen, oder mit Hilfe von GStreamer. Für dieses existiert ein Plugin zum Zugriff auf die Kameradaten.

Zusammenfassend ergibt sich das in Abbildung 3.3 dargestellte Bild der Software-Architektur des MFA, die sich aus den verschiedenen Softwarekomponenten zusammensetzt. Die Anwendung selbst kommuniziert nicht direkt mit der Hardware oder dem Linux Kernel, sondern nutzt dazu entsprechende Services oder Bibliotheken. Dies verkürzt die Entwicklungszeit der Anwendung erheblich. Zudem ist so eine von der Hardware unabhängige Entwicklung der Anwendung möglich, da sämtliche Bibliotheken und Services auch auf normalen Desktop-PCs verfügbar sind.

3.3. Anforderungen des Benutzers an den MFA

Die Software des MFA soll aus Sicht des Anwenders folgende Anforderungen erfüllen:

A.1 Die Aufzeichnung der Video- und Tonsignale über die Kamera und das Mikrophon des MFA soll gestartet und gestoppt werden können.

[5] `http://git.ozlabs.org/?p=ppp.git;a=summary` (4.5.2011)
[6] `http://gpsd.berlios.de/` (4.5.2011)
[7] `http://qt.nokia.com/products/qt-addons/mobility` (4.5.2011)

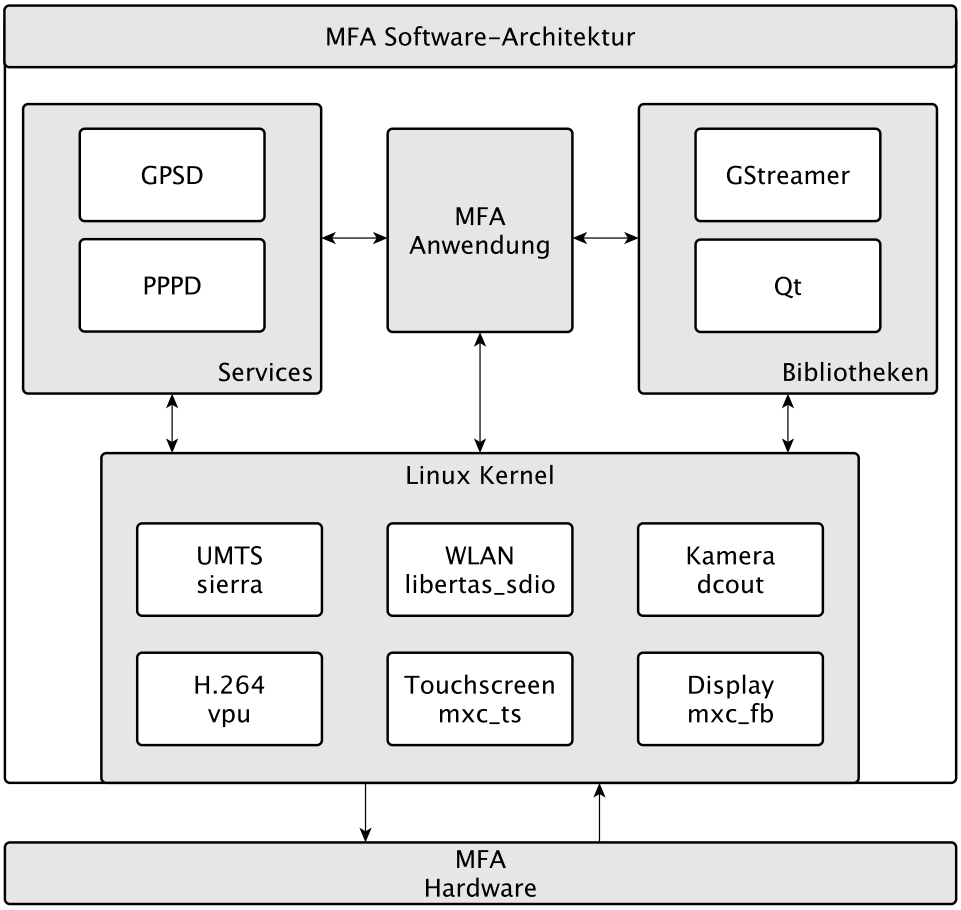


Abbildung 3.3.: Die Software-Architektur des MFA.

A.2 Die Laustärke der aufgezeichneten Tonsignale soll geregelt werden können.

A.3 Die Parameter des Audiokodierers sollen eingestellt werden können.

A.4 Die Auflösung und die Bildwiederholrate der Kamera soll geregelt werden können.

A.5 Die Parameter des Videokodierers sollen eingestellt werden können.

A.6 Die Anzeige einer Vorschau des Videobildes am Bildschirm des MFA soll aktiviert und deaktiviert werden können.

A.7 Die Helligkeit des Bildschirms soll geregelt werden können.

A.8 Die Verbindung zum MFA-Server soll aufgebaut werden können.

A.9 Die Adresse des Servers soll konfiguriert werden können.

A.10 Das Ziel der aufgezeichneten Daten soll gewählt werden können:

- Datei auf der SD-Karte

- Liveübertragung zum MFA-Server
- Beides

A.11 Die Wiedergabe von auf der SD-Karte gespeicherten Video- und Tondaten soll gestartet und gestoppt werden können.

A.12 Die Lautstärke der Tonausgabe soll geregelt werden können.

A.13 Die Aufzeichnung von Positionsdaten in eine Datei auf der SD-Karte soll gestartet und gestoppt werden können.

A.14 Die Übertragung von Positionsdaten zum MFA-Server soll gestartet und gestoppt werden können.

3.4. Entwurf der Anwendung

Aus der Analyse der zuvor beschriebenen Funktionen, Einschränkung und Anforderungen ergibt sich eine Unterteilung der Anwendung in folgende Funktionsbereiche:

- Die Steuerung der Multimediakomponenten.
- Die Darstellung der grafischen Benutzeroberfläche.
- Die Auswertung des GPS-Signals.
- Die Verwaltung der Netzwerverbindungen.
- Die Kommunikation mit dem MFA-Server.

Eine objektorientierte Analyse dieser Funktionsgruppen führt zu dem in Abbildung 3.4 dargestellten Klassendiagramm. Jede der Klassen kapselt einen der zuvor beschriebenen Funktionsbereiche. Die Funktionsweise und Interaktion der Klassen werden im Folgenden beschrieben.

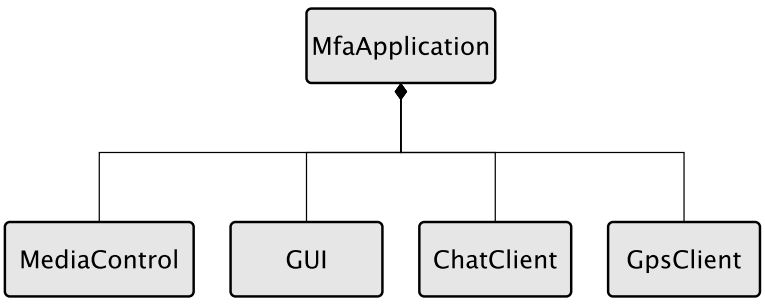


Abbildung 3.4.: Klassendiagramm der Anwendung

3.4.1. Kommunikation zwischen den Objekten der Anwendung

Signals/Slots

Da die Entwicklung der Anwendung mit der Qt-Bibliothek durchgeführt wird, ist es sinnvoll den in Qt integrierten Signals/Slots-Mechanismus zur Kommunikation zwischen den einzelnen Objekten der Klasse zu nutzen. Die Funktionsweise dieses Mechanismus entspricht der des Observer-Designpatterns [20].

Der Signals/Slots-Mechanismus ermöglicht die Kommunikation zwischen von einander unabhängigen Objekten. Jedes Objekt, dessen Klasse von der Qt-Basisklasse QObject abgeleitet ist, kann beliebig viele Signale definieren. Ein Signal wird von einem Objekt dann emittiert, wenn sich der Zustand des Objekts verändert hat und es andere Objekte darüber informieren will. Empfänger eines Signals ist ein Slot. Ein Slot ist eine normale Methode einer Klasse die aufgerufen wird, sobald das mit dem Slot verbundene Signal emittiert wird. Jedes Signal verfügt über bestimmte Parameter, die beim Emittieren des Signals gesetzt werden müssen. Diese Parameter werden an die mit dem Signal verbundenen Slots übergeben. Signale und Slots zweier Objekte werden durch einen Funktionsaufruf mit einander verbunden. Der große Vorteil dabei ist, daß sich die beiden Objekte nicht kennen müssen. Die Abhängigkeiten unter den Klassen werden so auf ein Minimum reduziert, wodurch die Wiederverwendbarkeit der einzelnen Programmteile stark erhöht wird.

QSettings-Klasse

Um Parameter, wie zum Beispiel Dateinamen oder Videoeinstellungen, zwischen den Objekten der Anwendung austauschen zu können, bietet sich die Verwendung der Qt-Klasse QSettings an. Diese Klasse ermöglicht es, Variablen in einer nicht flüchtigen, über mehrere Programmstarts hinweg erhalten bleibenden, Datenbank zu speichern. Der Zugriff auf die Datenbank erfolgt über beliebig definierbare QSettings-Schlüssel. Jedem Schlüssel kann ein les- und schreibbarer Wert zugewiesen werden.

Durch die persistente Speicherung der Variablenwerte in einer Datei, kann einem Datenverlust durch eine plötzliche, unvorhersehbare Terminierung des Programms, etwa aufgrund eines leeren Akkus, vorgebeugt werden.

3.4.2. Multimedia-Bibliothek

Die Funktionen zur Steuerung der Video- und Audiohardware sollen in einer Multimedia-Bibliothek gekapselt werden. Die Bibliothek soll möglichst flexibel einsetzbar und einfach erweiterbar gestaltet werden, um sie in künftigen Projekten wiederverwenden zu können. Basis der Bibliothek ist GStreamer und die von Freescale entwickelten Plugins zur Nutzung der Hardware des MFA. Eine Liste der Plugins ist in Tabelle 3.1 enthalten.

Die Bibliothek soll die aufwändige Erstellung und Konfiguration der Plugins vor dem Nutzer der Bibliothek verbergen, und ihm eine einheitliche Schnittstelle zur Verwendung folgender Funktionen bieten:

GStreamer-Plugin	Verwendung
`mfw_v4lsrc`	Ermöglicht es, einen unkomprimierten Datenstrom von der Kamera einzulesen.
`mfw_v4lsink`	Dient dazu, einen unkomprimierten Videodatenstrom am Display des MFA darzustellen.
`alsasrc`	Liest einen unkomprimierten Datenstrom vom Mikrophon ein.
`alsasink`	Gibt einen unkomprimierten Audiodatenstrom über den Audioausgang wieder.
`mfw_vpuencoder`	Kodiert einen Videodatenstrom in das H.264-Format.
`mfw_vpudecoder`	Dekodiert einen H.264-Datenstrom.
`ffenc_mp2`	Komprimiert einen Audiodatenstrom in das MPEG-1 Audio Layer 2 (MP2)-Format.
`mad`	Dekomprimiert einen MP2-Datenstrom.
`rtph264pay`	Verpackt ein H.264-Datenpaket in ein RTP-Paket.
`rtpmpapay`	Verpackt ein MP2-Datenpaket in ein RTP-Paket.
`udpsink`	Versendet RTP-Pakete über eine Netzwerkverbindung.
`filesrc`	Liest Daten aus einer Datei ein.

Tabelle 3.1.: GStreamer-Plugins für den MFA.

- Erstellung von komprimierten Video- und Tonaufnahmen
- Transport der komprimierten Aufnahmen über eine Netzwerkverbindung in Echtzeit
- Wiedergabe von bestehenden Aufnahmen

Da die Bibliothek möglichst modular aufgebaut sein soll, bietet es sich an die Plugins in Klassen zu kapseln und diese Klassen frei mit einander kombinierbar zu gestalten. Diese Überlegung führt zum Konzept der Datenquellen und Datensenken. Eine Klasse ist eine Datenquelle, wenn sie einen Datenstrom produziert und anderen Klassen zugänglich macht. Eine Klasse ist eine Datensenke, wenn sie den Datenstrom einer Datenquelle übernimmt und in irgend einer Form verarbeitet. Aus den Anforderungen an die Bibliothek und den vorhandenen Plugins, ergibt sich die in Tabelle 3.2 gezeigte Aufteilung der Bibliothek in Datenquellen (mit dem Postfix „Source" bezeichnet) und -senken (mit dem Postfix „Sink" bezeichnet):

Klasse	Beschreibung	Plugins
LiveSource	Stellt komprimierte Video- und Audiodatenströme in Echtzeit zur Verfügung.	mfw_v4lsrc mfw_vpuencoder alsasrc ffenc_mp2
FileSource	Liest Video- und Audiodaten aus einer Datei ein.	filesrc

Klasse	Beschreibung	Plugins
FileSink	Schreibt komprimierte Video- und Audiodatenströme in eine Datei.	filesink
DisplaySink	Dekomprimiert H.264- und MP2-Datenströme und gibt sie über das Display und den Audioausgang wieder.	mfw_vpudecoder mad mfw_v4lsink alsasink
NetworkSink	Verschickt H.264- und MP2-Datenströme per UDP über eine Netzwerkverbindung.	rtph264pay rtpmpapay udpsink

Tabelle 3.2.: Klassen der Multimedia-Bibliothek.

Jede Datenquelle soll mit jeder Datensenke zusammenarbeiten. So soll etwa eine LiveSource mit einer FileSink und einer NetworkSink kombiniert werden können, um Kamera- und Mikrophonaufnahmen gleichzeitig in eine Datei zu speichern und über eine Netzwerkverbindung zu übertragen. Dieses modulare Konzept führt zu der in Abbildung 3.5 dargestellten Klassenhierarchie. Die einzelnen Klassen erfüllen dabei folgende Aufgaben:

MediaSource

Die Klasse MediaSource bildet die Basis für alle Video- und Tondatenquellen der Bibliothek. Sie kommuniziert mit den GStreamer-Plugins, über die der Zugriff auf die Kamera und das Mikrophon des MFA erfolgt. Jede neu implementierte Datenquelle muß von MediaSource abgeleitet werden.

Eine MediaSource ist eine Datenquelle, die einen Datenstrom erzeugt und an MediaSink-Objekte weiterleitet. Für jedes Paket des Datenstrom werden die Signale `newVideoData()` beziehungsweise `newAudioData()` emittiert. Die Empfänger der Signale sind alle registrierten MediaSink-Objekte. Jede MediaSource kann entweder eine Videodatenstrom oder einen Tondatenstrom oder einen Videodatenstrom und einen Tondatenstrom liefern. Die Datenströme werden komprimiert, im H.264-Format oder im MP2-Format, weitergeben.

Um einen Datenstrom zu starten oder zu stoppen, müssen die Funktionen `start()` oder `stop()` aufgerufen werden. Mit den Funktionen `addMediaSink(MediaSink)` und `removeMediaSink(MediaSink)` können MediaSink-Objekte registriert oder wieder entfernt werden.

MediaSink

Die Klasse MediaSink bildet die Basis aller Klassen, die Datenströme einer MediaSource verarbeiten wollen, etwa um sie wiederzugeben, oder um sie über das Netzwerk weiterzuleiten.

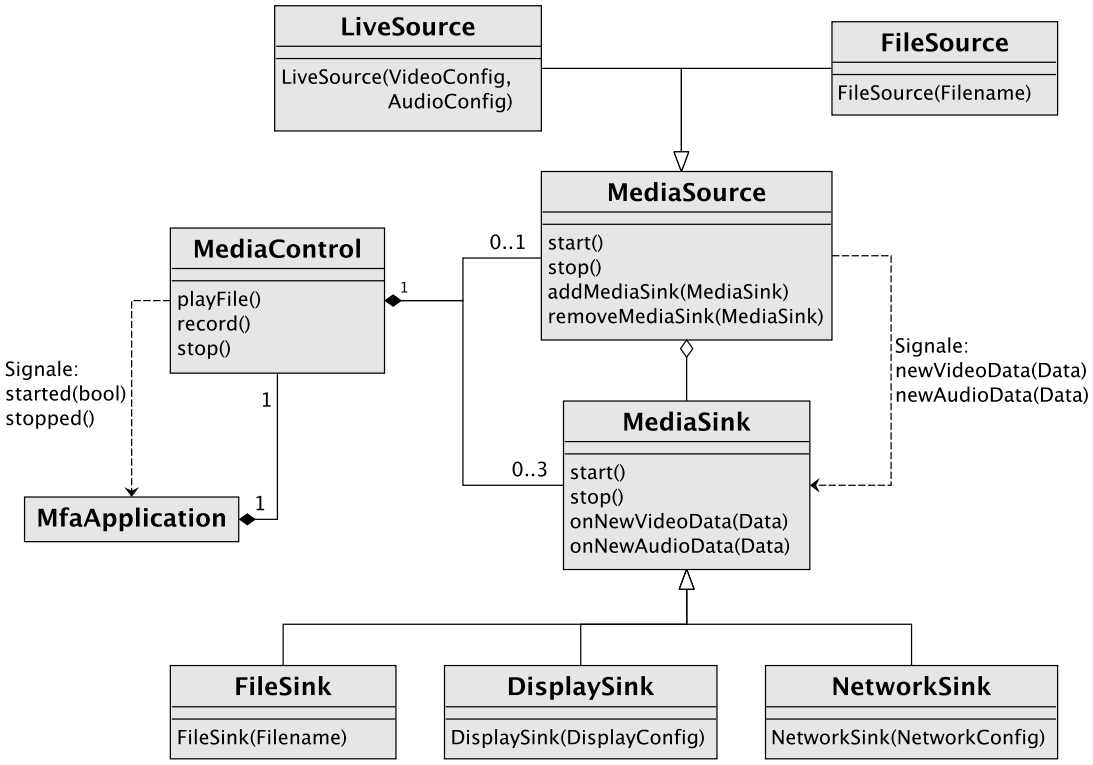


Abbildung 3.5.: Klassendiagramm der Medienbibliothek

Der Datenaustausch mit der MediaSource erfolgt über die beiden Slots `onNewVideoData(data)` und `onNewAudioData(data)`. Diese werden mit den Signalen `newVideoData` und `newAudioData` der MediaSource verbunden, sobald die MediaSink bei einer MediaSource registriert wird. Der Parameter `data` enthält die kodierten Video- beziehungsweise Audiodaten, die von der MediaSink weiterverarbeitet werden sollen.

Bei einer MediaSource können mehrere MediaSink-Objekte gleichzeitig registriert werden. Alle MediaSink-Objekte werden mit dem selben Datenstrom versorgt. Beispielsweise kann eine FileSink die Daten in eine Datei speichern, während sie eine NetworkSink gleichzeitig über das Netzwerk verschickt. Die MediaSink-Objekte können unabhängig von einander gestoppt werden, ohne sich gegenseitig zu beeinflussen.

LiveSource

Die Klasse LiveSource ermöglicht die Nutzung der Kamera und des Mikrophons. Wird sie gestartet, liefert sie, je nach Konfiguration, entweder einen Videodatenstrom, einen Audiodatenstrom, oder beides. Die Konfiguration der Kamera und des Mikrophons erfolgt über die beiden Parameter VideoConfig und AudioConfig. Diese werden im Konstruktor an die Klasse übergeben. VideoConfig enthält folgende Parameter:

- Horizontale Kameraauflösung
- Vertikale Kameraauflösung
- Bildwiederholrate
- Bitrate des Codec

AudioConfig enthält folgende Parameter:

- Abtastrate
- Bitrate des Codec

Nach Aufruf von `start()` beginnt die LiveSource Daten von der Kamera und dem Mikrophon zu kodieren und an die registrierten MediaSink-Objekte weiterzuleiten.

FileSource

Objekte der Klasse FileSource lesen Video- und Tondaten aus einer Datei. Der Name der Datei wird im Konstruktor an das Objekt übergeben. Die Datei muß ein Matroska Media Container [8] sein. Die im Container enthaltenen Datenströme werden entpackt und durch die Signale `newVideoData` und `newAudioData` an die registrierten MediaSink-Objekte übergeben. Folgende Medienformate werden unterstützt:

- MP2, MP3
- G.711 μ-law, G.711 a-law [21]
- H.264
- MPEG-4

FileSink

Eine FileSink verpackt die vom MediaSource-Objekt empfangenen Daten in einen Matroska-Container und schreibt diesen in eine Datei. Der Name der zu schreibenden Datei wird über den Konstruktor an das Objekt übergeben.

[8] `http://matroska.org` (4.5.2011)

DisplaySink

DisplaySink-Objekte dekodieren die von der MediaSource empfangenen Video- und Audiodaten. Die dekodierten Videodaten werden am Display dargestellt, die Tondaten über den Audioausgang wiedergegeben. Die Konfiguration der Klasse erfolgt Über den Parameter `DisplayConfig`, der an den Konstruktor übergeben wird. Der Parameter enthält folgende Werte:

- Horizontale Größe des Videofensters
- Vertikale Größe des Videofensters
- x/y-Koordinaten der linken, oberen Ecke des Videofensters

NetworkSink

Ein NetworkSink-Objekt verschickt die von der MediaSource empfangenen Video- und Tondaten über eine Netzwerkverbindung. Das Ziel der Daten wird über den an den Konstruktor der Klasse übergebenen Parameter `NetworkConfig` konfiguriert. Dieser enthält folgende Werte:

- IP-Adresse des Ziels
- Zielport des Tondatenstroms
- Zielport des Videodatenstroms

Die Kommunikation mit dem Zielserver erfolgt nach dem Real-time Transport Protocol (RTP) auf Basis von UDP. Sobald die NetworkSink gestartet wurde, erstellt sie eine dem Session Description Protocol (SDP) entsprechende Beschreibung des versendeten Datenstroms.

MediaControl

Die Klasse MediaControl bildet die Schnittstelle zwischen der Multimedia-Bibliothek und der MFA-Anwendung. Über den Slot `playFile()` kann die Wiedergabe einer Datei gestartet werden. Hierfür werden eine FileSource und eine DisplaySink erstellt und mit einander verbunden. Alle von der Klasse FileSource unterstützen Datenformate können wiedergegeben werden. Die zur Erstellung der beiden Klassen FileSource und DisplaySink notwendigen Parameter werden über ein QSettings-Objekt eingelesen. Die hierfür verwendeten Schlüssel mit dem Prefix `mfa/media/playback/*` sind in Tabelle 3.3 beschrieben. Das Objekt emittiert das Signal `started(true)`, falls die Wiedergabe erfolgreich gestartet werden konnte. Falls nicht, wird das Signal `started(false)` emittiert.

Um die Aufnahme von Bild und Ton zu starten, muß der Slot `record()` aufgerufen werden. Der Slot erstellt eine LiveSource, eine FileSink und, falls über den Schlüssel `mfa/media/record/stream` konfiguriert, eine NetworkSink, die miteinander verbunden und gestartet werden. Die zur Erstellung der Objekte notwendigen Parameter

werden aus den QSettings-Schlüsseln mit dem Prefix `mfa/media/record/*` übernommen (siehe Tabelle 3.3). Sobald die Aufzeichnung erfolgreich gestartet werden konnte, emittiert das Objekt das Signal `started(true)`. Tritt beim Versuch die Aufzeichnung zu starten ein Fehler auf, wird das Signal `started(false)` emittiert.

Um eine Wiedergabe oder eine Aufzeichnung zu stoppen, muß der Slot `stop()` aufgerufen werden. Sobald der Slot alle aktiven MediaSource- und MediaSink-Objekte stoppen konnte, wird das Signal `stopped()` emittiert.

3.4.3. Grafische Benutzeroberfläche

Das Ziel bei der Gestaltung der Benutzeroberfläche ist es, eine Bedienung der Anwendung entsprechend der in Abschnitt 3.3 definierten Anforderungen zu ermöglichen.

Die auf dem Display des MFA angezeigten Inhalte unterteilen sich in zwei Kategorien: Videobilder und die Benutzeroberfläche. Da das Display des MFA relativ klein ist, können Videobilder und Benutzeroberfläche nicht gleichzeitig dargestellt werden. Die Benutzeroberfläche soll daher als Overlay, das über dem Video ein- und ausgeblendet werden kann, realisiert werden. Der Wechsel zwischen Videoansicht und Benutzeroberfläche soll vom Benutzer möglichst schnell und einfach durchgeführt werden können, und optisch ansprechend erfolgen. Hierfür bietet sich die Verwendung der, schon in Abschnitt 3.2.3 beschriebenen, unabhängig von einander ansprechbaren, Framebuffer an. So kann, wie in Abbildung 3.6 ersichtlich ist, die Darstellung von Videos auf Framebuffer fb0 erfolgen, während die Benutzeroberfläche auf fb1 angezeigt wird. Das Ein- und Ausblenden der Benutzeroberfläche kann über die Regelung der Durchsichtigkeit des Framebuffer fb1 realisiert werden. Im ausgeblendeten Zustand ist fb1 vollkommen transparent, nur das Videobild ist sichtbar. Beim Einblenden der Benutzeroberfläche wird die Durchsichtigkeit von fb1 in kurzen Zeitabständen schrittweise erhöht, wodurch der Effekt einer sanften Einblendung entsteht. Das Ausblenden erfolgt auf gegenteilige Art und Weise.

Im normalen Betriebszustand zeigt das Display das Videobild, oder, falls kein Video abgespielt oder aufgezeichnet wird, ein Hintergrundbild an. Die Benutzeroberfläche ist nicht sichtbar. Diese wird eingeblendet, sobald der Anwender den Touchscreen bedient. Das Ausblenden der Benutzeroberfläche erfolgt, wenn der Benutzer das entsprechende grafische Element aktiviert, oder nach einer bestimmten Zeitspanne, in der vom Benutzer keine Eingaben am Touchscreen durchgeführt wurden.

Gestaltung der Benutzeroberfläche

Dieser Abschnitt beschreibt das Design der grafischen Benutzeroberfläche (engl. Graphical User Interface, GUI). Der Entwurf der Oberfläche wurde mit Hilfe des GUI-Editors der Qt-Entwicklungsumgebung QtCreator [9] erstellt. Für jeden der zuvor identifizierten Funktionsbereiche der Anwendung soll ein eigener Dialog erstellt werden. Daraus ergibt sich die in Abbildung 3.7 gezeigte, baumartigen Struktur der Benutzeroberfläche.

[9] `http://qt.nokia.com/products/developer-tools` (4.5.2011)

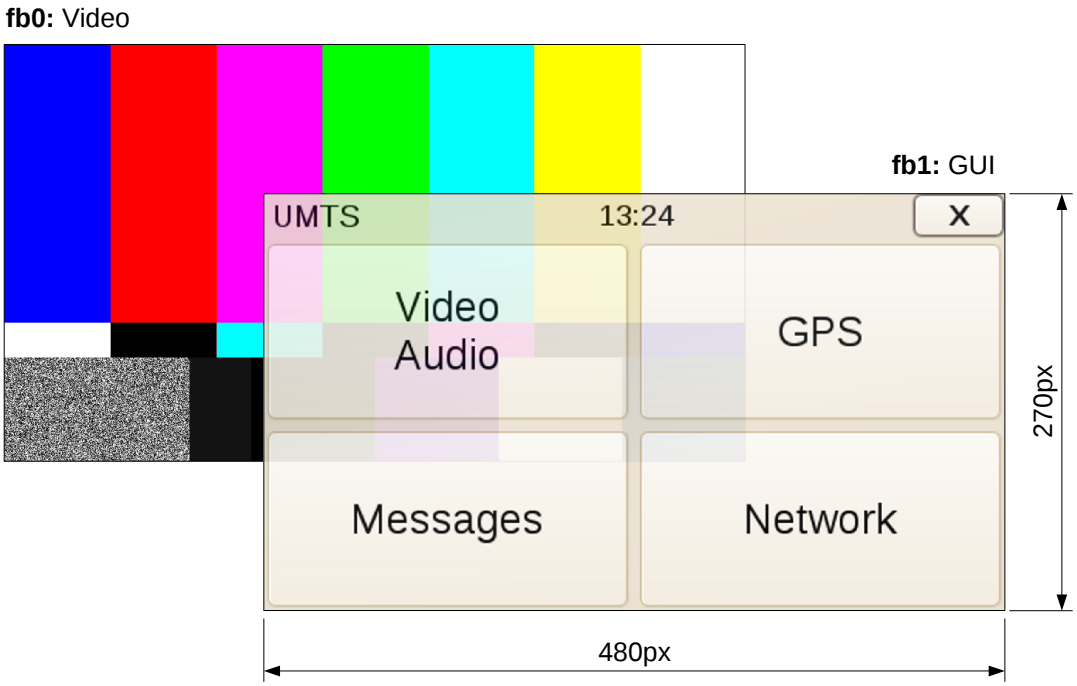


Abbildung 3.6.: Gleichzeitige Darstellung von Videos und Benutzeroberfläche

Start: Der Startbildschirm der Anwendung (siehe Abbildung 3.8) wird eingeblendet, sobald der Benutzer den Touchscreen bedient. In der Statuszeile am oberen Bildschirmrand werden die aktuelle Uhrzeit sowie ein Knopf zum schließen der Benutzeroberfläche, und ein Knopf, über den der Benutzer zurück zum Startbildschirm navigieren kann, angezeigt. Alle Dialoge der grafischen Benutzeroberfläche verfügen über diese Statuszeile. Darunter befinden sich vier Knöpfe, über die der Anwender die Unterdialoge erreichen kann.

Video/Audio: Der Video/Audio-Dialog besteht aus zwei Unterdialogen, die der Anwender über die Reiter am rechten Bildschirmrand erreichen kann. Nach Aufruf des Dialoges wird der in Abbildung 3.9a dargestellte Reiter angezeigt.

Betätigt der Benutzer den Knopf zum starten einer Aufnahme, wird das Signal `requestStartRecord()` emittiert. Der Knopf ändert sich daraufhin in einen „Stop" Knopf, bei dessen Betätigung das Signal `requestReocrdStop()` gesendet wird. Sobald eine Aufnahme gestartet wurde, wird die Benutzeroberfläche ausgeblendet.

Über den „Select File" Knopf gelangt der Benutzer zu dem in Abbildung 3.9c dargestellten Dialog. Dieser zeigt eine Liste aller auf dem Gerät gespeicherten Videodateien an. Sobald der Benutzer eine Datei auswählt, wird der Dialog geschlossen und das Signal `requestPlaybackStart()` emittiert. Der Name der gewählten Datei wird im QSettings-Schlüssel `mfa/media/playback/filename` gespeichert. Sobald die Wiedergabe der gewähl-

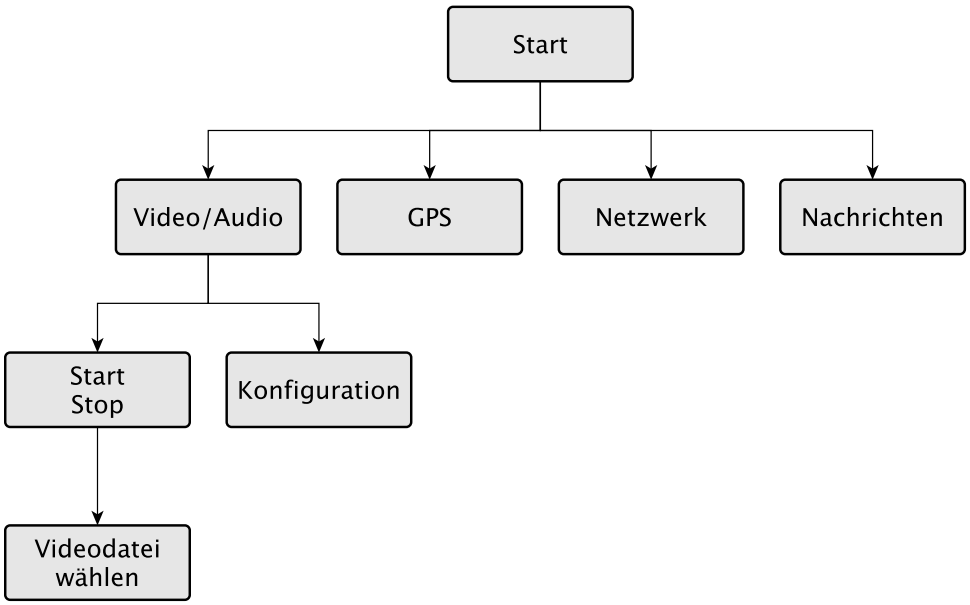


Abbildung 3.7.: Baumartige Struktur der Benutzeroberfläche

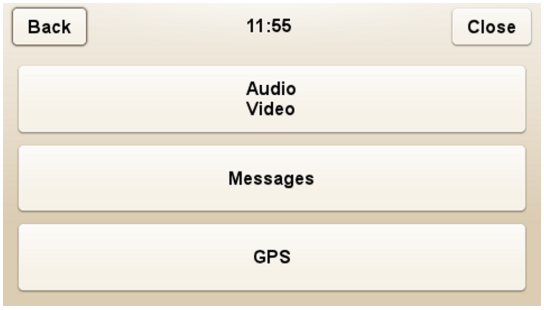


Abbildung 3.8.: Startdialog der Anwendung

ten Datei beginnt, wird der „Select File" Knopf zu einem „Stop" Knopf, bei dessen Betätigung das Signal `requestPlaybackStop()` emittiert wird.

Der in Abbildung 3.9b gezeigte „Config"-Reiter zeigt den Dialog zur Konfiguration der Video- und Audioeigenschaften an. Über die Checkbox „Stream Record" kann der Benutzer wählen, ob eine Aufnahme auch live über die Netzwerkverbindung gesendet werden soll. Die Checkbox „Record Audio" legt fest, ob der Ton mitaufgezeichnet wird. Im Abschnitt „Resolution" kann der Anwender die Videoauflösung wählen, in der die Aufnahme erstellt wird. Der Abschnitt „Bitrate" dient zur Wahl der Bitrate, in der die Aufnahmen kodiert wird. Die Lautstärke der Tonwiedergabe kann über den „Audio Volume" Regler eingestellt werden. Alle Eigenschaften werden in den QSettings-Schlüsseln mit dem Prefix `mfa/media/record/*` gespeichert (siehe Tabelle 3.3).

(a) Aufnahmen starten, Videos wiedergeben

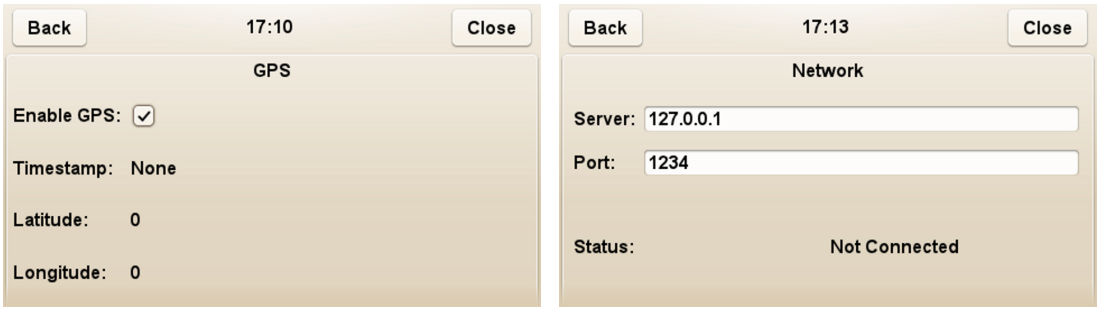
(b) Konfiguration von Video und Ton

(c) Dialog zur Auswahl des abzuspielenden Videos

Abbildung 3.9.: Dialoge zur Konfiguration der Video und Toneigenschaften

GPS: Der GPS-Dialog ist in Abbildung 3.10a dargestellt. Mit der Checkbox „Enable GPS" kann der Benutzer die Aufzeichnung der GPS-Signale steuern. Bei Betätigung der Checkbox wird, je nach Zustand der Checkbox, das Signal `enableGps()` oder das Signal `disableGps()` emittiert. Die Felder „Latitude" und „Longitude" beschreiben die aktuelle GPS-Position, die der Dialog über den Slot `setGpsInfo(Timestamp, Latitude, Longitude)` vom GpsClient empfängt.

Netzwerk: Im Netzwerk-Dialog (siehe Abbildung 3.10b) kann der Benutzer die Verbindungsparameter einstellen, die von der Anwendung zum Verbindungsaufbau mit dem Server verwendet werden. Der Verbindungsaufbau erfolgt automatisch beim Start der Anwendung und muß nicht vom Benutzer initiiert werden. Über das Eingabefeld „Server" kann der Benutzer die IP-Adresse oder den Host-Namen des Servers eingeben. In das Eingabefeld „Port" kann der Benutzer die Portnummer des Servers eintragen. Ändert der Benutzer eine Einstellung wird das Signal `networkSettingsChanged()` emittiert. Die vom Benutzer eingegebenen Daten werden in den QSettings-Schlüsseln mit dem Prefix `mfa/net/*` gespeichert (siehe Tabelle 3.3). Über die Slots `updateStatusConnected()` und `updateStatusDisconnected()` erhält der Netzwerk-Dialog Informationen über den Verbindungsstatus vom ChatClient. Konnte der ChatClient eine Verbindung aufbauen, wir der Text im Feld „Status" auf „Connected" gesetzt. Falls die Verbindung zum Server abbricht oder noch keine Verbindung aufgebaut werden konnte, wird der Text „Not Connected" angezeigt.



(a) GPS-Dialog (b) Konfiguration der Netzwerkparameter

Abbildung 3.10.: Links der GPS-Dialog, rechts der Netzwer-Dialoge

Nachrichten: Der Nachrichten-Dialog (siehe Abbildung 3.11) enthält ein Textfeld, in dem die vom Server empfangenen Nachrichten angezeigt werden. Die Nachrichten sind chronologisch sortiert, die neueste Nachricht befindet sich am Anfang des Textfeldes. Eine neue Nachricht wird über den Slot `addMessage(DateTime, Text)` hinzugefügt. Der Slot wird durch das Signal `textMessage(DateTime, Text)` des ChatClient aktiviert.

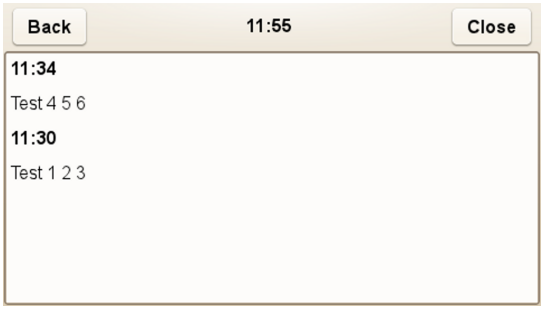


Abbildung 3.11.: Nachrichten-Dialog

Texteingabe: Sobald der Anwender ein Texteingabefeld aktiviert, öffnen sich die in den Abbildungen 3.12a und 3.12b dargestellte virtuelle Tastatur. Über die beiden Reiter am unteren Bildschirmrand kann der Benutzer zwischen der Eingabe von Buchstaben oder Zahlen (sowie Sonderzeichen) wechseln. Um die Tastatur zu schließen und den getippten Text in das entsprechende Eingabefeld zu übernehmen, muß der Benutzer den „Ok" Knopf rechts oben drücken.

3.4.4. ChatClient

Die Klasse `ChatClient` ermöglicht es der Anwendung, Nachrichten vom Server zu empfangen und an diesen zu senden. Über den Slot `connectToServer()` kann der ChatClient angewiesen werden, eine Verbindung zum Server aufzubauen. Die zum Verbindungsaufbau notwendigen Parameter werden aus den QSettings-Schlüsseln mit dem Prefix `mfa/net/*` eingelesen (siehe Tabelle 3.3).

(a) Buchstaben (b) Sonderzeichen und Zahlen

Abbildung 3.12.: Die virtuelle Tastatur

Der Zustand der Verbindung wird mit Hilfe von TCP Keep Alive überwacht. Sollte die Verbindung abbrechen, versucht der ChatClient automatisch, sie wieder aufzubauen. Falls sich der Zustand der Verbindung ändert, emittiert der ChatClient die Signale `connected()` oder `disconnected()`, um das GUI über die Zustandsänderung zu informieren.

Nach einem erfolgreichen Verbindungsaufbau zum Server erhält der MFA von diesem eine Konfigurationsnachricht. Diese enthält den Video- und den Audioport, an die der MFA Video- und Audiopakete einer Liveübertragung schicken soll. Die Ports werden in den in Tabelle 3.3 beschriebenen QSettings-Schlüsseln `mfa/media/net/*` gespeichert und über das Signal `configMessage(VideoPort, AudioPort)` weitergegeben.

Vom Server empfangene Textnachrichten sollen am Display des MFA angezeigt werden. Sobald der ChatClient eine Textnachricht empfängt, wird deren Text über das Signal `textMessage(DateTime, Text)` an den Nachrichten-Dialog des GUI weitergegeben.

3.4.5. GpsClient

Der GpsClient liest Positionsdaten des GPS über die Location API der Qt Mobility Bibliothek ein. Für jede neue Position emittiert er das Signal `newPosition(TimeStamp, Longitude, Latitude)`, das einen Zeitstempel und die Koordinaten der Position enthält. Über die Slots `enable()` und `disbale()` kann die Aufzeichnung von GPS-Daten aktiviert oder deaktiviert werden. Die Slots sind mit den entsprechenden Signalen des GPS-Dialoges verbunden. Im QSettings-Schlüssel `mfa/gps/enabled` ist der aktuelle Zustand des GpsClient gespeichert.

3.4.6. MfaApplication

Nachdem die Anwendung gestartet wurde, erzeugt sie ein Objekt der Klasse MfaApplication. Dieses erzeugt die weiteren Objekt, wie in Abbildung 3.4 dargestellt, und verbindet deren Slots und Signale mit einander. Sobald die Initialisierung abgeschlossen ist, wird der ChatClient durch Aufruf der Methode `connectToServer()` angewiesen, eine Verbindung

zum Server aufzubauen. Die Anwendung zeigt die grafische Benutzeroberfläche und wartet auf Aktionen des Benutzers.

QSettings-Schlüssel	Verwendung
`mfa/media/playback/filename`	Name der wiederzugebenden Datei.
`mfa/media/playback/x`	Horizontale Position des Videofensters.
`mfa/media/playback/y`	Vertikale Position des Videofensters.
`mfa/media/playback/width`	Horizontale Auflösung des Videofensters.
`mfa/media/playback/height`	Vertikale Auflösung des Videofensters.
`mfa/media/record/filename`	Dateiname der aktuellen Aufnahme.
`mfa/media/record/dir`	Verzeichnis in dem Aufnahmen abgelegt werden.
`mfa/media/record/width`	Horizontale Auflösung des Videobildes.
`mfa/media/record/height`	Vertikale Auflösung des Videobildes.
`mfa/media/record/framerate`	Bildwiederholrate des Videobildes.
`mfa/media/record/videoBitrate`	Bitrate des Videoencoders.
`mfa/media/record/audioBitrate`	Bitrate des Audioencoders.
`mfa/media/record/samplerate`	Abtastrate der Tonaufzeichnung.
`mfa/media/record/stream`	Aufzeichnung per Netzwerk übertragen.
`mfa/media/net/host`	IP-Adresse oder Hostname des Ziels.
`mfa/media/net/videoPort`	Port, an den die Videopakete gesendet werden.
`mfa/media/net/audioPort`	Port, an den die Audiopakete gesendet werden.
`mfa/net/server`	Host-Name oder IP-Adresse des Servers.
`mfa/net/port`	Port des Servers.
`mfa/gps/enabled`	Zustand des GPS-Empfängers.

Tabelle 3.3.: Von der Anwendung verwendete QSettings-Schlüssel und deren Bedeutung.

4. Implementierung der Software

4.1. Entwicklungsumgebung

4.1.1. Hardware

Zum Startzeitpunkt des Software-Projekts war die MFA-Hardware noch nicht fertiggestellt. Daher wurde während der ersten Entwicklungsphase ein i.MX27 Platform Development Kit (PDK) als Enwicklungsplattform verwendet. Das PDK verfügt über die selbe Hardware-Grundausstattung wie der MFA, besitzt aber kein UMTS-Modem, keinen einsatzfähigen WLAN-Chip und keinen GPS-Empfänger. Um während der Programmierung der Anwendung dennoch auf eine funktional möglichst vollständige Hardwareplattform zurückgreifen zu können, wurde das PDK um externe Komponenten ergänzt. Als Ersatz für das UMTS-Modem diente ein PCI Express Minicard Development Kit der Firma Sierra Wireless. Diese Development Kit verwendet den selben Chipsatz, der auch beim MFA Verwendung findet. Es wird über USB an das PDK angeschlossen. Der fehlende WLAN-Chip wurde durch einen USB-WLAN-Stick nachgerüstet. Als GPS-Empfänger kam eine, ebenfalls per USB angeschlossene, GPS-Maus zum Einsatz. Zusammen mit diesen externen Komponenten bildet das PDK ein zum MFA funktional äquivalentes System. Ein Bild des PDK ist in Abbildung 4.1 dargestellt. Diese Testplattform wurde nur zu Beginn des Projekts verwendet und später, mit der Verfügbarkeit des MFA, durch diesen ersetzt. Die erstellte Software läuft, dank der Abstraktion der Hardwareunterschiede durch den Linux-Kernel, sowohl am PDK als auch am MFA. Als Entwicklungsrechner wird ein handelsüblicher Desktop-PC eingesetzt.

4.1.2. Software

Für die Entwicklung der MFA-Anwendung finden folgende Entwicklungswerkzeuge und Bibliotheken Verwendung:

- Gnu Compiler Collection [1]
- Qt
- GStreamer
- OpenEmbedded [2]
- QtCreator

[1] `http://gcc.gnu.org/`
[2] `http://www.openembedded.org/`

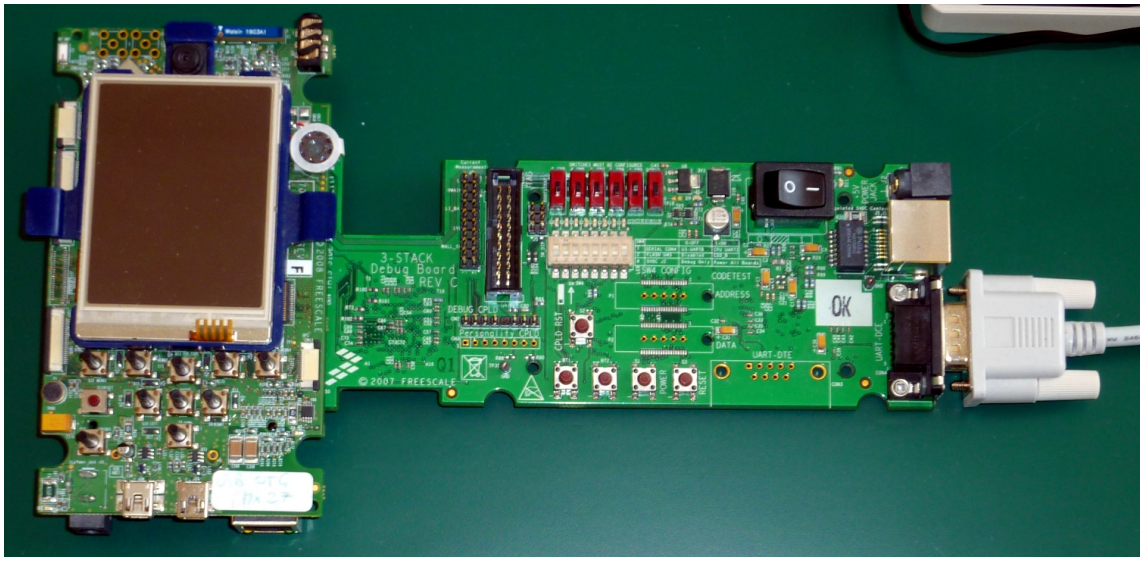


Abbildung 4.1.: Das i.MX27 PDK.

Compiler

Die Ziel-Hardware, das sogenannte Target, auf der die Anwendung ausgeführt werden soll, verfügt nur über eine sehr begrenzte Hardwareressourcen. Die Geschwindigkeit der CPU und die Größe von RAM und Festspeicher sind zu gering, um Kompiliervorgänge in zufriedenstellender Zeit ausführen zu können. Daher findet die Entwicklung der Anwendung auf einem handelsüblichen PC, auf dem das Linux-Betriebsystem Ubuntu läuft, statt. Dieser Entwicklungsrechner wird im folgenden als Host bezeichnet. Um am Host eine Anwendung für das Target erstellen zu können, ist eine Cross-Compiler-Toolchain notwendig. Diese enthält einen Compiler, der am Host ausgeführt wird, aber Binärcode für das Target erzeugt. Die für dieses Projekt gewählte Cross-Compiler-Toolchain ist die GNU Compiler Collection in Version 4.3. Die Collection enthält einen C- und einen C++-Compiler, die optimierten Binärcode für die ARM-CPU des MFA erzeugen können.

Bibliotheken

Neben der Anwendung selbst, müssen auch alle von der Anwendung verwendeten Bibliotheken für das Target kompiliert werden. Von der Anwendung direkt verwendete Bibliotheken sind Qt und GStreamer. Qt ist für die Kompilierung mittels Cross-Compiler vorbereitet, folgender Aufruf erzeugt eine gültige ARM-Version der Bibliothek:

```
./configure -v -prefix  /usr/local/Trolltech/Qt-4.7.1-arm-mfa \
  -opensource -release -platform linux-g++ \
  -embedded arm -xplatform qws/linux-arm-g++ -nomake examples \
  -nomake demos -nomake docs -nomake translations \
  -qt-zlib -qt-mouse-tslib -qt-libjpeg -qt-libmng -qt-libpng \
  -qt-libtiff -qt-gif -qt-gfx-multiscreen -qt-gfx-transformed \
  -glib -gstreamer \
```

```
-I$DEV_IMAGE/usr/include -I$DEV_IMAGE/usr/include/glib-2.0/ \
-I$DEV_IMAGE/usr/lib/glib-2.0/include/ \
-I$DEV_IMAGE/usr/include/libxml2/ \
-I$DEV_IMAGE/usr/include/gstreamer-0.10
-L$DEV_IMAGE/usr/lib -L$DEV_IMAGE/lib \
-L$DEV_IMAG/lib -L$DEV_IMAGE/usr/lib \
-reduce-relocations -shared \
-no-phonon -no-phonon-backend -no-multimedia -no-qt3support \
-no-dbus -no-cups -no-openvg -no-opengl -no-nas-sound \
-no-gtkstyle -no-nis -no-declarative -no-scripttools \
-no-audio-backend -no-accessibility -no-svg -no-webkit \
-no-stl -no-exceptions -no-sql-mysql -no-sql-odbc \
-no-sql-psql -no-sql-ibase -no-mouse-pc -no-xmlpatterns \
-no-javascript-jit -no-script -fast
```

Durch diese Konfiguration werden alle vom MFA nicht benötigten Funktionen der äußerst umfangreichen Qt-Bibliothek deaktiviert. Dies reduziert die Größe der Bibliothek und verkürzt die Startzeit der Anwendung.

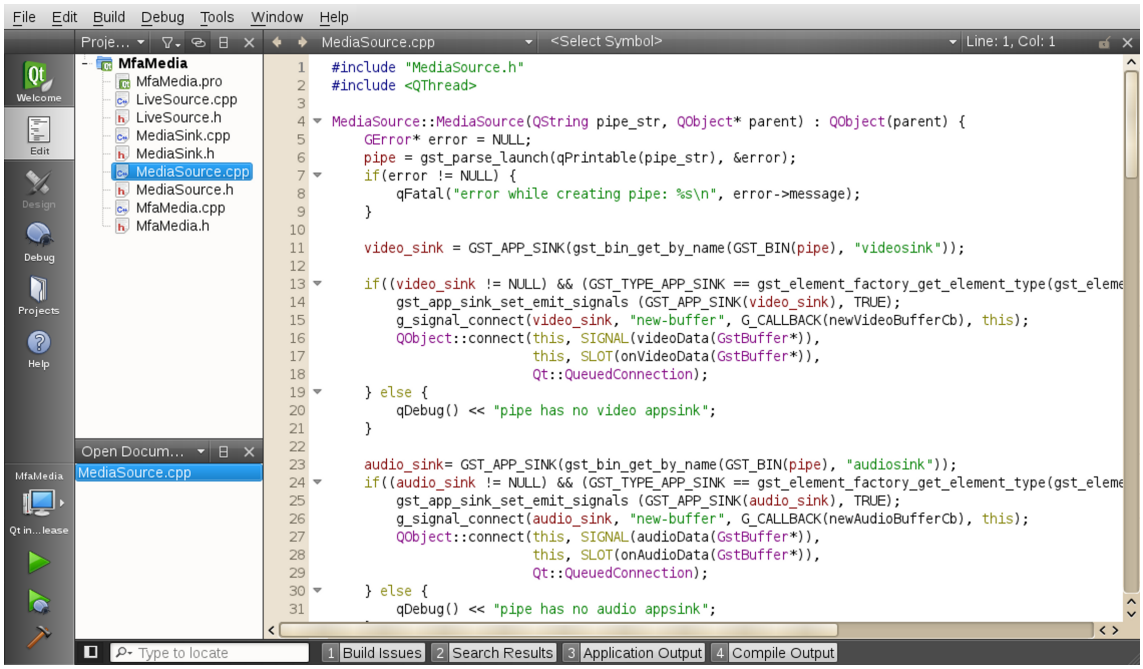
Die Bibliothek GStreamer hat, aufgrund der umfangreichen Unterstützung diverser Audio- und Videoformate, viele Abhängigkeiten zu weiteren Bibliotheken, die alle für das Target kompiliert werden müssen. All diese Abhängigkeiten korrekt zu erfüllen ist eine zeitraubende Aufgabe. Zudem sind nicht alle Bibliotheken für die Übersetzung durch einen Cross-Compiler vorbereitet, was manuelle Eingriffe in deren Quellcode erforderlich machen würde. Um diese langwierige Aufgabe dennoch einfach und automatisiert durchführen zu können, wird OpenEmbedded eingesetzt.

Build-Framework

OpenEmbedded ist ein Framework zur Erstellung kompletter Linux-Distributionen für Embedded Systems. Es automatisiert die Kompilierung der am Target benötigten Softwarepakete. Zudem kann es ein vollständiges, zur Entwicklung der MFA-Anwendung geeignetes, am Host nutzbares, Software Development Kit (SDK) generieren. Das SDK enthält neben dem Cross-Compiler, Qt und GStreamer alle weiteren, benötigten Bibliotheken. Die zur Erstellung des SDK notwendige Vorgehensweise ist im OpenEmbedded-Benutzerhandbuch dokumentiert [22].

IDE

Für die Programmierung der Anwendung wird das Qt Creator Integrated Development Environment (IDE) verwendet. Es bietet alle, von ähnlichen Entwicklungsumgebungen bekannten, Komfortfunktionen, wie zum Beispiel Syntax-Highlighting, automatische Codevervollständigung oder Debugger. Zudem unterstützt es Qt-spezifische Syntax-Erweiterungen wie den Signals/Slots-Mechanismus und enthält einen leistungsfähigen GUI-Designer. Ein Screenshot der IDE ist ein Abbildung 4.2 dargestellt.




```
#include "MediaSource.h"
#include <QThread>

MediaSource::MediaSource(QString pipe_str, QObject* parent) : QObject(parent) {
    GError* error = NULL;
    pipe = gst_parse_launch(qPrintable(pipe_str), &error);
    if(error != NULL) {
        qFatal("error while creating pipe: %s\n", error->message);
    }

    video_sink = GST_APP_SINK(gst_bin_get_by_name(GST_BIN(pipe), "videosink"));

    if((video_sink != NULL) && (GST_TYPE_APP_SINK == gst_element_factory_get_element_type(gst_eleme
        gst_app_sink_set_emit_signals (GST_APP_SINK(video_sink), TRUE);
        g_signal_connect(video_sink, "new-buffer", G_CALLBACK(newVideoBufferCb), this);
        QObject::connect(this, SIGNAL(videoData(GstBuffer*)),
                         this, SLOT(onVideoData(GstBuffer*)),
                         Qt::QueuedConnection);
    } else {
        qDebug() << "pipe has no video appsink";
    }

    audio_sink= GST_APP_SINK(gst_bin_get_by_name(GST_BIN(pipe), "audiosink"));
    if((audio_sink != NULL) && (GST_TYPE_APP_SINK == gst_element_factory_get_element_type(gst_eleme
        gst_app_sink_set_emit_signals (GST_APP_SINK(audio_sink), TRUE);
        g_signal_connect(audio_sink, "new-buffer", G_CALLBACK(newAudioBufferCb), this);
        QObject::connect(this, SIGNAL(audioData(GstBuffer*)),
                         this, SLOT(onAudioData(GstBuffer*)),
                         Qt::QueuedConnection);
    } else {
        qDebug() << "pipe has no audio appsink";
```

Abbildung 4.2.: Die MFA-Anwendung in der QtCreator IDE.

4.2. Laufzeitumgebung

Die Laufzeitumgebung der Anwendung bilden ein mit OpenEmbedded generiertes Dateisystem und ein Linux-Kernel in Version 2.6.22. Der Kernel wurde von Freescale, dem Hersteller des i.MX27, und von Bluetechnix, dem Hardware-Fertiger des MFA, durch zahlreiche Patches an den MFA angepasst.

Um den Kernel auf dem MFA starten zu können, ist ein Bootloader notwendig. Gebräuchliche Bootloader sind U-Boot [3] und RedBoot [4]. Am MFA findet RedBoot Verwendung, der, wie auch der Kernel, von Freescale und Bluetechnix an die MFA-Hardware angepasst wurde. Sobald der MFA mit Strom versorgt wird, beginnt der Bootvorgang und der Bootloader wird gestartet. Der Bootloader lädt das Kernel-Image aus dem NAND-Flashspeicher des MFA in dessen RAM und startet es. Der Kernel initialisiert die Hardware und bindet das Dateisystem ein. Das Dateisystem kann am MFA entweder im NAND-Flash, im NOR-Flash, oder auf einer SD-Karte gespeichert sein. Zudem ist es möglich, ein Dateisystem via NFS über das Netzwerk einzubinden, was vorallem während der Entwicklung der Anwendung von großem Vorteil ist.

Für den produktiven Einsatz ist die SD-Karte den Flashspeichern vorzuziehen, da der Anwender das System durch Wechsel der SD-Karte sehr einfach selbst aktualisieren kann. Um den Flashspeicher mit einem neuen Dateisystem zu beschrieben, sind eine Verbindung

[3] `http://www.denx.de/wiki/U-Boot` (4.5.2011)
[4] `http://ecos.sourceware.org/redboot` (4.5.2011)

zur seriellen Schnittstelle eines PC und Kenntnisse in der Bedienung des Bootloaders erforderlich. Durch eine falsch ausgeführte Aktualisierung des Flashspeichers kann der MFA dauerhaft zerstört werden.

Sobald das Dateisystem vom Kernel eingebunden wurde, übergibt dieser die Kontrolle an den init-Prozess. Dieser startet die Konfigurationsskripte und schließlich die MFA-Anwendung. In Abbildung 4.3 wird der Boot-Vorgang veranschaulicht.

Abbildung 4.3.: Der Bootvorgang am MFA

4.3. Programmierung der Anwendung

Die Implementierung der Anwendung folgt den im Design-Kapitel aufgestellten Überlegungen. Während der Implementierung musste das Design nur leicht verändert und erweitert werden. So war ursprünglich geplant, die GPS-Daten über einen Daemon (gpsd) per Inter Process Communication (IPC) einzulesen. Die Kommunikation zwischen der Anwendung und dem Daemon erwies sich allerdings als problematisch, weswegen dieser durch die Bibliothek Qt Mobility ersetzt wurde. Auch der Einsatz von QSettings-Objekten zur Speicherung von Parametern war ursprünglich nicht geplant. Während der ersten Phase der Implementierung erwies sich dies als praktikabel und führte daher zu einem Re-Design der Anwendung.

4.3.1. Multimedia-Bibliothek

Der Zugriff auf die Video- und Audiohardware des MFA erfolgt über das Multimedia Frameworks GStreamer.

GStreamer

GStreamer ist eine umfangreiche Sammlung von Plugins zur Verarbeitung von Video- und Tondaten, die über eine C-API programmiert werden können. Plugins stellen sogenannte „Elemente" zur Verfügung, von denen jedes eine bestimmte Aufgabe erfüllt. Elemente besitzen Eingänge, über die sie einen Datenstrom aufnehmen, und Ausgänge, über die sie den manipulierten Datenstrom weitergeben. Der Ausgang eines Elements ist mit dem Eingang des nachfolgenden Elements verbunden, so daß sich lange, pipelineartig Ketten von Elementen bilden. In Abbildung 4.4 ist ein Beispiel aus der GStreamer Dokumentation für eine derartige Pipeline abgebildet.

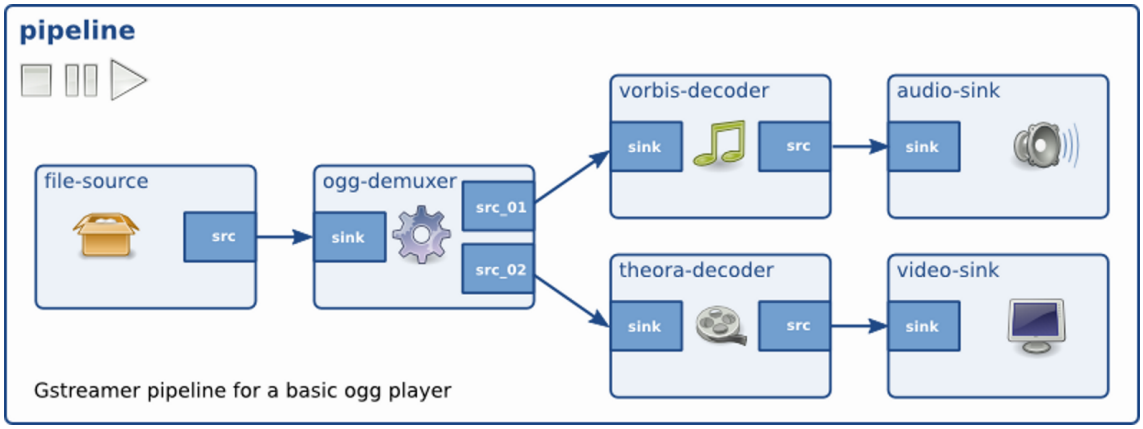


Abbildung 4.4.: Ein Beispiel einer GStreamer-Pipeline [GStreamer Application Development Manual [23, S. 7].

Eine Pipeline besteht aus verschiedenen Elementen:

- Source Elemente: Diese bilden den Anfang einer GStreamer-Pipeline. Sie stellen Daten aus einer Quelle, zum Beispiel aus einer Datei oder vom Kamerainterface, bereit.
- Sink Elemente: Sie befinden sich am Ende einer Pipeline. Ihre Aufgabe ist es, die verarbeiteten Daten in einer Datei zu speichern, über ein Netzwerk zu versenden oder in einen Datenpuffer der Anwendung zu schreiben.
- Multiplexer: Durch sie können mehrere Datenströme kombiniert werden, etwa um einen Video- und einen Audiodatenstrom zusammen in einer AVI-Datei speichern zu können.
- Demultiplexer: Trennt mehrere, verwobene Datenströme wieder auf, zum Beispiel um den Videodatenstrom und den Audiodatenstrom aus einer AVI-Datei zu extrahieren.
- Encoder: Die Kodierer können einen Datenstrom in ein bestimmtes Format kodieren, zum Beispiel Tondaten in MP3, oder Videodaten in H.264.
- Decoder: Sie wandeln einen kodierten Datenstrom in einen nicht kodierten um.
- Filter/Effekte/Konverter: Diese Elemente manipulieren den Datenstrom auf bestimmte Weise, etwa um ein Videobild zu invertieren, oder die Abtastrate eines Audiosignal zu verändern.

Jede Pipeline besitzt einen Bus, über den sie Nachrichten mit der Anwendung austauschen kann. Eine Pipeline erzeugt Nachrichten für verschiedenste Ereignisse, etwa „End of Stream" wenn das Ende des abzuspielenden Datenstromes erreicht wurde, oder „Error" falls ein Fehler auftritt. Um Nachrichten vom Bus zu empfangen, muß die Anwendung eine Callback-Funktion beim Bus registrieren. Diese wird vom Bus für jede neue Nachricht aufgerufen.

Folgendes Beispiel zeigt die Beschreibung eine GStreamer-Pipeline, die ein Videosignal mit einer Auflösung von horizontal 640 und vertikal 480 Pixeln von der Kamera einliest,

anschließend eine H.264-Kodierung mit einer Bitrate von 1024 kbit durchführt und das Ergebnis in einen Datenpuffer der Anwendung schreibt. Diese Beschreibung kann direkt an die GStreamer-Bibliothek übergeben werden, die daraus eine gültige Pipeline erstellt.

```
mfw_v4lsrc capture-width=640 capture-height=480 !
mfw_vpuencoder codec=std_avc bitrate=1024 !
appsink name=videosink
```

Durch den Einsatz von GStreamer kann eine aufwändige, direkte Programmierung der Video- und Audiohardware vermieden werden. Zudem ist eine einfache Weiterverarbeitung der Video- und Tondaten durch die zahlreich vorhandenen Plugins möglich.

MediaSource

Die MediaSource-Klasse bildet die Basis für alle Klassen, die Video- oder Audiodaten produzieren. Sie kapselt eine GStreamer-Pipeline in eine C++-Klasse. Die Events und Callback-Funktionen der GStreamer-Pipeline werden über Signals und Slots durch die Klasse weiter gegeben. Dies ermöglicht die einfach Verwendung von MediaSource Objekten zusammen mit anderen Qt-Objekten. Um eine reibungslose Zusammenarbeit zwischen Qt und GStreamer zu bewerkstelligen, muß die Qt-Bibliothek mit GLib-Unterstützung konfiguriert und kompiliert werden.

`MediaSource(QString pipe_str, QObject* parent)`: Der Parameter `pipe_str` des Konstruktors enthält die Beschreibung der gekapselten GStreamer-Pipeline. Er wird direkt an die GStreamer-Bibliothek übergeben, und muß daher der GStreamer-Syntax zur Beschreibung von Pipelines entsprechen. Zudem müssen in der Pipeline mindestens ein appsink-Element mit Namen „videosink" oder ein appsink-Element mit Namen „audiosink", oder beide, enthalten sein. Dies ist notwendig, um die Video- und Audiodatenquellen einer Pipeline automatisch identifizieren zu können. Erfüllt eine Pipeline diese Anforderungen nicht, wird eine Fehlermeldung ausgegeben.

Die appsink bildet eine Schnittstelle zum Austausch von Daten zwischen der GStreamer-Pipeline und der MediaSource. Der Datenaustausch erfolgt über Callback-Funktionen, die von der MediaSource bei den appsink-Elementen mit Namen „videosink" und „audiosink" registriert werden. Sobald die Pipeline die Verarbeitung eines Datenpuffers abgeschlossen hat, ruft die appsink ihre Callback-Funktion auf, und übergibt dieser den Datenpuffer. Für jeden Datenpuffer emittiert die Callback-Funktion das `newVideoData(data)` oder das `newAudioData(data)` Signal. Dadurch werden die Daten an alle, entsprechend dem Signals/Slots-Mechanismus registrierten, MediaSink-Objekte übergeben.

Zudem registriert der Konstruktor eine Callback-Funktion beim Bus der Pipeline, um Nachrichten von diesem empfangen zu können.

`void addMediaSink(MediaSink* sink):` Durch Aufruf dieser Funktion wird eine MediaSink bei der MediaSource registriert. Die Signale `newVideoData(data)` und `newAudioData(data)` der MediaSource werden mit den Slots `onNewVideoData(data)` und `onNewAudioData(data)` der MediaSink verbunden. Die Anzahl an gleichzeitig registrierten MediaSink-Objekten ist nicht limitiert. MediaSink-Objekte können auch zu einer bereits gestarteten, aktiven MediaSource hinzugefügt werden. Falls die MediaSink vor Aufruf dieser Funktion bereits bei der MediaSource registriert war, passiert nichts.

`void removeMediaSink(MediaSink* sink):` Diese Funktion entkoppelt ein MediaSink-Objekt von der MediaSource, indem die Signal/Slot-Zuordnung aufgehoben wird. War die MediaSink vor Aufruf dieser Funktion nicht bei der MediaSource registriert, passiert nichts. Eine MediaSink kann auch von einer bereits gestarteten, aktiven MediaSource entfernt werden.

`bool start():` Diese Funktion startet die GStreamer-Pipeline und veranlasst die MediaSource, Daten an die registrierten MediaSink-Objekte zu senden. Sobald die Initialisierung der GStreamer-Pipeline abgeschlossen ist und die Pipeline Daten produziert, emittiert die MediaSource das `playing()` Signal. Falls die Pipeline erfolgreich gestartet werden konnte, retourniert die Funktion „true", sonst „false".

`void stop():` Der Aufruf dieser Funktion stoppt die MediaSource. Sobald die GStreamer-Pipeline gestoppt und freigegeben wurde, emittiert die MediaSource das `stopped()` Signal. Das Signal wird, ohne vorhergehenden Aufruf von stop(), auch dann emittiert, wenn vom Bus der GStreamer-Pipeline eine End-of-Stream-Nachricht empfangen wurde.

MediaSink

Die MediaSink-Klasse ist die Basis aller Klassen, die Video- und Tondaten wiedergeben, oder auf sonstige Art und Weise verarbeiten. Sie kapselt eine GStreamer-Pipeline in eine C++-Klasse und bindet deren Events und Callback-Funktionen an Signals und Slots. Dies ermöglicht, analog zur MediaSource-Klasse, eine einfach Verwendung von MediaSink Objekten zusammen mit anderen Qt-Objekten.

`MediaSink(QString pipe_str, QObject* parent):` Der Konstruktor der MediaSink-Klasse erstellt die durch den Parameter `pipe_str` beschriebene GStreamer-Pipeline. Dies passiert analog zur Vorgehensweise im Konstruktor der MediaSource-Klasse, allerdings mit dem Unterschied, daß in der Beschreibung der Pipeline keine appsink-Element, sondern appsrc-Elemente vorhanden sein müssen. Es müssen entweder eine appsrc mit Namen „videosrc" oder eine appsrc mit Namen „audiosrc" oder beide in der Beschreibung enthalten sein. Die appsrc bildet das Gegenstück zur appsink. Sie nimmt über einen Funktionsaufruf Daten von der MediaSource entgegen und leitet sie an die Pipeline weiter. Der Konstruktor registriert eine Callback-Funktion beim Bus der Pipeline, um Nachrichten von diesem empfangen zu können.

`void onNewVideoData(data):` Dieser Slot wird mit dem `newVideoData(data)` Signal der MediaSource verbunden, wenn die MediaSink bei einer MediaSource registriert wird. Der Slot wird aufgerufen, sobald an der MediaSource neue Videodaten verfügbar sind. Die im Parameter `data` enthaltenen Daten werden an die appsrc der GStreamer-Pipeline übergeben und von dieser weiterverarbeitet.

`void onNewAudioData(data):` Diese Funktion entspricht der `onNewVideoData(data)` Funktion, verarbeitet aber Audiodaten. Daher ist sie mit dem `newAudioData(data)` Signal der MediaSource verbunden und wird aufgerufen, sobald neue Tondaten verfügbar sind.

`bool start():` Diese Funktionen startet die von der MediaSink gekapselte GStreamer-Pipeline. Bevor sie aufgerufen wird, ignoriert die Pipeline alle über die Slots `onNewVideoData(data)` und `onNewAudioData(data)` empfangenen Daten. Falls die Pipeline erfolgreich gestartet werden konnte, retourniert die Funktion „true", sonst „false".

`void stop():` Diese Funktion stoppt die von der MediaSink gekapselte GStreamer-Pipeline. Nach Aufruf dieser Funktion haben Aufrufe der beiden Slots `onNewVideoData(data)` und `onNewAudioData(data)` solange keinen Effekt, bis wieder die Funktion `start()` aufgerufen wird.

LiveSource

Die LiveSource-Klasse basiert auf der MediaSource-Klasse. Als Quelle für live aufgenommene Video- und Tondaten steuert sie das Mikrophon, die Kamera und VPU.

`LiveSource(VideoConfig vconf, AudioConfig aconf, QObject* parent):` Der Konstruktor der LiveSource-Klasse ruft den Konstruktor der Basisklasse MediaSource auf und übergibt diesem die Beschreibung der in Abbildung 4.5 dargestellten GStreamer-Pipeline. Die Beschreibung wird aus den Werten der in `vconf` und `aconf` enthaltenen Parameter gebildet. Diese enthalten die schon im Abschnitt 3.4.2 beschriebenen Parameter.

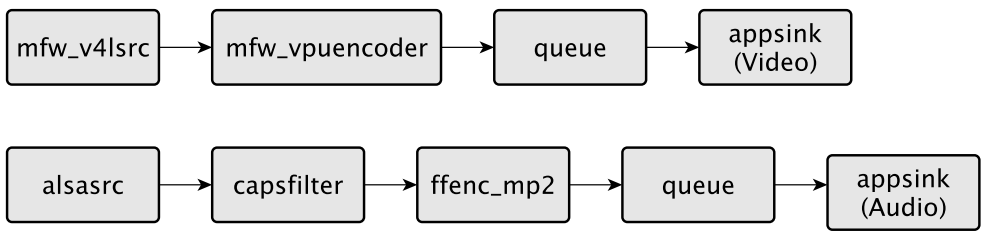


Abbildung 4.5.: GStreamer LiveSource Pipeline

Die von der Kamera aufgezeichneten Bilder werden von der VPU in das H.264-Format kodiert und über ein queue-Element an die appsink weitergeben. Eine queue dient als Datenpuffer und gewährleistet die synchrone Verarbeitung von Video- und Tondatenstrom. Die vom Mikrophon aufgezeichneten Töne werden von einem Softwareencoder in das MP2-Format kodiert und ebenfalls über eine queue an die appsink weitergeben.

FileSource

Die Klasse FileSource basiert auf der MediaSource-Klasse. Sie liest Video- und Audiodaten aus einer Datei ein und leitet sie an eine auf der MediaSink basierenden Klasse weiter.

`FileSource(QString filename, QObject* parent):` Der Konstruktor der Klasse erwartet den Namen der zu öffnenden Datei als Parameter. Aus diesem generiert er die textuelle Beschreibung der GStreamer-Pipeline und ruft mit dieser den Konstruktor der Basisklasse MediaSource auf. Die Struktur der so erstellten Pipeline ist in Abbildung 4.6 dargestellte.

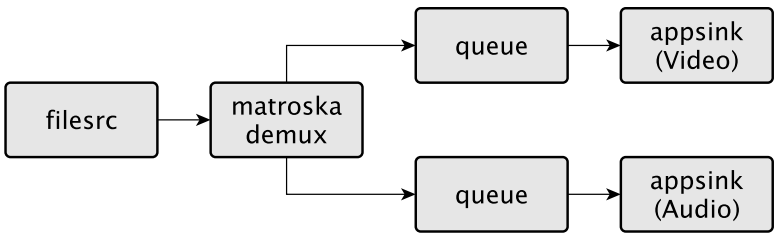


Abbildung 4.6.: GStreamer FileSource Pipeline

Die durch den Parameter `filename` beschrieben Datei muß ein Matroska Media Container sein. In diesem können Videodatenströme in den Formaten H.264 oder MPEG-4, sowie Tondatenströme in dem Formaten MP3, MP2, alaw oder μlaw enthalten sein. Die Video- und Tondatenströme werden vom GStreamer-Element matroskademux aufgespalten und an die appsink-Elemente weitergegeben.

FileSink

Die Klasse FileSink bildet das Gegenstück zur Klasse FileSource. Sie nimmt einen Video- und einen Audiodatenstrom entgegen und schreibt beide in einen Matroska Media Container.

`FileSink(QString filename, QObject* parent):` Der Konstruktor der Klasse nimmt über den Parameter `filename` den Namen der zu schreibenden Datei entgegen. Aus diesem generiert er die textuelle Beschreibung der in Abbildung 4.7 dargestellten GStreamer-Pipeline. Mit dieser Beschreibung ruft er den Konstruktor der Basisklasse MediaSink auf.

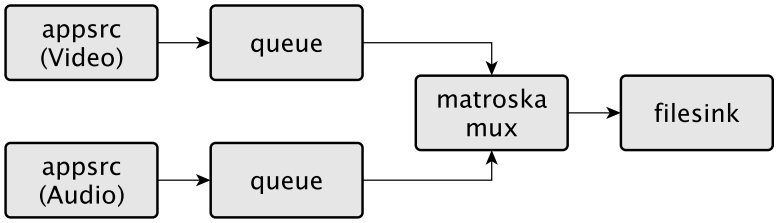


Abbildung 4.7.: GStreamer FileSink Pipeline

Über die beiden appsrc-Elemente nimmt die Klasse eine Video- und einen Audiodatenstrom entgegen. Diese werden vom GStreamer-Element „matroskamux" in einen Matroska Media Container verpackt, der vom filesink-Element in die Datei mit Namen `filename` geschrieben wird.

DisplaySink

Über die Methoden der Klasse DisplaySink können Videodaten am Display des MFA angezeigt und Audiodaten über den Audioausgang des MFA wiedergegeben werden.

`DisplaySink(DisplayConfig conf, QObject* parent):` Der Konstruktor der Klasse DisplaySink ruft den Konstruktor der Basisklasse MediaSink mit der Beschreibung der in Abbildung 4.8 dargestellten GStreamer-Pipeline auf. Die Beschreibung wird mit den im Parameter `conf` gespeicherten Werten gebildet. Eine Beschreibung der Parameter findet sich in Abschnitt 3.4.2.

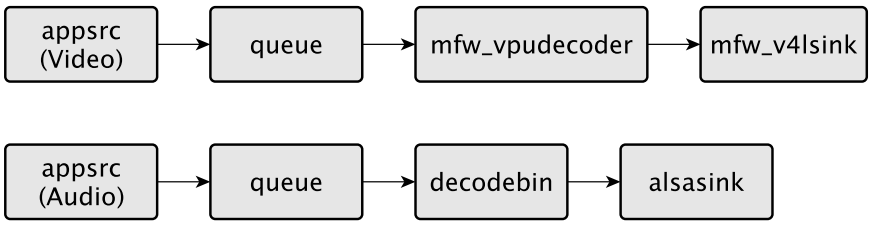


Abbildung 4.8.: GStreamer DisplaySink Pipeline

Die über die appsrc empfangenen Videodaten werden über ein queue an das Decoder-Element mfw_vpudecoder übergeben. Dieses gibt das dekodierte Video an das Element

mfw_v4lsink weiter, das die Videobilder am Bildschirm des MFA anzeigt. Die Tondaten gelangen über eine queue in das Element decodebin. Dieses wählt, abhängig vom Format der Tondaten, automatisch einen passenden Softwaredecoder und wandelt die Daten in ein für die Wiedergabe geeignetes Format um. Das Element alsasink gibt die Daten über den Audioausgang des MFA wieder.

NetworkSink

Die Klasse NetworkSink sendet Video und Ton per UDP an den MFA-Server.

`NetworkSink(NetConfig conf, QObject* parent):` Mit den im Parameter `conf` (siehe 3.4.2) enthaltenen Werten erstellt der Konstruktor der Klasse die Beschreibung der in Abbildung 4.9 dargestellten GStreamer-Pipeline. Diese Beschreibung wird an den Konstruktor der Basisklasse MediaSink übergeben.

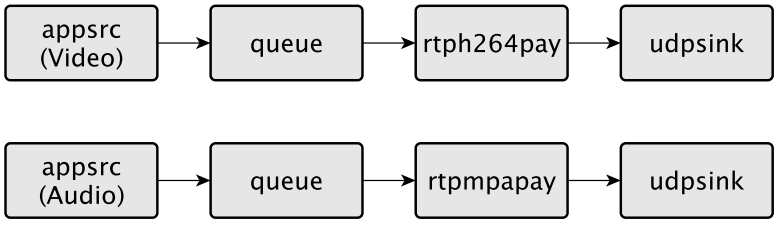


Abbildung 4.9.: GStreamer NetworkSink Pipeline

Von der appsrc gelangen die Videodaten, die im H.264-Format kodiert sein müssen, über eine queue in das GStreamer-Element rtph264pay. Diese verpackt die einzelnen Videodatenpuffer in RTP-Pakete. Das Element udpsink sendet die Pakete per UDP an ihr Ziel. Die Verarbeitung der Tondaten erfolgt analog. Die Audiodaten müssen im MP2-Format kodiert sein. Das Element rtpmpapay verpackt die MP2-Daten in RTP-Pakete, die von einer weiteren udpsink verschickt werden.

MediaControl

Die Klasse MediaControl bildet die Schnittstelle zwischen der Multimedia-Bibliothek und der MFA-Anwendung. Über die Funktionen der Klasse kann die Anwendung Ton- und Videodaten aufzeichnen oder abspielen. Die Methoden der Klasse emittieren die beim Design der MediaControl in Abschnitt 3.4.2 beschriebenen Signale.

`MediaControl(QObject* parent):` Der Konstruktor der Klasse initialisiert das Objekt.

`void playFile()`: Der Slot `playFile()` startet die Wiedergabe einer Video- oder Audiodatei auf dem Bildschirm oder über die Lautsprecher des MFA. Die Funktion erstellt ein Objekt des Typs FileSource und verbindet es mit einem Objekt des Typs DisplaySink. Die zur Erstellung der Objekte notwendigen Parameter werden über die in Tabelle 3.3 beschriebenen QSettings-Schlüssel eingelesen.

`void record()`: Der Slot `record()` startet die Aufnahme von Videobildern und Tönen über die Kamera und das Mikrophon. Hierfür wird ein Objekt des Typs LiveSource mit einem Objekt des Typs FileSink verbunden. Falls der QSettings-Schlüssel `media/record/stream` den Wert „true" hat, wird die LiveSource zudem mit einem Objekt des Typs NetworkSink verbunden. Die zur Erstellung der Objekte notwendigen Parameter werden über die in Tabelle 3.3 definierten QSettings-Schlüssel eingelesen.

`stop()`: Der Slot `stop()` versucht die gerade laufende Wiedergabe oder Aufnahme zu stoppen. Sollte weder eine Wiedergabe, noch eine Aufnahme aktiv sein, passiert nichts.

4.3.2. Grafische Benutzeroberfläche

Die grafische Benutzeroberfläche (engl. Graphical User Interface, GUI) der Anwendung wird mit der Qt4 GUI-Bibliothek realisiert. Diese bietet eine umfangreiche Sammlung an grafischen Elementen und ist auch für den Einsatz in Embedded Systems geeignet.

Framebuffer

Die Darstellung von Videos und GUI soll, wie im Design-Kapitel beschrieben und in Abbildung 3.6 gezeigt, auf die beiden Framebuffer fb0 und fb1 aufgeteilt werden. Beim Start einer Qt-Anwendung kann über die Umgebungsvariable `QWS_DISPLAY` festgelegt werden, auf welchem Framebuffer das GUI angezeigt werden soll. Da das GUI der MFA-Anwendung auf Framebuffer fb1, über Framebuffer fb0, dargestellt werden soll, muß der Wert der Umgebungsvariable `QWS_DISPLAY` vor Programmstart auf `LinuxFb:/dev/fb0` gesetzt werden.

In Qt sind keine Funktionen zur Steuerung der im Design-Kapitel beschriebenen Transparenzeffekte der Framebuffer enthalten. Um diese Effekte nutzen zu können muß der Framebuffer-Treiber des Linux-Kernel direkt programmiert werden. Die hierfür notwendigen Funktionsaufrufe sind in der Klasse `FrameBuffer` gekapselt.

`FrameBuffer(QString dev)`: Die Klasse FrameBuffer ist ein Singleton [20], der Konstruktor der Klasse ist daher privat. Um eine Instanz der Klasse zu erzeugen, muß die Methode `dev()` aufgerufen werden.

`static FrameBuffer* dev(QString dev):` Diese statische Methode erzeugt eine Instanz der FrameBuffer-Klasse, die das im Parameter `dev` angegebenen Framebuffer-Device repräsentiert. Der Parameter `dev` kann am MFA die Werte `/dev/fb0` oder `/dev/fb1` annehmen.

Für jeden Framebuffer des MFA kann nur eine Instanz der Klasse FrameBuffer erzeugt werden. Wird die Methode aufgerufen, prüft Sie, ob für den Parameter `dev` bereits eine Instanz in einer internen Liste gespeichert ist. Falls nicht, wird eine neue Instanz erzeugt, diese in die Liste eingefügt, und retourniert. Falls sich schon eine Instanz in der Liste befindet, wird diese retourniert. So kann sicher gestellt werden, daß nicht zwei, um das selbe Device konkurrierende, Framebuffer-Objekte erstellt werden.

`void fadeIn(int duration):` Die Methode `fadeIn()` verringert die Transparenz des Framebuffers, bis er vollkommen sichtbar ist. Die Transparenz des Framebuffer kann über das ioctl `MX2FB_SET_GBL_ALPHA` gesetzt werden, dem ein Transparenzwert als Parameter übergeben wird. Die Funktion ruft das ioctl in einer Schleife auf. Bei jedem Schleifendurchlauf wird die Transparenzwert schrittweise verringert. Die Anzahl der Schritte hängt vom Parameter `duration` ab. Über ihn kann der Anwender bestimmen, wie lange das einblenden des Framebuffers dauern soll. Der Wert wird in Millisekunden angegeben. Um die restliche Anwendung während des durchlaufens der Schleife nicht zu blockieren, wir bei jeder Iteration die Qt-Funktion `QCoreApplication::processEvents()` aufgerufen. Diese veranlasste Qt, alle zwischenzeitlich aufgetretenen Ereignisse abzuarbeiten.

`void fadeOut(int duration):` Der Ablauf dieser Methode entspricht dem der Methode `fadeIn()`, allerdings mit dem Unterschied, daß die Transparenz des Framebuffers schrittweise erhöht wird. Nach dem Aufruf dieser Funktion ist der Framebuffer vollkommen transparent.

Hauptfenster

Das GUI der Anwendung besteht aus einem Hauptfenster, das die in Abschnitt 3.4.3 beschriebenen Dialoge anzeigt. Das Hauptfenster ist in der Klasse MfaStartWindow implementiert. Es ist von der Qt-Klasse QMainWindow abgeleitet.

Am oberen Bildschirmrand des Hauptfenster befindet sich die Statuszeile. Dieses ist eine Instanz eines QWidget und dient als Container für die Knöpfe „Back" und „Close", sowie die Uhrzeit. Die Knöpfe sind Instanzen der Klasse QPushButton. Deren Signale `clicked()` sind mit Slots in der Klasse MfaStartWindow verbunden. Durch drücken des „Close"-Knopf, wird das Hauptfenster unsichtbar. Durch drücken des „Back"-Button, kann der Benutzer in der Dialog-Hierarchie (siehe Abbildung 3.7) eine Ebene nach oben navigieren.

Unter der Statuszeile befindet sich ein QStackedWidget. Ein QStackedWidget besteht aus mehreren Seiten, von denen jede Container eines Widget ist. Es ist immer nur eine Seite eines QStackedWidget sichtbar. Welche das ist, kann durch Aufruf der Methode `QStackedWidget::setCurrentIndex(int)` bestimmt werden.

Die im Design-Kapitel beschriebenen Dialog bestehen jeweils aus Widgets, die als Container für die eigentlichen Element der Dialog fungieren. Der Konstruktor der Klasse MfaStart-Window instantiiert jedes der Dialog-Widgets, und fügt die Instanzen als neue Seiten in das QStackedWidget ein.

Die bei Programmstart angezeigte Startseite des QStackedWidgets besteht aus den in Abbildung 3.8 gezeigten Buttons. Durch drücken der Buttons kann der Benutzer zu den einzelnen Unterdialogen navigieren. Hierfür wird die entsprechende Seite des QStackedWidget aktiviert. Befindet sich der Benutzer in einem Unterdialog und betätigt den „Back"-Button, zeigt das QStackedWidget wieder die Startseite an.

Dialog-Widgets

Die in den Abbildungen 3.9, 3.10a, 3.10b, 3.11 und 3.12 gezeigten Dialoge wurden entsprechend den Vorgaben im Design implementiert. Jeder Dialog besteht aus einem Widget, das als Container für die Elemente des Dialoges dient. Diese Widgets werden, wie zuvor beschrieben, in die Seiten des QStackedWidget eingefügt. Die Implementierungen beschränken sich im wesentlichen auf die Erstellung der einzelnen Widgets und deren Elemente, die Zuordnung der Signale zu den Slots und, falls notwendig, die Zuweisung der Benutzereingaben zu den entsprechenden QSettings-Schlüsseln (siehe Tabelle 3.3.

4.3.3. ChatClient

Die im Design beschriebene Funktionalität des ChatClient wird durch die Klassen MfaMessage, MfaSocket und MfaChatClient implementiert.

MfaMessage

Eine MfaMessage ist ein assoziatives Feld, das eine beliebige Anzahl von Schlüssel-Wert-Paaren aufnehmen kann. Ein Schlüssel ist vom Typ String, ein Wert ist vom Typ QVariant. Die Qt-Klasse QVariant kann Variablen beliebigen Typs in sich kapseln. Jedes MfaMessage-Objekt besitzt einen Typ-Parameter. Derzeit existieren folgende Nachrichten-Typen:

- Config: Beinhaltet die Ports an die der MFA Video- und Audiodaten streamen soll. Wird beim Verbindungsaufbau vom Server zum MFA geschickt.
- Text: Enthält eine Textnachricht, die vom Server zum MFA gesandt wird.
- Stream Start: Der MFA sendet diese Nachricht zum Server, sobald der Video- und Audiodatenstrom gestartet wird.
- Stream Stop: Wird vom MFA an den Server geschickt, wenn der Video- und Audiodatenstrom gestoppt wird.
- GPS: Enthält vom MFA aufgezeichnete GPS-Koordinaten und eine Zeitstempel, die vom MFA an den Server gesendet werden.

MfaSocket

Die Implementierung der Klasse MfaChatClient baut auf der Klasse MfaSocket auf. Ein MfaSocket erweitert einen QTcpSocket um Methoden zum senden und empfangen von MfaMessage-Objekten.

`void connectToHost(HostName, Port)`: Diese Methode baut eine Verbindung zu einem Server auf. War der Verbindungsaufbau erfolgreich, emittiert der MfaSocket das Signal `connected()`.

`void sendMessage(MfaMessage)`: Durch Aufruf dieser Methode kann eine Nachricht zum Server übertragen werden.

`void newMessage(MfaMessage)`: Das Signal `newMessage(MfaMessage)` wird vom MfaSocket emittiert, sobald dieser eine neue Nachricht vom Server empfangen hat.

`void enableKeepAlive()`: Zur Überwachung der Verbindung kann der MfaSocket TCP Keep Alive nutzen. TCP Keep Alive wird vom QTcpSocket nicht unterstützt und muß von der Funktion durch einige Aufrufe der Linux-Systemcalls `setsockopt()` und `getsockopt()` aktiviert werden. Falls der MfaSocket einen Verbindungsabbruch erkennt, emittiert er das Signal `disconnected()`.

MfaChatClient

Die Klasse MfaChatClient ermöglicht es den anderen Teilen der Anwendung, Nachrichten zu einem Server zu senden und von diesem zu empfangen. Zur Kommunikation mit dem Server verwendet der MfaChatClient einen MfaSocket.

Die Implementierung der Klasse folgt den Vorgaben im Design. Die Klasse besteht aus den dort angeführten Methoden und Slots und emittiert die dort beschriebenen Signale.

Um den automatischen Wiederaufbau einer unterbrochenen Verbindung zu implementieren, verwendet der MfaChatClient einen Timer. Dieser wird gestartet, sobald der MfaSocket das Signal `disconnected()` empfängt. Bei einem Timeout des Timers, versucht der MfaChatClient die Verbindung zum Server erneut aufzubauen. Das Timeout ist frei konfigurierbar und beträgt derzeit 15 Sekunden. Der Timer wird solange neu gestartet, bis der Verbindungsaufbau erfolgreich war.

4.3.4. GpsClient

Der GpsClient ist in der Klasse MfaGpsClient implementiert. Zum einlesen der GPS-Daten wird die Location API der Qt Mobility Bibliothek verwendet. Die Implementierung hält sich an die im Design aufgestellten Vorgaben.

Die Kommunikation mit der Qt Mobility Bibliothek erfolgt über die Klasse QNmeaPositionInfoSource. Ein QNmeaPositionInfoSource kann NMEA-Daten eines GPS-Device einlesen. Die Datenquelle kann aber auch eine Datei sein, in der zuvor aufgezeichnete NMEA-Daten gespeichert sind. Dies ist vorallem während der Entwicklung der Anwendung sehr hilfreich.

Die eingelesenen Daten werden über das Signal `positionUpdated(QGeoPositionInfo)` an den MfaGpsClient übergeben, der die relevanten Informationen aus dem Parameter vom Typ `QGeoPositionInfo` extrahiert, und über das Signal `newPosition(Timestamp, Latitude, Longitude)` an die Anwendung weitergibt.

4.3.5. MfaApplication

Die Implementierung der MfaApplication befindet sich in der Klasse MfaApp. Diese ist von der Klasse QApplication abgeleitet. Sie instantiiert, wie im Design beschrieben, die einzelnen Objekte der Anwendung und verbindet deren Signale und Slots mit einander. Der MfaGpsClient wird angewiesen, eine Verbindung zum Server aufzubauen und das MfaStartWindow wird eingeblendet.

Die MfaApp-Klasse kümmert sich zudem um das ein- und ausblenden des Framebuffer fb1. Der Framebuffer wird eingeblendet, wenn der Benutzer den Touchscreen berührt. Um auf dieses Ereignis detektieren zu können, überschreibt die Klasse die Methode `QApplication::qwsEventFilter(QWSEvent* event)`. Diese Methode empfängt alle für das Fenstersystem relevanten Ereignisse. Falls der Parameter `event` ein Touchscreen-Event beschreibt und Framebuffer fb1 transparent ist, wird die Methode `FrameBuffer::fadeIn()` aufgerufen. Umgekehrt wird der Framebuffer durch den Aufruf der Methode `FrameBuffer::fadeOut()` ausgeblendet, sobald der Benutzer den Close-Button des GUI betätigt, oder wenn ein Timer abläuft. Der Timer wird in der Methode `qwsEventFilter()` bei jedem erkannten Touchscreen-Event zurück gesetzt und neu gestartet.

5. Resultate und Ausblick

In Abschnitt 3.1 im Kapitel Design wurden folgende Anforderungen identifiziert, die von der Anwendung erfüllt werden sollen:

- Aufzeichnung von Video- und Tondaten
- Wiedergabe von Video- und Tondaten am MFA
- Tonübertragung zwischen MFA und Empfänger
- Videoübertragung zwischen MFA und Empfänger
- Übertragung der GPS Positionsdaten
- Intuitive Bedienung des MFA

Es folgt nun eine Analyse, in wie weit und mit welchen Einschränkungen diese Anforderungen von der implementierten Anwendung erfüllt werden.

5.1. Aufzeichnung

Die Funktionen zur Aufzeichnung von Video- und Tondaten konnten erfolgreich in der Multimedia-Bibliothek implementiert werden. Abbildung 5.1 zeigt den MFA beim abspielen einer von sich selbst erstellten Aufnahme.

Durch die Verwendung der VPU, kann die Bibliothek Videos mit einer Auflösung von horizontal 16 bis 640 und vertikal 16 bis 480 Pixeln bei einer Bildwiederholrate von bis zu 30 fps erstellen und in das H.264 Baseline Profile kodieren. Dies ist trotz der beschränkten Hardwareressourcen des SoC in Echtzeit möglich. Während einer Videoaufnahme bewegt sich die Auslastung der ARM-CPU in einem Bereich um 5%, es stehen also noch genügend Ressourcen zur Bearbeitung anderer Aufgaben zur Verfügung. Die CPU-Auslastung wurde mit dem Programm „top" ermittelt.

Die Audiodaten werden mit einer Abtastrate von 16 kHz aufgezeichnet und in das MP2-Format codiert. Obwohl die Kodierung der Daten in Software durch die ARM-CPU erfolgt, entsteht nur eine geringe CPU-Aulastung von ca. 5%. Alternativ zum MP2-Codec wurde auch der μ-law-Codec getestet. Dieser liefert qualitativ deutlich schlechtere Ergebnisse, erzeugt aber eine CPU-Auslastung von nur 1%. Aufgrund der schlechten Audioqualität des μ-law Codec, wurde der MP2-Codec als Standardcodec gewählt.

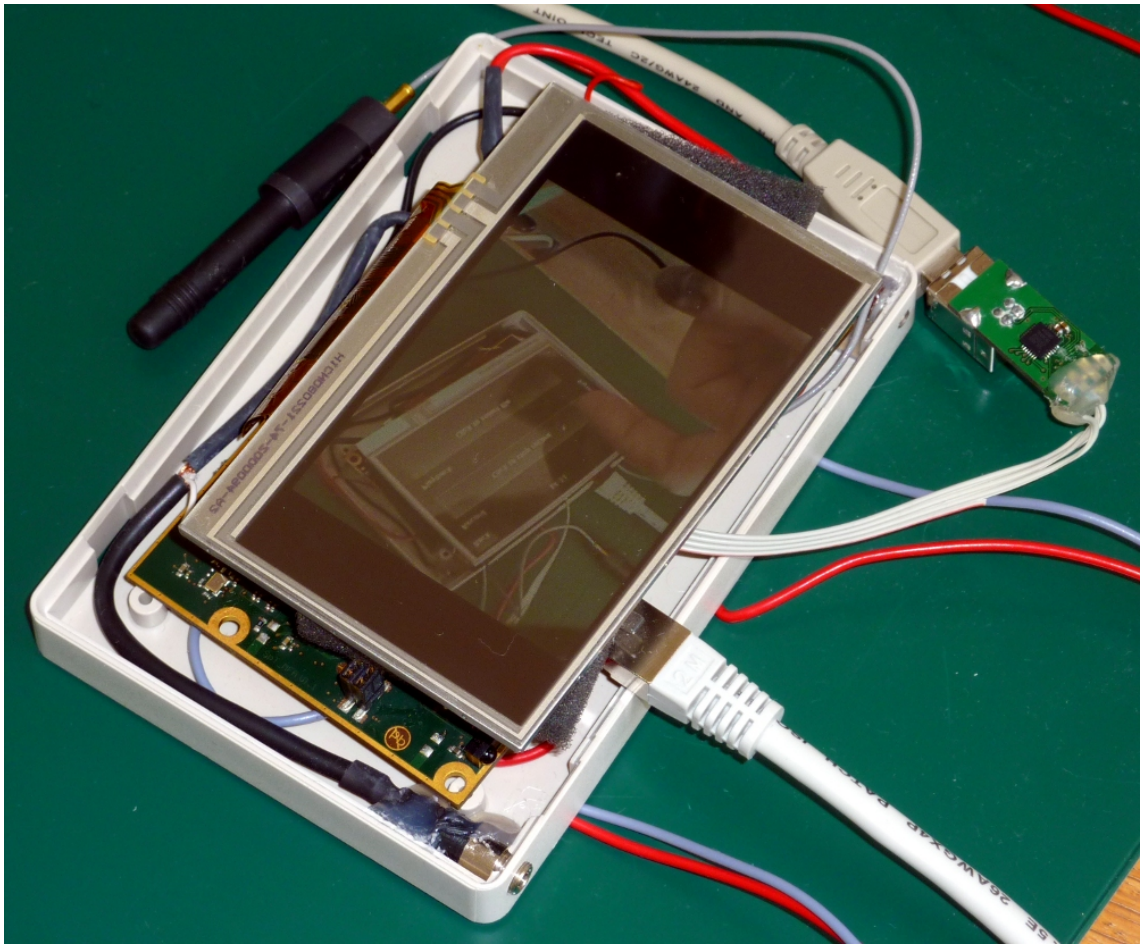


Abbildung 5.1.: Der MFA beim abspielen einer von der Anwendung erstellten Aufnahme von sich selbst.

Die Bibliothek kann entweder nur Videodaten, nur Tondaten, oder beide synchron aufzeichnen. Die Daten werden in einem Matroska-Container gespeichert. Über die Benutzeroberfläche kann der Anwender derzeit nur synchrone Video- und Tonaufzeichnungen starten. Es ist aber problemlos möglich, das GUI um eine Option zur Aufzeichnung von Audio ohne Video zu erweitern.

Um die bei der Aufzeichnung anfallende Datenmenge abschätzen zu können, wurde die Größe der nicht kodierten und der H.264-kodierten Videobilder ermittelt. Aufgezeichnet wurde das Kamerabild mit einer Auflösung von 640 mal 480 Pixeln und einer Bildwiederholrate von 25fps. Der H.264-Encoder wurde mit einer Bitrate von 1024 Mbit/s konfiguriert. In Tabelle 5.2 sind die Durchschnittswerte einer 75 s langen Videoaufnahme angegeben. Die Größe der nicht kodierten Videobilder ist konstant. Diese werden vom H.264-Encoder auf weniger als 1% ihrer ursprünglichen Größe komprimiert.

	Raw Video	H.264 Video
Dauer [s]	75	
øBildgröße [B]	460800	4263,56
øDatenrate [KB/s]	11516,21	106,65
Datenmenge [MB]	872	8

Tabelle 5.2.: Größe und Datenrate nicht kodierter und kodierter Videobilder einer 75 s langen Aufnahme.

5.2. Wiedergabe

Die Multimedia-Bibliothek kann Videodaten in den Formaten H.264 Baseline Profile und MPEG-4 Simple Profile wiedergeben. Die Auflösung der Videos kann bis zu 640 mal 480 Pixel betragen, die darstellbare Größe ist aber Aufgrund der Auflösung des Display auf 480 mal 272 Pixel beschränkt. Zu große Videobilder werden vor der Darstellung auf dem Display automatisch skaliert. Die Bildwiederholrate kann bis zu 30 fps betragen. Die ARM-CPU wird beim abspielen von Videos mit einer Auslastung von ca. 2% nur sehr gering belastet.

Die von der Bibliothek abspielbaren Audioformate sind MP3, MP2, a-law und μ-law. Alle Formate müssen von der ARM-CPU dekodiert werden. Bei MP3 führt dies zu einer CPU-Auslastung von bis zu 20%. Die Dekodierung der anderen Formate führt zu einer CPU-Auslastung von weniger als 3%.

Die Video- und Audiodaten müssen in einem Matroska-Container gespeichert sein. Es sollte aber problemlos möglich sein, die Bibliothek um die Unterstützung von AVI-Containern zu erweitern.

Über das GUI der Anwendung kann der Benutzer am MFA gespeicherte Video- oder Audiodateien auswählen, und deren Wiedergabe starten.

5.3. Ton- und Videoübertragung

Die Live-Übertragung der Video- und Tonaufnahmen über eine Netzwerkverbindung konnte in der Multimedia-Bibliothek erfolgreich implementiert werden. Die Übertragung kann über das LAN, WLAN oder eine UMTS-Verbindung erfolgen. Die kodierten Video- und Tondaten werden in RTP-Pakete verpackt und per UDP an ihr Ziel gesendet. Dank des Einsatzes standardisierter Formate und Protokolle kann der vom MFA generiert Videostrom von gängigen Videospielern dargestellt werden. Erfolgreich Getestet wurden der VLC Media Player [1] unter Windows und Linux, sowie der Totem Video Player [2] und der MPlayer

[1] `http://www.videolan.org/vlc/` (4.5.2011)
[2] `http://gnome.org/projects/totem/` (4.5.2011)

[3] unter Linux.

Die Latenz der Übertragung ist sehr gering, hängt aber vorallem vom Empfänger der Daten ab. Der VLC Media Player puffert die empfangenen Daten vor dem abspielen, was zu einer Verzögerung von ca. 3 Sekunden führt. Bei Verwendung des Totem Video Player zum Empfang der Daten bleibt die Latenz deutlich unter einer Sekunde.

Störungen treten vorwiegend aufgrund von Problemen bei der Netzwerkverbindung auf. In einem LAN ist die Qualität der Übertragung sehr gut. In Funknetzen, wie WLAN oder UMTS-Verbindungen, kann die zur Verfügung stehende Bandbreite stark schwanken, was sich sehr negativ auf die Übertragungsqualität auswirkt. Die Bibliothek soll daher um Mechanismen erweitert werden, die es ermöglichen, die vom Video- und Audioencoder verwendete Bitrate dynamisch an die vorhandene Bandbreite anzupassen. Da der Sender keine Information darüber erhält, mit welcher Latenz die UDP-Pakete an ihr Ziel gelangen oder ob und wieviele UDP-Pakete verloren gehen, sollte der Empfänger der Daten eine Rückmeldung über die Qualität des Datenstroms an den MFA senden. Anhand der Rückmeldungen könnte der MFA automatisch Einfluß auf Bitrate, Auflösung und Bildwiederholrate nehmen, und so die Größe des Datenstroms reduzieren oder gegebenenfalls auch wieder erhöhen. Die Rückmeldungen könnten vom Server durch eine Analyse des empfangenen Datenstrom automatisch erstellt werden. Denkbar wäre auch eine manuelle Bewertung der Übertragungsqualität durch den Anwender des Servers. Zur Übertragung der Rückmeldungen würde sich die Verwendung des RTCP anbieten [24].

Da zum Datentransport derzeit UDP verwendet wird, kann es zu Problemen kommen wenn sich der MFA oder das Ziel des Datenstroms in einem privaten Netzwerk hinter einem Router befinden. In [25] werden die Problematik und mögliche Lösungsansätze beschrieben. Durch die Verwendung des RTSP am MFA könnte die Problematik zumindest teilweise gelöst werde. Bei RTSP erfolgt der Verbindungsaufbau durch den Empfänger der Video- und Tondaten. Gängige Medienspieler unterstützen Techniken zum „NAT hole punching" [26] die es ermöglichen, auch Ziele, die sich in privaten Netzwerken hinter Routern befinden, zu erreichen. Das RTSP sieht zudem Möglichkeiten vor, die RTP-Pakete nicht per UDP, sondern über eine vom Empfänger initiierte TCP-Verbindung zu übertragen. Dies sollte die Router-Problematik weiter entschärfen.

Bei der Verwendung des UMTS-Modems zur Datenübertragung muß mit Störungen und einer sehr begrenzten Bandbreite gerechnet werden. Um die Qualität in Abhängigkeit von der Videobitrate abschätzen zu können, wurden der UDP-Datenstrom einer Videoübertragung aufgezeichnet und ausgewertet. Die UDP-Pakete wurden beim Empfänger der Daten mit Hilfe des Programms „Wireshark" [4] aufgezeichnet.

Die Messungen wurden jeweils für die Videobitraten 512 kbit/s, 1024 kbit/s und 2048 kbit/s durchgeführt, wobei jeweils Videobilder mit einer Auflösung von 640 mal 480 Pixeln bei einer Bildwiederholrate von 25 Bildern pro Sekunde versendet wurden. Die UMTS-Verbindung wurde ausgehend vom MFA mit einer SIM-Karte von A1 mit dem Datentarif

[3] `http://www.mplayerhq.hu/`
[4] `http://www.wireshark.org` (4.5.2011)

„B.FREE Breitband" [5] aufgebaut. Die Ergebnisse und Auswertungen der Messungen sind in Tabelle 5.4 zusammengefasst.

Verlorene Pakete sind jene Pakete, die beim Empfänger entweder gar nicht oder zu spät, um sie für die Videodekodierung nutzen zu können, ankommen. Die Anzahl der verlorenen Pakete ist für die Bitraten 512 kbit/s und 1024 kbit/s sehr gering. Das übertragene Video kann hier beim Empfänger ohne große Störungen wiedergegeben werden. Anders sieht die Situation bei einer Videobitrate von 2048 kbit/s aus. Hier sind die Störungen so stark, daß die Videodarstellung beim Empfänger sehr schlecht ist. Das Video kann nur mit langen Aussetzern und vielen Artefakten dargestellt werden. Grund für die Störungen ist die beschränkte Upload-Bandbreite. Offensichtlich ist die durchschnittliche Upload-Datenrate auf 1641 kbit/s begrenzt und ist nicht ausreichend für die Übertragung des mit 2048 kbit/s kodierten Videos.

	Videobitrate [kbit/s]		
	512	**1024**	**2048**
Pakete/s	54,52	91,88	152,10
Bytes/Paket	1058	1229	1349
Übertragungsrate [kbit/s]	462	903	1641
Verlorene Pakete [%]	2,09	3,43	13,03

Tabelle 5.4.: Parameter der Videoübertragung vom MFA per UMTS. Gemessen beim Empfänger der Daten.

Schließlich wurde auch noch der Einfluß des zur Übertragung verwendeten Netzwerkprotokolls untersucht. In Abbildung 5.2 sind die beim Empfänger mit Wireshark gemessenen Datenraten eines Videodatenstroms dargestellt. Der Datenstrom wurde vom MFA über eine LAN-Verbindung und über eine UMTS-Verbindung übertragen. Dabei wurden jeweils die Protokolle TCP und UDP parallel zur Übertragung der Daten verwendet, was die pro Videostrom zur Verfügung stehende Bandbreite halbierte. Dies wurde zweimal, für die Videobitraten 768 kbit/s und 1024 kbit/s, durchgeführt.

Für eine Videobitrate von 768 kbit/s sind die Unterschiede zwischen den Protokollen sehr gering. Die Übertragung erfolgt weitgehend fehlerlos und die Bilder können beim Empfänger ohne Störungen wiedergegeben werden. Bei einer Videobitrate von 1024 kbit/s kommt es zu Problemen bei der Übertragung per UMTS/TCP und UMTS/UDP. Bei UMTS/UDP kann das Video beim Empfänger nur mit großen Störungen und Aussetzern dargestellt werden. Bei UMTS/TCP werden die einzelnen Videobilder zwar weitgehend fehlerlos dargestellt, allerdings mit einer stetig größer werdenen Latenz. Nach einer Übertragungszeit von 60 s lag das UMTS/TCP-Video ca. 6 s zurück. Ursache für diese Probleme bei UMTS/TCP und UMTS/UDP dürfte eine zu geringe Bandbreite der UMTS-Verbindung sein. Dies führt

[5] `http://www.a1.net/privat/bfreebreitbandinfo` (4.5.2011)

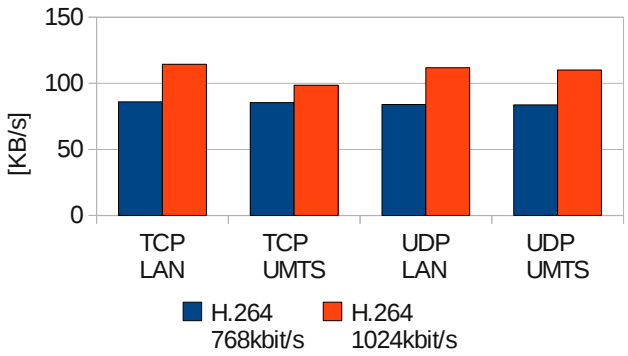


Abbildung 5.2.: Beim Empfänger gemessene Datenrate bei der Übertragung eines Videodatenstroms über UMTS und LAN, jeweils mit TCP und UDP.

bei UDP zu Störungen, da die vom Videodecoder benötigten Videopakete verloren gehen oder nicht rechtzeitig eintreffen. Bei TCP werden die einzelnen Videobilder erst dekodiert, wenn alle zu einem Bild gehörenden TCP-Pakete empfangen wurden. Konnte ein TCP-Paket nicht korrekt empfangen werden, wird es erneut angefordert. Dies führt dazu, daß die einzelnen Bilder zwar fehlerlos dargestellt werden, allerdings mit einer reduzierten Bildwiederholrate und einer großen Latenz. In [27] sind diese Effekte und Gegenmaßnahmen näher beschrieben.

5.4. Bedienung

Die Benutzeroberfläche des MFA ist auf das wesentlich reduziert. Sie ermöglicht dem Benutzer die Ausführung folgender Aktionen:

- Starten von Aufnahmen
- Konfiguration der Video- und Tonparameter
- Abspielen von bestehenden Aufnahmen
- Anzeige vom Server empfangener Nachrichten
- Anzeige der textuellen GPS-Informationen
- Konfiguration der Server-Adresse

Abbildung 5.3 zeigt den MFA während der Darstellung des Startbildschirms der grafischen Oberfläche. Diese lässt sich Aufgrund ihres modularen Aufbaus problemlos um zusätzliche Dialog erweitern.

Der Framebuffer des MFA bietet einige interessante Funktionen, die im Rahmen dieses Projekts noch nicht genutzt wurden. So kann der Framebuffer des MFA virtuell, über die physikalische Auflösung des Display hinaus, vergrößert werden. Über ein virtuelles Fenster, das beliebig verschoben werden kann, wird der Ausschnitt des virtuellen Framebuffers

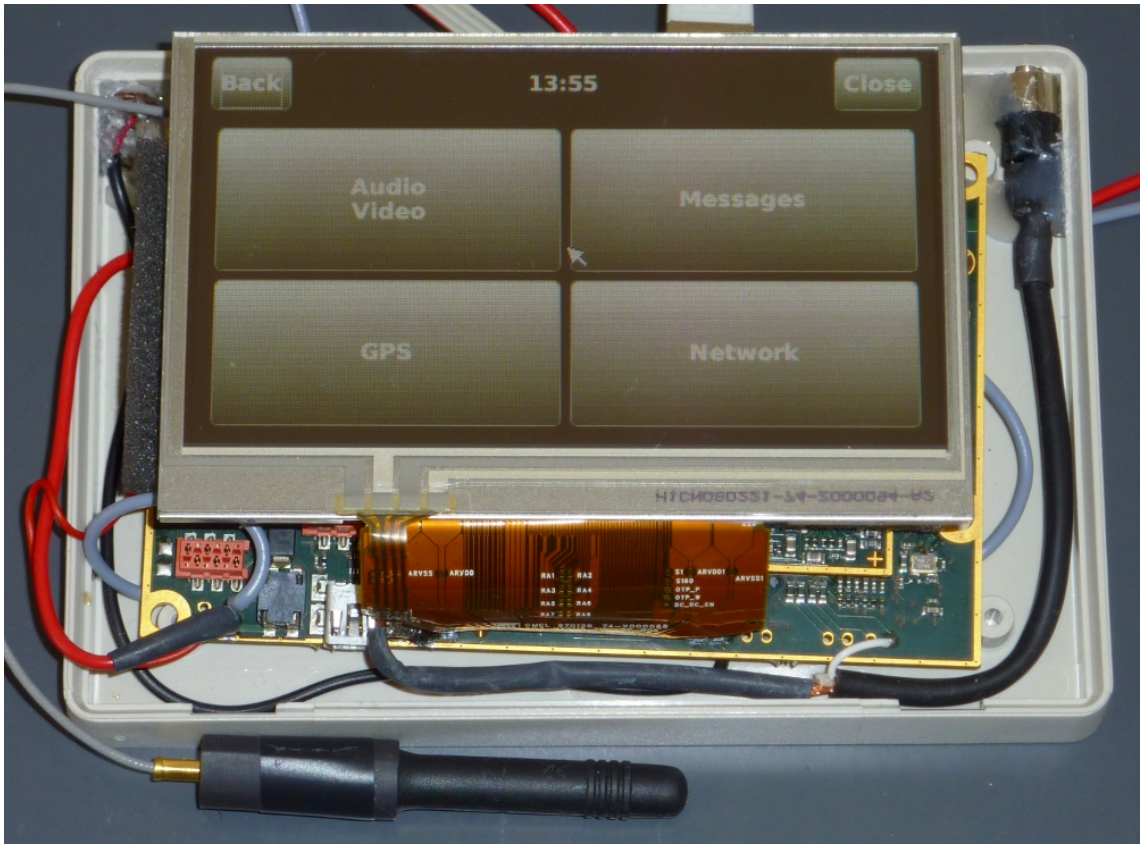


Abbildung 5.3.: Der Startbildschirm der grafischen Benutzeroberfläche.

bestimmt, der auf dem Display angezeigt wird. Die Positionierung des Fensters erfolgt sehr schnell und ohne Belastung der ARM-CPU. In einer erweiterten Version der grafischen Oberfläche könnte diese Funktion zur flüssigen Darstellung von Animationen oder Scroll-Effekten genutzt werden.

5.5. Stromaufnahme

Ein wichtiger Aspekt bei der Implementierung der Anwendung ist die Stromaufnahme des MFA. Der MFA ist ein mobiles Gerät, das über einen Akku mit Energie versorgt wird. Um die Laufzeit des MFA abschätzen zu können, wurde die Stromaufnahme für verschiedene Betriebszustände gemessen. Bei den Messungen wurden jeweils vier Fälle Unterschieden: Betrieb des MFA mit einer LAN-Verbindung ohne WLAN- und ohne UMTS-Modul (LAN), nur mit WLAN-Modul (WLAN), nur mit UMTS-Modul (UMTS), sowie Betrieb mit WLAN- und UMTS-Modul (WLAN+UMTS). Alle Messungen wurden bei einer Versorgungsspannung von 10 V durchgeführt. Die Messungen wurden mit einem Voltcraft VLP 1303pro Labornetzgerät durchgeführt.

5.5.1. Display

Um den Einfluß des Display auf die Stromaufnahme beurteilen zu können, wurde eine Messung direkt nach abgeschlossenem Bootvorgang, ohne Initialisierung der VPU und WLAN/UMTS-Module, durchgeführt. Die Stromaufnahme wurde bei ausgeschaltetem Display, Display mit schwarzem Inhalt und Display mit weißem Inhalt durchgeführt.

In Tabelle 5.6 und in Abbildung 5.4 sind die Messergebnisse zusammengefasst. Es zeigt sich, daß das AMOLED-Display, wie erwartet, für weiße Inhalte den höchsten Stromverbrauch aufweist. Bei der Gestaltung der grafischen Oberfläche sollte dies berücksichtigt werden.

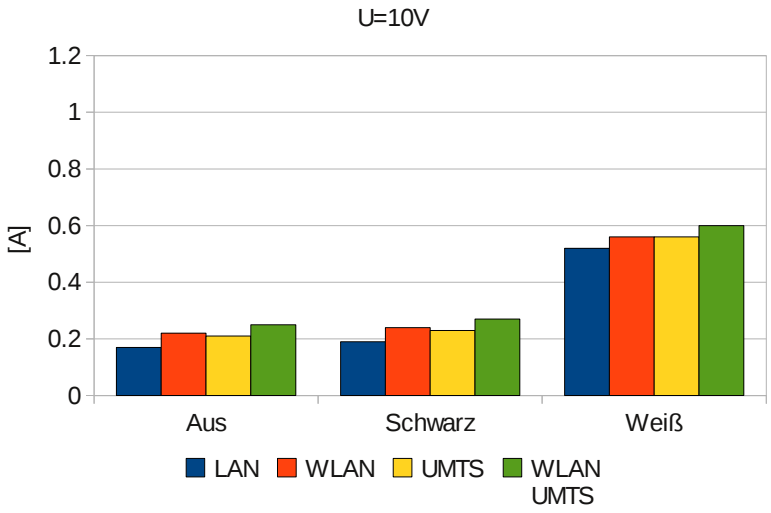


Abbildung 5.4.: Stromaufnahme des MFA bei ausgeschaltetem und aktivem Display mit schwarzem und weißem Inhalt.

5.5.2. Videoencoder und Kamera

Neben dem Display tragen auch der Videoencoder, die VPU und die Kamera einen Teil zum Stromverbrauch bei. Die Kamera verfügt über eine eigene Spannungsversorgung, die durch die Software ein- und ausgeschaltet werden kann. Wieviel Strom eine Kamera benötigt, ist vom Kameramodell abhängig. Gemessen wurden der Stromverbrauch des MFA jeweils nach Initialisierung der Videohardware, einmal bei deaktivierter und einmal bei aktivierter Spannungsversorgung der Kamera.

In Tabelle 5.6 und in Abbildung 5.5 sind die Ergebnisse zusammengefasst. Es zeigt sich, daß die derzeit verwendete Kamera einen Strom von etwa 0,15 A bei einer Versorgungsspannung von 3 V benötigt.

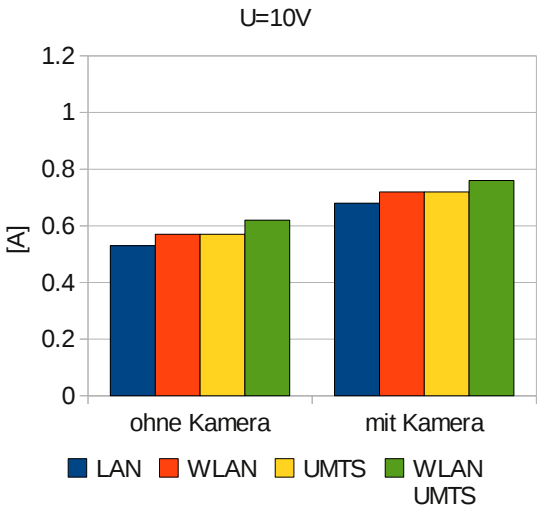


Abbildung 5.5.: Stromaufnahme des MFA nach Initialisierung der Videohardware, gemessen mit aktivierter und deaktivierter Kamera.

5.5.3. Datenübertragung

Die Stromaufnahme des MFA ist während der Übertragung von Ton- und Videodaten stark von dem zur Übertragung verwendeten Netzwerkdevice abhängig. Es wurden zwei Messreihen aufgenommen. Einmal direkt nach Initialisierung der Hardwaremodule aber ohne aktive Übertragung. Einmal während der Übertragung eines Videostroms vom MFA zu einem Empfänger. Der Videostrom hat eine Auflösung von horizontal 640 und vertikal 480 Pixeln, bei einer Bildwiederholrate von 30 Bildern pro Sekunde, die von der VPU mit einer Bitrate von 512 kbit/s kodiert werden. Die Messung der Stromaufnahme wurde bei Verwendung des LAN, des WLAN, des UMTS und bei gleichzeitiger Übertragung der Daten über WLAN und UMTS gemessen.

In Tabelle 5.6 und in Abbildung 5.6 sind die Ergebnisse dargestellt. Wie erwartet ist die Stromaufnahme bei gleichzeitiger Übertragung per WLAN und UMTS maximal. Separat betrachtet weist, wenig überraschend, das UMTS-Modul den höchsten Stromverbrauch auf. Auffallend ist, daß die Stromaufnahme des WLAN-Moduls während der Übertragung nicht sehr stark ansteigt. Ursache dafür könnte ein nicht optimal implementierter Treiber sein, der die Energiesparfunktionen des WLAN-Moduls nicht optimal ausnutzt.

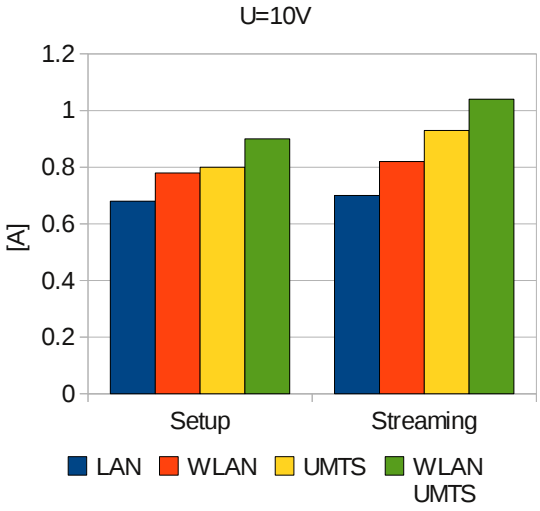


Abbildung 5.6.: Stromaufnahme des MFA, links nach Initialisierung der LAN/WLAN/UMTS-Schnittstelle, rechts während der Übertragung eines Videostreams.

5.5.4. Aufteilung des Stromverbrauchs

In Abbildung 5.7 ist die Verteilung der Stromaufnahme auf die einzelnen Komponenten des MFA während der Übertragung eines Videostroms über das UMTS-Modul dargestellt. Der Sektor „System" des Diagramms beschriebt den Stromverbrauch des Basissystems, also vorallem von CPU und RAM. Der Anteil des aktiven Bildschirms mit weißem Inhalt ist in Sektor „Display" angegeben, der der aktiven Kamera in Sektor „Kamera". Im passiven Grundzustand benötigt das UMTS-Modul etwa 13% des Gesamtstroms (Sektor „UMTS"). Während der Datenübertragung erhöht sich dieser Wert um 14%-Punkte (Sektor „Übertragung"). In Summe ergibt dies einen Anteil von 27% am gesamten Stromverbrauch. Der größte Verbraucher ist das Display, allerdings nur bei rein weißem Hintergrund. Wird das Display deaktiviert oder nur ein dunkler Inhalt dargestellt, wird das UMTS-Modul zum größten Stromverbraucher. Es zeigt sich, daß die Stromaufnahme relativ einfach durch einen optimierten Einsatz des Displays gesenkt werden kann.

Geplant ist, die Stromaufnahme des MFA weiter zu reduzieren. Die Treiber der verschiedenen Hardwaremodule sollen zur konsequente Nutzung der Stromsparmodi optimiert werden. Zudem sollen Probleme mit dem Suspend-To-RAM-Modus behoben werden, um diesen Einsatzfähig zu machen.

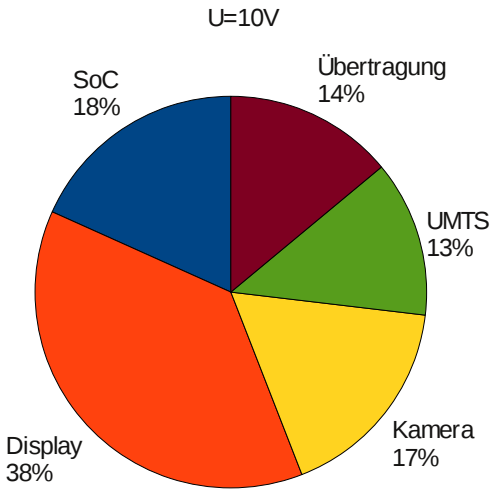


Abbildung 5.7.: Verteilung der Stromaufnahme des MFA auf die einzelnen Komponenten während einer Videoübertragung per UMTS.

Display	LAN [A]	WLAN [A]	UMTS [A]	WLAN+UMTS [A]
Aus	0.17	0.22	0.21	0.25
Schwarz	0.19	0.24	0.23	0.27
Weiß	0.52	0.56	0.56	0.6
Kamera (Display: Weiß)				
Deaktiviert	0.53	0.57	0.57	0.62
Aktiviert	0.68	0.72	0.72	0.76
Übertragung (Display: Weiß, Kamera: Aktiviert)				
Setup	0.68	0.78	0.8	0.9
Stream	0.7	0.82	0.93	1.04

Tabelle 5.6.: Stromaufnahme des MFA in verschiedenen Betriebszuständen bei einer Versorgungsspannung von 10 V.

6. Schlußbemerkung

Abschließend möchte ich bemerken, daß die Entwicklung von Software für ein Embedded System eine spannende Aufgabe ist. Im Gegensatz zur Entwicklung von Desktop-Anwendung, ist die Ziel-Hardware hier fix vorgegeben. Einige, sonst als variable anzusehende Systemparameter, wie zum Beispiel die Geschwindigkeit der CPU oder die Größe des Arbeitspeichers, können als konstante Größen angenommen werden. Einerseits führt dies zu einer Reduzierung der Komplexität des Software-Entwurfprozesses. Andererseits sind Ressourcen wie CPU-Geschwindigkeit und Speichergröße, im Vergleich zu einem aktuellen Desktop-Rechner, stark eingeschränkt, wodurch die gewonnene Reduzierung an Komplexität wieder aufgewogen wird. Nicht alles, was mit einem aktuellen Desktop-PC möglich ist, ist auch mit einem Embedded System möglich. Aber dennoch erstaunlich viel. Gefragt sind einfache, aber effiziente Lösungen.

Als größte Hürde bei der Entwicklung der Anwendung erwies sich die Programmierung der GStreamer-Bibliothek. Die Erstellung einer gültigen Pipeline ist eine Kunst für sich. Gepaart mit der eigenwilligen glib-API, deren Programmierung bestenfalls als „umständlich" beschrieben werden kann, resultiert daraus mancher Arbeitstag, der einzig und allein der Fehlersuche gewidmet ist. Dennoch scheint der Aufwand, im Vergleich zu einer direkten Programmierung der Hardware, gerechtfertigt.

Die Programmierung mit Qt ist hingegen eine wahre Freude. Die API ist hervorragend dokumentiert und unterstützt den Programmierer durch eine große Fülle an verfügbaren Klassen und Funktionen. Die Entwicklungszeit einer Anwendung verkürzt sich durch die Verwendung von Qt erheblich.

Im Großen und Ganzen war die Planung und Durchführung dieses Projekts eine interessante und lehrreiche Erfahrung, die ich nicht missen möchte.

A. Anhang

A.1. Blockdiagramm des i.MX27

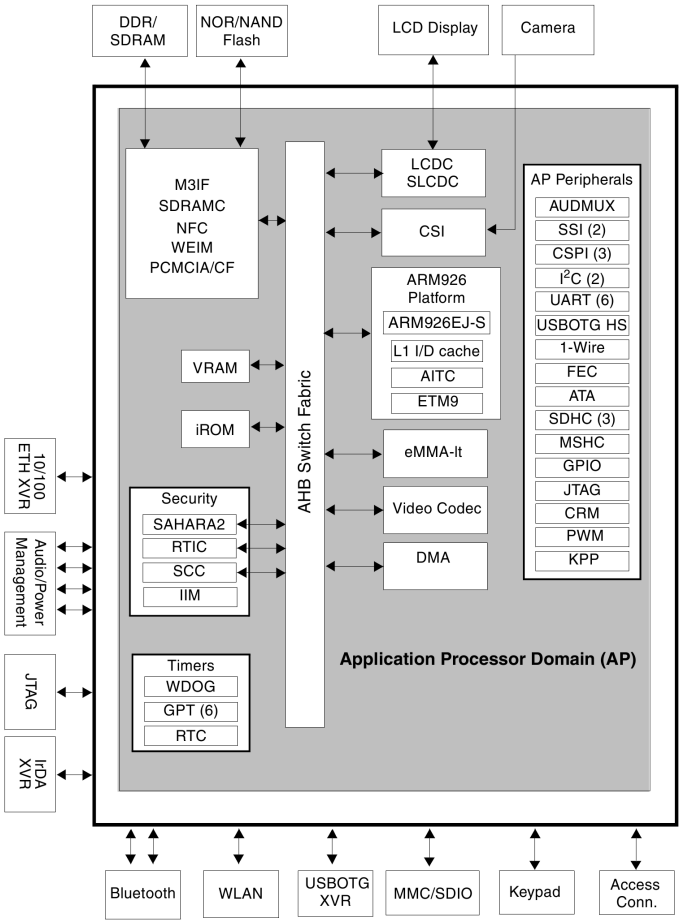


Abbildung A.1.: Blockdiagramm des i.MX27 [i.MX27 Datasheet] S.3

Literatur

[1] S. Bilavarn u. a. „Embedded Multicore Implementation of a H.264 Decoder with Power Management Considerations“. In: *Digital System Design Architectures, Methods and Tools, 2008. DSD '08. 11th EUROMICRO Conference on*. 2008, S. 124 –130.

[2] A.U. Batur, B.E. Flinchbaugh und III Hayes M.H. „A DSP-based approach for the implementation of face recognition algorithms“. In: *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*. Bd. 2. 2003, II –253–6 vol.2.

[3] International Telecommunication Union. *ITU-R Recommendations BT Series*. online. `http://www.itu.int/rec/R-REC-BT/en` (4.5.2011).

[4] Bin Fan, Lianfeng Shen und Tiecheng Song. „The Design and Implementation of a Wireless Real-Time Video Transmission System over WLAN“. In: *Information Science and Engineering (ICISE), 2009 1st International Conference on*. 2009, S. 684 –687.

[5] Jin Zhou und Xien Ye. „Design and implementation of embedded video terminal based on Z228“. In: *Audio, Language and Image Processing, 2008. ICALIP 2008. International Conference on*. 2008, S. 829 –833.

[6] M. Kuwahara und K. Yoneyama. „A portable camcorder/server for wireless video transmission“. In: *Consumer Electronics, IEEE Transactions on* 51.2 (2005), S. 351 –356.

[7] V. Soni und R. Mendiratta. „Next-generation wlan architecture for high performance networks“. In: *Wireless, Mobile and Multimedia Networks, 2008. IET International Conference on*. 2008, S. 125 –129.

[8] R.M. Schreier und A. Rothermel. „A Latency Analysis on H.264 Video Transmission Systems“. In: *Consumer Electronics, 2008. ICCE 2008. Digest of Technical Papers. International Conference on*. 2008, S. 1 –2.

[9] MPEG-4 Industry Forum. *MPEG-4 Simple and Advanced Simple Profiles*. online. `http://www.mpegif.org/public/documents/vault/m4-out-30037.pdf` (4.5.2011).

[10] N. Vun und M. Ansary. „Implementation of an embedded H.264 live video streaming system“. In: *Consumer Electronics (ISCE), 2010 IEEE 14th International Symposium on*. 2010, S. 1 –4.

[11] *TMS320DM6446 DVEVM v2.0 Getting Started Guid*. SPRUE66E. `http://www.ti.com/litv/pdf/sprue66e` (4.5.2011). Texas Instruments. 2008.

[12] H. Schulzrinne, A. Rao und R. Lanphier. *Real Time Streaming Protocol (RTSP).* RFC 2326 (Proposed Standard). `http://www.ietf.org/rfc/rfc2326.txt` (4.5.2011). Internet Engineering Task Force, 1998.

[13] H. Schulzrinne u. a. *RTP: A Transport Protocol for Real-Time Applications.* RFC 3550 (Standard). Updated by RFCs 5506, 5761, 6051. `http://www.ietf.org/rfc/rfc3550.txt` (4.5.2011). Internet Engineering Task Force, 2003.

[14] M. Handley, V. Jacobson und C. Perkins. *SDP: Session Description Protocol.* RFC 4566 (Proposed Standard). `http://www.ietf.org/rfc/rfc4566.txt` (4.5.2011). Internet Engineering Task Force, 2006.

[15] Jun-Wei Gao und Ke-Bin Jia. „Embedded Video Surveillance System Based on H.264". In: *Multimedia Information Networking and Security, 2009. MINES '09. International Conference on.* Bd. 1. 2009, S. 282 –286.

[16] Zu jue Chen, Zhi xiong Zhang und Jian jiang Zhang. „Design and implementation of video player system based on embedded system and Qt/E". In: *Visual Information Engineering, 2008. VIE 2008. 5th International Conference on.* 2008, S. 468 –472.

[17] Pixavi. *Xcore ST6000.* `http://www.pixavi.com/imgs/hardware/Xcore-ST-6000-hand.jpg` (4.5.2011).

[18] TS-Market. *m.AVR H.264.* `http://www.ts-market.com/upload/iblock/fb8/26441.jpg` (4.5.2011).

[19] LiveU. *LU-30.* `http://liveu.tv/images/LU-30_00.jpg` (4.5.2011).

[20] E. Gamma u. a. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Professional, 1995. ISBN: 978-0-201-63361-0.

[21] International Telecommunication Union. *Pulse Code Modulation (PCM) of Voice Frequencies, ITU-T Recommendation G.711.* `http://www.itu.int/rec/T-REC-G.711-198811-I/en` (4.5.2011). 1993.

[22] H. Hans u. a. *OpenEmbedded User Manual.* `http://docs.openembedded.org/usermanual/usermanual.pdf` (4.5.2010). 2009.

[23] W. Taymans u. a. *GStreamer Application Development Manual (0.10.31.2).* `http://www.gstreamer.net/data/doc/gstreamer/head/manual/manual.pdf` (5.4.2011). 2011.

[24] N. Baldo u. a. „RTCP feedback based transmission rate control for 3G wireless multimedia streaming". In: *Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004. 15th IEEE International Symposium on.* Bd. 3. 2004, 1817 –1821 Vol.3.

[25] B. Ford, P. Srisuresh und D. Kegel. „Peer-to-peer communication across network address translators". In: *Proceedings of the annual conference on USENIX Annual Technical Conference.* ATEC '05. `http://www.brynosaurus.com/pub/net/p2pnat.pdf` (4.5.2011). Anaheim, CA: USENIX Association, 2005, S. 13–13.

[26] Zhang Yamei und Cai Pengfei. „Research on using UDP to traverse NAT under P2P network environment". In: *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on.* Bd. 3. 2010, S. V3–32 –V3–35.

[27] Chi-Fai Wong u. a. „TCP streaming for low-delay wireless video“. In: *Quality of Service in Heterogeneous Wired/Wireless Networks, 2005. Second International Conference on*. 2005, 6 pp. –41.