

Diplomarbeit

Bidirektionale Videosprechanlage

Erweiterung einer Raspberry Pi basierten Videogegensprechanlage

Höhere Technische Bundeslehr- und Versuchsanstalt Anichstraße

Abteilung Elektrotechnik

Ausgeführt im Schuljahr 2019/20 von:

Andreas Grain 5AHET
Matthias Mair 5AHET

Betreuer:

DI(FH) Mario Prantl

Innsbruck, am 2020-03-25

Abgabevermerk:

Betreuer:

Datum:

Andreas Grain / Matthias Mair

Erklärungen

Eidesstattliche Erklärung

Wir erklären an Eides statt, dass wir die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht haben.

Ort, Datum

Andreas Grain

Ort, Datum

Mathias Mair

Gender-Erklärung

Aus Gründen der besseren Lesbarkeit wird in dieser Diplomarbeit die Sprachform des generischen Maskulinums angewendet. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig zu verstehen ist.

Andreas Grain / Matthias Mair

Inhaltsverzeichnis

Erklärungen	i
Eidesstattliche Erklärung	i
Gender-Erklärung	i
Inhaltsverzeichnis	ii
Kurzfassung/Abstract	vii
Deutsch	vii
English	viii
Projektergebnis	ix
Software - Andreas Grain	ix
Hardware - Matthias Mair	x
1 Einleitung	1
1.1 Funktionalität	1
1.2 Aufgabenteilung	2
1.3 Hinweise und Definitionen	3
I Software - Andreas Grain	5
2 Rahmenwerk	7
2.1 Anforderungen	7
2.2 Vergleich	7
2.2.1 Microsoft Xamarin	7
2.2.2 Webtechnologiebasierte Rahmenwerke	8
2.3 Build-Umgebung	9
2.3.1 Visual Studio 2019	9
2.3.2 Android Software Development Kit (SDK)	11
2.3.3 Build-Vorgang	11
3 Verwendete Software-Module	15
3.1 Real Time Streaming Protocol	15
3.1.1 RTSP Version 2	15
3.1.2 Anfragen-Aufbau	15
3.2 LibVLC Sharp	17
3.2.1 Erwähnenswerte Funktionen	17
3.2.2 Nachteile	19
3.2.3 Alternativen	19
3.3 GStreamer	19
3.3.1 Modularität	19

3.3.2 Version	20
3.3.3 Alternativen	20
3.3.4 Vorteile	21
3.3.5 Funktionsweise	21
3.3.6 Zusatzmodul	21
3.4 Live555 Proxy	22
4 Push-Benachrichtigung	25
4.1 Push Notification Service	25
4.1.1 Funktionsweise	26
4.2 AppCenter Push	26
5 Programm-Dokumentation	27
5.1 Übersicht	27
5.2 Portable Project	28
5.2.1 App.xaml.cs	28
5.2.2 MainPage.xaml	29
5.2.3 MainPage.xaml.cs	32
5.2.4 MediaClasses.cs	38
5.3 PiBell.Android	42
5.3.1 MainActivity.cs	42
6 Software-Ausblick	45
6.1 Gesicherte Verbindung	45
6.2 Weitere mobile Plattformen	45
II Hardware - Matthias Mair	47
7 Ausgangssituation	49
7.1 Hardware-Konfiguration	49
7.2 Problemdefinition	49
8 Technische Grundlagen	51
8.1 Power over Ethernet	51
8.2 Spannungsregler	52
8.3 Surface Mounting Technology	53
8.3.1 Entwicklung	53
8.3.2 Bauteilcodes	54
8.3.3 Mögliche Fertigungsfehler	57
8.4 Serielle Schnittstellen	59
8.4.1 Universal Asynchronous Receiver/Transmitter	59
8.4.2 Serial Peripheral Interface	59
8.5 Audio-Verstärker Class D	61
9 Berechnung	63
9.1 Spannungsabfall	63

10 Verbesserte Hardwarelösung	67
10.1 Konzept	67
10.2 Spannungsversorgung	68
10.2.1 Anforderungen	68
10.2.2 Auswahl Spannungsregler	68
10.2.3 Funktion des Spannungsreglers	69
10.2.4 Zusatzbeschaltung des Spannungsreglers	69
10.3 Lautsprecher-Verstärker	70
10.3.1 Anforderungen	70
10.3.2 Auswahl	71
10.3.3 Funktionen	71
10.3.4 Zusatzbeschaltung	72
10.3.5 Kühlung	72
10.4 Mikrocontroller	72
10.4.1 Atmel AVR	73
10.4.2 Architektur	73
10.4.3 Funktion	74
10.4.4 Zusatzbeschaltung	74
11 Hardware-Programmierung	77
11.1 Programm-Entwicklung	77
11.1.1 Mikrocontroller	77
11.1.2 Raspberry Pi	83
12 Platinen-Entwicklung	87
12.1 EAGLE	87
12.1.1 Versionsvergleich	87
12.1.2 Entwicklung mit EAGLE	88
13 Hardware-Ausblick	89
13.1 Gehäuse für Platine	89
13.2 Platine Version 2	89
III Appendix	91
A Datenblätter	93
A.1 Mikrocontroller Atmel ATmega328P	93
A.2 Audio-Verstärker PAM8403	98
A.3 Spannungsregler LMR33630CDDAR	103
B Fertigungsdokumentation	109
B.1 Code-Listing	109
B.2 Schaltpläne	124
B.3 Platinen-Design	125
B.4 Stückliste	125

Literatur und Quellen	127
Abkürzungen	131
Abbildungsverzeichnis	133
Tabellenverzeichnis	135

Kurzfassung/Abstract

Deutsch

Die vorliegende Diplomarbeit beschäftigt sich mit der Erweiterung einer Raspberry Pi (RPi)-basierten Videosprechanlage um eine Smartphone-Applikation, sowie der grundlegenden Überarbeitung der Stations-Hardware. Zusätzlich soll ein Linux-basierter, aus dem Internet erreichbarer Server zum Verteilen der Video-Streams eingerichtet werden.

Dieses Projekt basiert auf einer früheren Diplomarbeit von Sebastian Wagner und Tobias Pfluger an der HTL Anichstraße aus dem Jahr 2017/18, welche ein voll funktionsfähiges Videosprechanlagensystem hervorbrachte. Die verwendete Hardware bietet Verbesserungspotential und wird im Zuge dieser Diplomarbeit weiterentwickelt.

Die Hardware-Erweiterung besteht grundsätzlich aus zwei Teilen: zum einen werden die vielen Elektronik-Bausteine in einer zentralen Platine vereint, was eine einfachere und kostengünstigere Fertigung erlaubt. Zusätzlich wird die aktuelle Hardware der Station um einen Watchdog-Timer erweitert, welcher im Fehlerfall die Anlage zurücksetzt.

Softwareseitig wird die Anlage um eine Smartphone-App ausgebaut, mit welcher der Fernzugriff auf das System ermöglicht werden soll.

English

This diploma thesis deals with the extension of a RPi-based video intercom system by a smartphone application, as well as the fundamental revision of the station hardware. In addition, a Linux-based server, accessible from the Internet, will be set up to distribute the video streams.

This project is based on an earlier diploma thesis by Sebastian Wagner and Tobias Pfluger at the HTL Anichstraße from the year 2017/18, which produced a fully functional video intercom system. The hardware used offers potential for improvement and will be further developed in the course of this diploma thesis.

The hardware extension basically consists of two parts: on the one hand, the many electronic components are combined in a central circuit board, which allows easier and more cost-effective production. In addition, the current hardware of the station is extended by a watchdog timer, which resets the system in case of an error.

On the software side, the system will be extended by a smartphone app, which will enable remote access to the system.

Projektergebnis

Software - Andreas Grain

Im Zuge dieser Diplomarbeit wurden folgende Software-Teile programmiert:

- Oberfläche der Applikation (PiBell/MainPage.xaml)
- Klassen zur Verwaltung der Dienste (PiBell/Classes/MediaClasses.cs)
- Berechtigungsmanagement (PiBell.Android/MainActivity.cs)
- Audio-Aufnahme-Service in Android (PiBell.Android/AudioRecordingService.cs)
- Audio-Aufnahme-Service in iOS (PiBell.iOS/AudioRecordingService.cs)

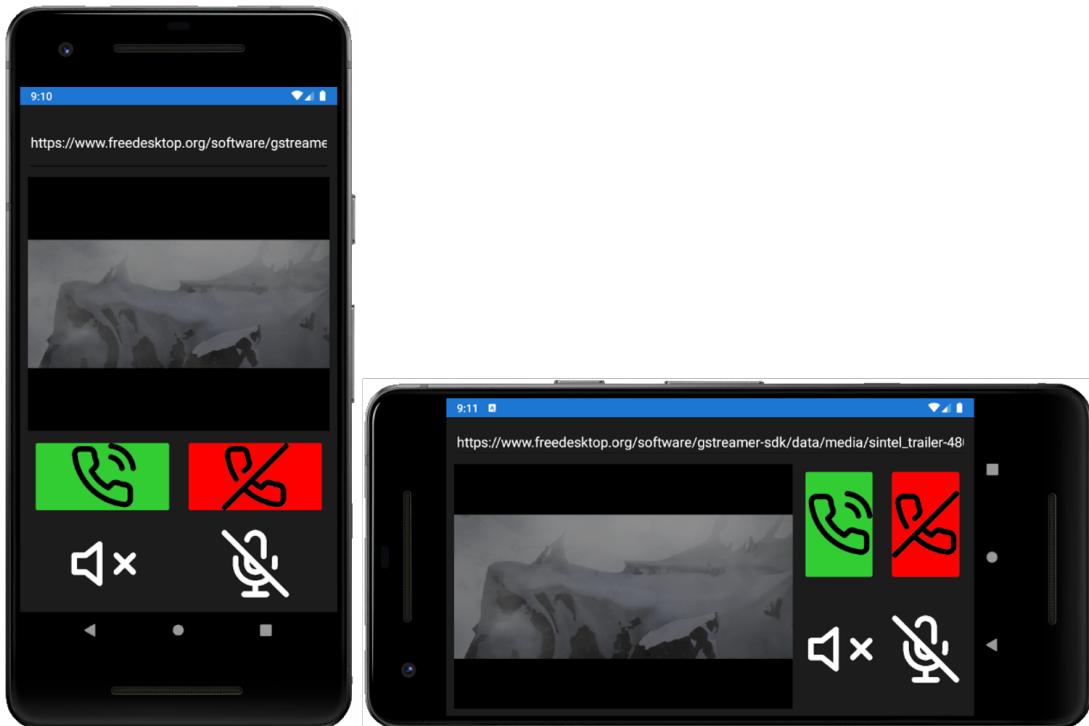


Abbildung 1: App-Ansichten

Aus diesen Software-Teilen wurde eine fertige Android-Applikation erstellt, mit der der Fernzugriff auf die fest installierte Videosprechanlage jederzeit möglich ist. Die Kommunikation wurde bidirektional für die Ton- und unidirektional für die Video-Übertragung

realisiert. Die Applikation wurde auf verschiedenen Systemen simuliert und in geringem Ausmaß getestet. Eine iOS-Version wurde aufgrund fehlender Endgeräte nicht fertiggestellt bzw. getestet.

Die programmierte Applikation ist in diesem Dokument ausführlich beschrieben und wesentliche Funktionen im Detail erklärt. Die Erläuterung des Programm-Codes ist kein normrelevantes Regelwerk, sondern ein Kommentar wie es der Autor Andreas Grain versteht! Im Zweifelsfall wird darauf hingewiesen, dass die offiziellen Angaben der Hersteller zu verwenden sind. Auf diese wird an relevanten Stellen verwiesen.

Verwendete Technologien und Software:

- Real Time Streaming Protocol (RTSP) zur Übertragung der Mediendaten über das Netzwerk
- LibVLC Sharp zum Abrufen und Darstellen des Video-Streams
- GStreamer zur Weiterverarbeitung der Mediendaten am Server
- Live555 Proxy zum Verteilen des RTSP-Streams
- Push Benachrichtigung zur Kontaktaufnahme mit dem Benutzer der App
- Visual Studio 2019 als Entwicklungsumgebung
- Xamarin-Rahmenwerk für die Entwicklung der Multiplattform-App
- ~~X~~LATEX zur Erstellung der Dokumentation

Hardware - Matthias Mair

Im Zuge der Diplomarbeit wurde eine Einzel-Surface-mounted Device (SMD)-Platine entwickelt, welche den internen Aufbau der Sprechanlage-Stationen vereinfacht. Die neu entwickelte Platine ersetzt folgende Bausteine der bisherigen Stations-Hardware:

- Spannungsregler zur Versorgung der Komponenten
- Lautsprecherverstärker zur Verstärkung des Lautsprechersignals
- Mikrofonverstärker zur Vorverstärkung des Mikrofon-Tons

Zusätzlich zu den alten Funktionen beinhaltet die neue Platine einen Watchdog-Timer, welcher die Betriebsstabilität der Station überwacht.

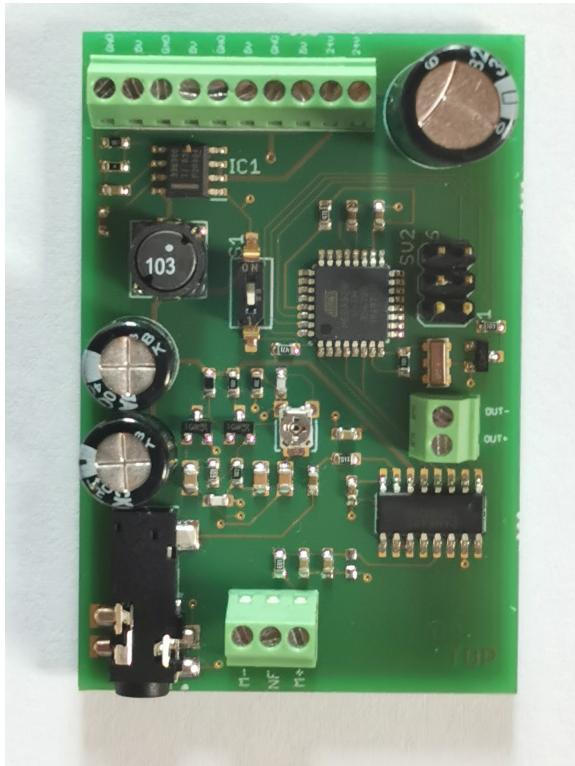


Abbildung 2: Bestückte Platine Top

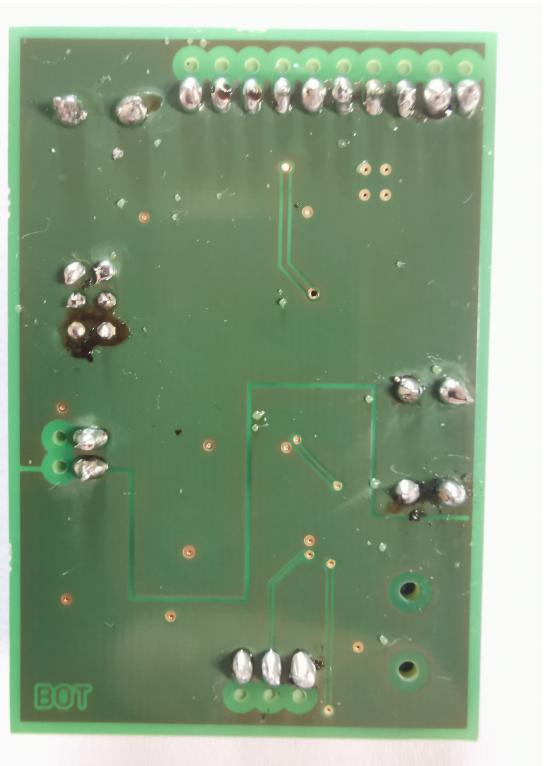


Abbildung 3: Bestückte Platine Bottom

Die Bestellung der Platine und der Großteil der Bestückung dieser erfolgte über DI(FH) Mario Prantls Firma Mario Prantl Solutions. Die Vervollständigung der Platine durch Einlöten der Through-Hole Technology (THT)-Komponenten erfolgte per Hand durch Matthias Mair.

Verwendete Technologien und Software:

- Einfach Anzuwendender Grafischer Layout-Editor (EAGLE) zur Erstellung des Schaltplans und Planung des Platinen-Layouts
- Arduino Integrated Development Environment (IDE) zur Programmierung des Mikrocontrollers

Im Zuge der Diplomarbeit wurde eine zweite Version der Platine entwickelt. Diese verfügt über zusätzliche Anschlusspunkte, welche mit digitalen Input/Outputs (IOs) des Mikrocontrollers verbunden sind. Die Platine ist vollständig geplant, konnte allerdings aufgrund der COVID-19-Krise in Tirol nicht im festgelegten Zeitrahmen gefertigt werden.

1 Einleitung

Jedes moderne Wohngebäude ist inzwischen mit einer Gegensprechanlage ausgestattet. Eine solche Anlage erlaubt es dem Wohnungsbesitzer mit jemandem vor der Haustür zu reden, bevor dieser hereingelassen wird. Manche Wohnungen besitzen bereits eine Videosprechanlage, die dem Bewohner nicht nur den Ton, sondern zusätzlich ein Video von der Außenstelle bereitstellt. Dem Gast wird jedoch immer noch nur der Ton von innen übertragen. Diese Applikation ist ortsgebunden, d.h. es müssen sowohl der Gast, als auch der Bewohner vor der entsprechenden Station stehen.

Im Zuge einer früheren Diplomarbeit des Schuljahres 2017/18 an der HTL Anichstraße entwickelten Sebastian Wagner und Tobias Pfluger eine Videogegensprechanlage, welche mehrere Innenstationen und die bidirektionale Video- und Tonübertragung unterstützt. Die Idee ist, dass in einem Wohnungskomplex pro Partei eine Innenstation verbaut wird, sowie eine Außenstation am Hauseingang. Die einzelnen Innenstationen, also Wohnungsparteien, können dann nicht nur mit der Außenstation, sondern auch mit anderen Innenstationen per Video und Ton kommunizieren. Die Stationen wurden mit Hilfe eines RPi realisiert, um die Kosten gering zu halten.

Am Anfang des 5. Schuljahres im Herbst 2019 wurde uns von unserem FI-Professor DI(FH) Mario Prantl angeboten, dieses Projekt durch eine Smartphone-App und eine Hardware-Überarbeitung zu verbessern. Für die Entwicklung der App sollte das Xamarin-Rahmenwerk verwendet werden, damit die Applikation auf sowohl Android-, als auch iOS-Geräten lauffähig ist. Der Benutzer soll über die App das Video der entsprechenden Station abrufen und mit dem Gesprächspartner sprechen können. Hierfür wird ein aus dem Internet erreichbarer Medien-Verwaltungsserver benötigt, der die Medien-Streams der Einzelstationen empfängt und an die Station des Gesprächspartners überträgt. Die Hardware-Überarbeitung hat mehrere Ziele; zum einen soll dadurch die Komplexität des internen Aufbaus einer Station verringert werden. Zum anderen soll mit Hilfe eines Watchdog-Timers das Langzeit-Betriebsverhalten verbessert werden.

1.1 Funktionalität

Ortsunabhängiger Anlagenzugriff: Um dem Anwender mehr Komfort bieten zu können, beschränkt sich die Anlage und dessen Funktion nicht mehr auf die fest installierte

Station, sondern ist auch über ein mobiles Gerät bedienbar. Dies bietet den Vorteil, dass auch wenn man sich gerade nicht in der Nähe der Innenstation befindet, mit Besuchern kommuniziert werden kann.

Paketdienst: Die Paketübergabe erfolgt üblicherweise persönlich. Wenn der Empfänger nicht zuhause ist, nimmt der Paketzusteller die Ware wieder mit, deponiert sie bei einer Paketabholstelle und hinterlässt dem Empfänger lediglich eine Benachrichtigung. Diese Vorgangsweise ist besonders ärgerlich, wenn man den Paketzusteller um wenige Minuten versäumt hat oder nicht schnell genug zur Innenstelle gelangen konnte. Beispielsweise befindet sich der Bewohner gerade auf dem Dachboden oder im Keller. Mit einer Smartphone-App ist eine sofortige Kontaktaufnahme mit dem Paketzusteller möglich und verhindert eine unnötige zusätzliche Bearbeitung des Paketversandes.

Sicherheit: Die neu gewonnene Ortsunabhängigkeit der Anlage bietet eine höhere Einbruchssicherheit, da man jederzeit Überblick über erwünschte bzw. unerwünschte Besucher hat. Über die Smartphone-Applikation ist das Öffnen des Türschlosses aus mehreren Gründen nicht möglich:

- Wenn der Benutzer nicht zuhause ist, ist eine Türöffnung in den allermeisten Fällen nicht erwünscht.
- Wenn der Benutzer zuhause ist, muss er sowieso zur Eingangstür gehen, um den Besuch in Empfang zu nehmen. Die im Eingangsbereich installierte Station dient hier zur Öffnung der Haustür.
- Falls unbefugter Zugriff auf die Applikation erfolgen sollte, besteht kein besonderes Sicherheitsrisiko hinsichtlich Einbruchs in die Wohnung.

1.2 Aufgabenteilung

Andreas Grain ist der Projekt-Hauptverantwortliche und zuständig für die App-Programmierung. Dies beinhaltet die Einarbeitung in und Verwendung des Xamarin-Rahmenwerks zur Erstellung einer Multiplattform-Smartphone-Applikation. Diese soll den Video-Stream der gewählten Station abrufen und anzeigen, sowie den Mikrofon-Ton des Mobilgerätes zur Anlage zurücksenden. Weiters ist er für die Einarbeitung in das GStreamer-Rahmenwerk zur zentralen Verarbeitung der Video- und Audio-Daten der Stationen und Mobilgeräte über einen Linux-basierten Server zuständig. Darüber

hinaus organisiert Andreas Grain sämtliche Treffen mit dem Betreuer, achtet auf die Einhaltung des Zeitplans und ist zuständig für das Erstellen des \LaTeX -Dokuments

Matthias Mair ist verantwortlich für die Hardware-Überarbeitung der Station. Diese beinhaltet die Einarbeitung in das Printed Circuit Board (PCB)-Entwicklungsprogramm EAGLE von der Firma Autodesk mit der Version 9.5.2, sowie das Recherchieren und Auswählen möglicher Verstärker-Schaltungen und -Integrated Circuits (ICs) für die Mikrofon- und Lautsprecher-Verstärkung. Im Zuge der Hardware-Überarbeitung sollen die vielen Einzelmodule in einer übersichtlichen Platine vereint werden. Für die Leiterplatte soll die weit verbreitete SMD-Technologie verwendet werden, da diese heutzutage Marktstandard für integrierte Geräte ist und im Vergleich zu traditionellen THT-Platinen viel platzsparender ist. Zusätzlich wird die Station um einen Watchdog-Timer erweitert, den Matthias Mair entwerfen und die dazugehörige Software schreibt.

1.3 Hinweise und Definitionen

In diesem Dokument werden technische Begriffe aus der Elektronik und Softwaretechnik verwendet. Die Bedeutung dieser Begriffe wird aus Gründen der Lesbarkeit nicht umschrieben oder erklärt. Die Autoren setzen daher entsprechende Grundkenntnisse in den Bereichen Elektronik und Softwaretechnik voraus, insbesondere:

- elektrische Grundgrößen wie Strom, Spannung, Widerstand, ...
- Programmier-Konzepte wie Unterprogramme, Schleifen, Zuweisungen, ...
- die Programmiersprachen C# und C++

Die im Dokument verwendeten Strom- bzw. Spannungsangaben sind immer Gleichgrößen (DC) und werden nicht jedes Mal ausdrücklich als solche gekennzeichnet. Sollten abweichende Strom- bzw. Spannungsformen zur Anwendung kommen, so wird dies explizit angeführt.

Teil I

Software - Andreas Grain

2 Rahmenwerk

2.1 Anforderungen

Das Rahmenwerk dient der Vereinfachung des Entwicklungsprozesses. Je nach Anwendung sind von diesem verschiedene Anforderungen zu erfüllen, daher ist die Wahl des richtigen Rahmenwerks besonders wichtig. Dieser Abschnitt beschäftigt sich mit dem Vergleich der Möglichkeiten, sowie einer endgültigen Auswahl eines der Rahmenwerke zur Verwendung im Diplomprojekt. Die geforderten Funktionen für dieses Projekt beinhalten:

- eine einfache Entwicklung der Multiplattform-App. Wenn möglich soll nur eine App geschrieben werden, die dann auf allen Zielplattformen (Android, iOS) lauffähig ist.
- Die Programmierung sollte in einer bereits bekannten Programmiersprache möglich sein. Für die Fachrichtung Elektrotechnik der HTL Anichstraße werden momentan die Sprachen C, C++ und C#, in untergeordnetem Ausmaß auch JavaScript für Webseiten-Entwicklung unterrichtet.
- Das Rahmenwerk und die dazugehörige Entwicklungsumgebung sollten für nicht-kommerzielle Anwendungen wie diese Diplomarbeit frei zur Verfügung stehen.
- Die Entwicklungswerzeuge sollten das Windows-, optional das Linux-Betriebssystem unterstützen.

2.2 Vergleich

Für die App-Programmierung stehen viele verschiedene Rahmenwerke bereits zur Verfügung.

2.2.1 Microsoft Xamarin

Hierbei handelt es sich um ein Rahmenwerk, mit dem man Multiplattform-Applikationen für unter anderem Android, iOS, Universal Windows Platform (UWP) und noch viele weitere Plattformen erstellen kann. Es basiert auf Microsofts .NET-Framework (.NET)

und dem Mono-Projekt, welches sich als Ziel gesetzt hat, das .NET auf andere Plattformen zu portieren. Xamarin bietet mehrere Projekttypen an, darunter Xamarin.Android und Xamarin.iOS für native App-Entwicklung und Xamarin.Forms für großteils plattformunabhängige Entwicklung. Eine Xamarin.Forms-Solution besteht daher aus mehreren Teilen:

- Portable/.NET-Standard Projekt, das den plattformunabhängigen Code beinhaltet
- natives Xamarin-Projekt für jede Zielplattform

Der Vorteil Xamarins ist, dass der Großteil des geschriebenen Codes im plattformunabhängigen Projekt bleibt und nur für wenige Funktionen auf native Programmierung zurückgegriffen wird, wie zum Beispiel für hardwarenahe Audio-Aufnahme. Außerdem ist das Xamarin-Projekt Open-Source, das bedeutet jeder kann den Quellcode betrachten und unter Umständen Verbesserungen vorbringen. Zusätzlich ist es für diese Anwendung kostenfrei.

2.2.2 Webtechnologiebasierte Rahmenwerke

In dieser Kategorie existieren sehr viele verschiedene Rahmenwerke, darunter Apache Cordova, Adobe PhoneGap, sowie Facebook React Native. Am Beispiel von Apache Cordova werden hier die Vor- und Nachteile dieses Ansatzes erläutert.

Cordova ist ein App-Entwicklungs-Rahmenwerk welches ursprünglich von der Firma Nitobi unter dem Name PhoneGap entwickelt wurde. Im Jahr 2011 wurde Nitobi von Adobe aufgekauft. Später wurde neben der Closed-Source-Version auch eine quelloffene Version unter dem Namen Cordova veröffentlicht. [vgl. 1]

Apps werden mittels gängiger Webtechnologie, wie zum Beispiel JavaScript, CSS und ähnlichen, entwickelt und realisiert, weshalb das Rahmenwerk alle gängigen Plattformen unterstützt. Der Vorteil von Cordova und ähnlichen Rahmenwerken liegt im enormen Support dieser Webtechnologien, jedoch sind Sprachen wie JavaScript für Back-End-Programmierung weniger geeignet. Noch dazu kommt, dass an der HTL Anichstraße großteils die Programmierung nur in C und C# gelehrt wird, weshalb die Entwicklung mit JavaScript im vereinbarten Zeitrahmen der Diplomarbeit nicht realistisch umsetzbar ist.

Aufgrund der oben erläuterten Vor- und Nachteile der möglichen Ansätze wurde die Verwendung von Xamarin für diese Diplomarbeit festgelegt.

2.3 Build-Umgebung

2.3.1 Visual Studio 2019

Aufgrund obiger Aufstellung wurde Visual Studio 2019 der Firma Microsoft als Entwicklungsumgebung gewählt. Visual Studio ist das offizielle Werkzeug für die Programmierung mit dem Xamarin-Rahmenwerk und die wahrscheinlich bestbekannte Entwicklungsumgebung überhaupt. Die Community-Edition für Schüler und Private steht gratis zum Download zur Verfügung. Diese beinhaltet alle wichtigen Entwicklungswerkzeuge wie zum Beispiel die automatische Code-Vervollständigung. Für die Nutzung wird nach den ersten 30 Tagen ein Microsoft-Konto benötigt.

Neben der Community-Edition existieren auch noch die Professional- und die Enterprise-Edition. Die Unterschiede der Funktionsumfänge sind in Tabelle 2.1 zusammengefasst.

Der Programmcode wird von Visual Studio 2019 in sogenannten Solutions organisiert. Am Beispiel einer Xamarin.Forms-Applikation lässt sich das sehr gut erläutern; die Solution enthält auf oberster Ebene drei Projekte: das portable Projekt, das Android- und das iOS-Projekt. Ein Projekt kann bereits für sich lauffähig oder nur Teil eines größeren Programms sein.

Visual Studio 2019 unterstützt viele verschiedene Einsatzbereiche, die bei der Installation ausgewählt werden müssen. Eine volle Installation mit allen Funktionen und Projektarten ist zwar möglich, benötigt allerdings bis zu 210 GB Speicherplatz des Systems. Eine typische Xamarin-Installation benötigt etwa 6.25 GB Festplattenspeicher. Für eine solche Installation muss im Visual Studio Installer das „Mobile development with .NET“-Paket ausgewählt und installiert werden. Siehe hierfür Abbildung 2.1.

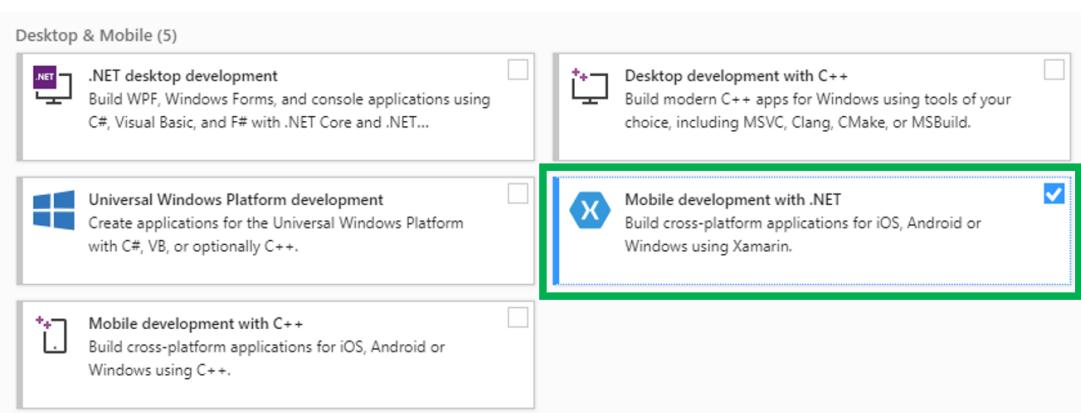


Abbildung 2.1: Auswahl des Xamarin-Rahmenwerks bei der Installation

Function	Community	Professional	Enterprise
Supported Usage Scenarios			
Enterprise		Yes	Yes
IDE			
Live Dependency Validation			Yes
Architectural Layer Diagrams			Yes
Architecture Validation			Yes
Code Clone			Yes
CodeLens	Partial	Yes	Yes
Advanced Debugging And Diagnostics			
IntelliTrace			Yes
Code Map Debugger Integration			Yes
.NET Memory Dump Analysis			Yes
Snapshot Debugger			Yes
Time Travel Debugging (Preview)			Yes
Testing Tools			
Live Unit Testing			Yes
IntelliTest			Yes
Microsoft Fakes (Unit Test Isolation)			Yes
Code Coverage			Yes
Cross-platform Development			
Embedded Assemblies			Yes
Xamarin Inspector			Yes
Xamarin Profiler			Yes

Tabelle 2.1: Unterschiede Visual-Studio-Editionen [vgl. 2]

Es empfiehlt sich, zusätzlich das Paket für „.NET desktop development“ zu installieren, damit neue, noch nicht bekannte .NET-Technologien in einer einfacheren Konsolen-Anwendung ausprobiert werden können. Das Paket benötigt ungefähr weitere 0.8 GB.

Auf nähere Details der Installation von Visual Studio wird hier nicht eingegangen, diese können aber auf der Microsoft-Dokumentationsseite [vgl. 3] nachgeschlagen werden.

2.3.2 Android SDK

Um Xamarin-Applikationen für Android-Geräte entwickeln zu können wird das von Google bereitgestellte Android SDK benötigt. Die einzelnen Komponenten können im Android-SDK-Manager des Visual Studio installiert werden. Dieser ist wie in Abbildung 2.2 dargestellt über das Menüband aufrufbar.

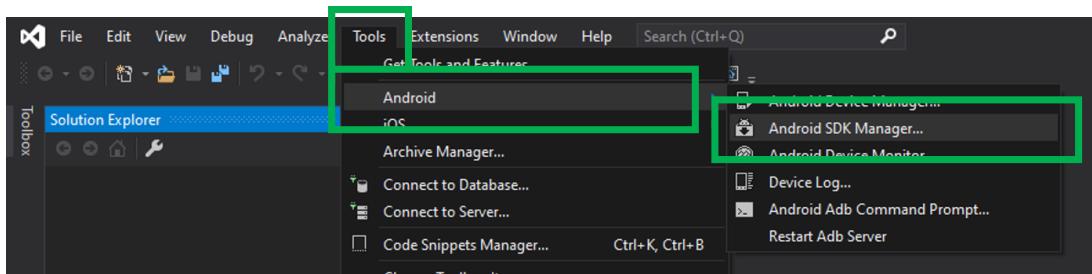


Abbildung 2.2: Android SDK Manager

Der SDK-Manager ist in zwei Seiten unterteilt – „Platforms“ und „Tools“.

Auf der „Platforms“-Seite können die benötigten SDK-Teile für jede Android-Version installiert werden. Benötigt wird nur die Version, mit der das Projekt später kompiliert werden soll. Außerdem kann man auf dieser Seite des SDK-Managers System-Abbildungen für den Android Emulator installieren.

Über die „Tools“-Seite können verschiedenste Entwicklungswerkzeuge installiert werden. Grundsätzlich erforderlich sind „Android SDK Tools“, „Android SDK Platform Tools“ und „Android SDK Build Tools“, um Android-Apps mit Xamarin entwickeln zu können. Für dieses Projekt werden Push-Benachrichtigungen via Google Cloud Messaging verwendet, daher werden noch zusätzlich die Pakete für die „Google Play Services“ sowie die „Google Cloud Messaging for Android Library“ installiert. Die dafür notwendige Auswahl wird in Abbildung 2.3 dargestellt. Wie Push-Benachrichtigungen genauer funktionieren, wird im Kapitel 4 behandelt.

2.3.3 Build-Vorgang

Um den geschriebenen Code in eine lauffähige Applikation umzuwandeln, wird ein sogenannter Compiler benötigt. Die Aufgabe des Compilers ist es, die geschriebene Hochsprache entweder direkt in Central processing unit (CPU)-Maschinensprache bzw. eine Art Zwischencode zu kompilieren, d.h. umzuwandeln. Im Fall einer einfachen C#-Konsolenanwendung ist der Build-Vorgang noch relativ simpel.

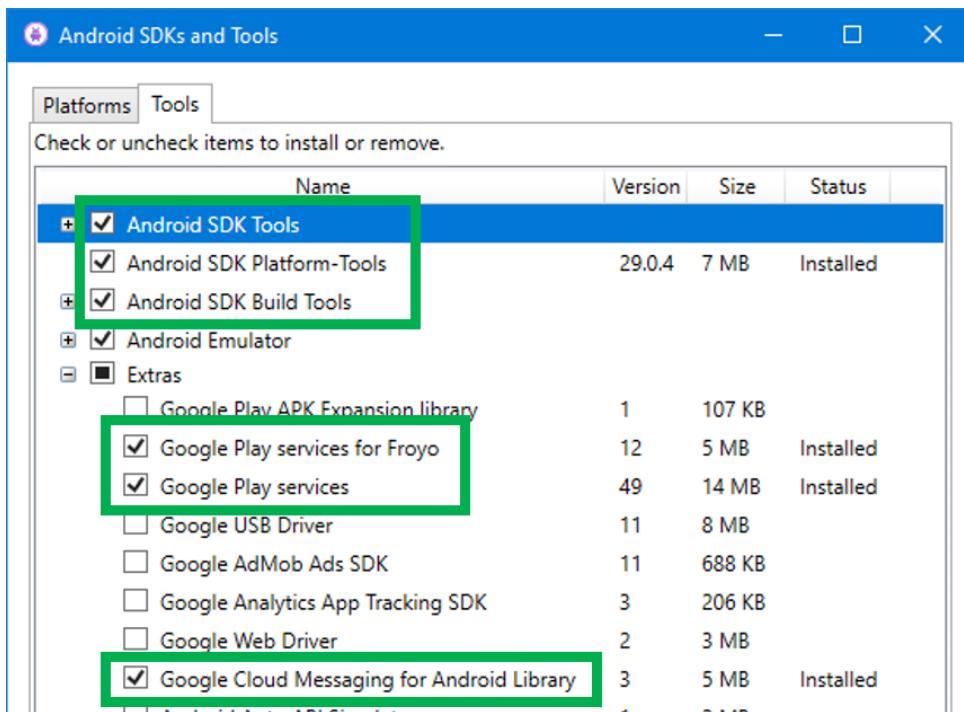


Abbildung 2.3: Notwendige Teile des Android SDK

Zuerst wird der gesamte Code vom Compiler in Common Interface Language (CIL), eine Zwischensprache des .NET, umgewandelt. Die Idee dahinter ist, dass diese CIL-Anweisungen plattformunabhängig sind und später auf der Zielplattform fertig kompiliert werden können. Die CIL-Anweisungen erscheinen nach außen als eine ausführbare .EXE-Datei. Wenn diese ausgeführt wird, wird jedoch zuvor das Programm von der Common Language Runtime (CLR) in die von der CPU ausführbare Maschinensprache übersetzt. Der gesamte Vorgang ist in Abbildung 2.4 dargestellt.

Ähnlich funktioniert dieser Vorgang auch bei einer Xamarin.Forms-App, allerdings wesentlich komplexer. Eine Xamarin.Forms-Solution besteht aus drei Einzel-Projekten (PiBell, PiBell.Android, PiBell.iOS), wobei das .NET-Standard-Projekt als Teil der anderen zwei verbaut werden muss. Dadurch ergibt sich ein wie in Abbildung 2.5 dargestellter Build-Vorgang.

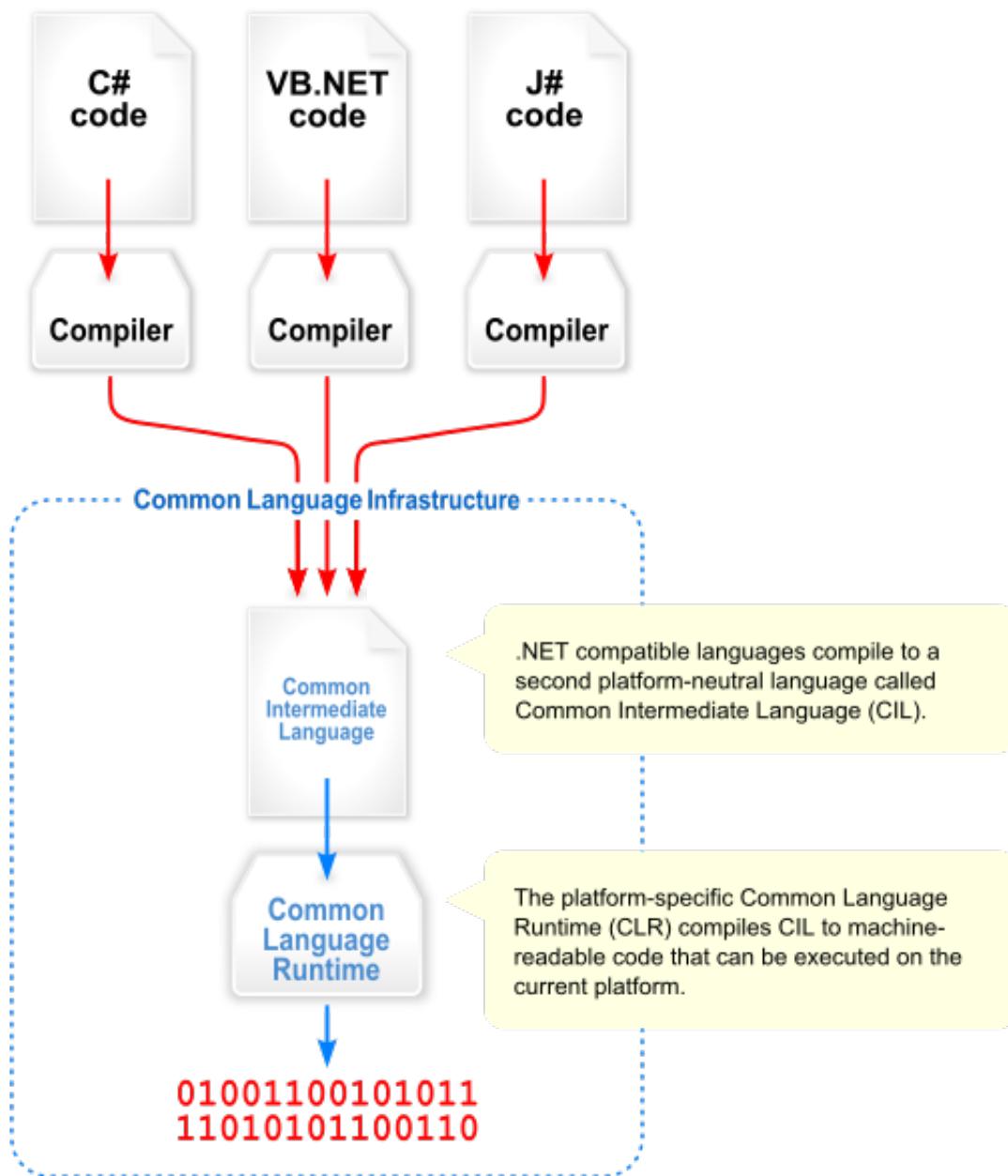


Abbildung 2.4: Build-Vorgang einer .NET-Applikation [4]

Abbildung 2.5: Build-Vorgang einer Xamarin.Forms-Solution

3 Verwendete Software-Module

3.1 Real Time Streaming Protocol

RTSP ist ein Netzwerkprotokoll zum Aufbauen und Verwalten von Netzwerkverbindungen zur Übertragung kontinuierlicher Medien-Daten (Streams). Dieses Netzwerkprotokoll ist im Internet Engineering Task Force (IETF)-Dokument *RFC 2326 - Real Time Streaming Protocol (RTSP)* festgelegt. Es wurde gemeinsam von Schulzrinne, Rao und Lanphier entwickelt. [vgl. 5]

3.1.1 RTSP Version 2

Im Zuge dieser Diplomarbeit wird die erste Version des RTSP-Protokolls verwendet. Neben der verwendeten Version 1 existiert auch eine neue Version 2, welche im IETF-Dokument *RFC 7826 - Real-Time Streaming Protocol Version 2.0* definiert ist. Die neue Version wurde von Schulzrinne, Rao, Lanphier u. a. erarbeitet. [vgl. 6]

Es wurde dennoch die erste Version verwendet, da die Stationen dieses bereits verwenden und die neue Version nicht rückwärtskompatibel ist.

RTSP wird in der Industrie vielseitig verwendet, um Live-Übertragungen von Überwachungskameras zu verwalten. Eine weitere Anwendung des RTSP Protokolls ist das Streamen von Medien-Daten zu einem Server, der diese dann beispielsweise transcodiert oder aufnimmt.

3.1.2 Anfragen-Aufbau

Das RTSP-Protokoll definiert mehrere Anfragen zur Verwaltung der Netzwerkverbindung. Diese Anfragen werden, ähnlich wie beim Hypertext Transfer Protocol (HTTP), in unverschlüsseltem Plain-Text verschickt. In Tabelle 3.1 sind sämtliche Anfragen des RTSP-Protokolls und deren Übertragungsrichtung aufgelistet.

Das Aussehen einer solchen Anfrage wird anhand des Beispiels der DESCRIBE-Methode veranschaulicht. Der Client beginnt die Anfrage mit dem Uniform Resource Locator (URL) des Medien-Streams, den er abrufen möchte und teilt dem Server mit, welche Formate er versteht.

Methode	Richtung	Objekt	Verbindlichkeit
DESCRIBE	C→S	P,S	empfohlen
ANNOUNCE	C→S, S→C	P,S	optional
GET_PARAMETER	C→S, S→C	P,S	optional
OPTIONS	C→S, S→C	P,S	erfordert, (S→C: optional)
PAUSE	C→S	P,S	empfohlen
PLAY	C→S	P,S	erfordert
RECORD	C→S	P,S	optional
REDIRECT	S→C	P,S	optional
SETUP	C→S	S	erfordert
SET_PARAMETER	C→S, S→C	P,S	optional
TEARDOWN	C→S	P,S	erfordert

Richtung: C...Client, S...Server
 Objekt: P...Präsentation, S...Streams

Tabelle 3.1: Anfrage-Arten des RTSP-Protokolls [5, aus dem Englischen übersetzt]

- 1 C->S: DESCRIBE rtsp://server.example.com/fizzle/foo RTSP/1.0 1
- 2 CSeq: 312 2
- 3 Accept: application/sdp, application/rts1, application/mheg 3

Der Server antwortet auf diese Anfrage mit dem Session Descriptor, der alle wichtigen Informationen des Medien-Streams beinhaltet, wie zum Beispiel Video- und Audio-Format, Übertragungsmethode, sowie allgemeine Informationen wie Titel und Beschreibung.

- 1 S->C: RTSP/1.0 200 OK 1
- 2 CSeq: 312 2
- 3 Date: 23 Jan 1997 15:35:06 GMT 3
- 4 Content-Type: application/sdp 4
- 5 Content-Length: 376 5
- 6 6
- 7 v=0 7
- 8 o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4 8
- 9 s=SDP Seminar 9
- 10 i=A Seminar on the session description protocol 10
- 11 u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps 11
- 12 e=mjh@isi.edu (Mark Handley) 12
- 13 c=IN IP4 224.2.17.12/127 13

```
14      t=2873397496 2873404696          14
15      a=recvonly                      15
16      m=audio 3456 RTP/AVP 0           16
17      m=video 2232 RTP/AVP 31          17
18      m=whiteboard 32416 UDP WB        18
19      a=orient:portrait               19
```

3.2 LibVLC Sharp

LibVLC ist ein Multimedia-Rahmenwerk der Non-Profit-Organisation VideoLAN, welche durch den VLC Media Player und dessen Funktionsumfang bekannt wurde. Der VLC Media Player funktioniert sozusagen als grafische Oberfläche zur LibVLC-Programmbibliothek. Die Bibliothek beinhaltet viele nützliche Funktionen, darunter die folgenden:

- Videos nahezu beliebigen Formates decodieren und darstellen
- Video und Audio aufnehmen und transcodieren für die weitere Verwendung
- Live-Streaming von Video-Daten, Webcam-Bild und Mikrofon-Ton.

LibVLC ist eine Bibliothek für die C-Sprache. Um diese in einem C#-Projekt verwenden zu können sind Bindings notwendig, welche Zugriff auf die C-Bibliothek geben. LibVLC wurde von der Non-Profit-Organisation VideoLAN entwickelt und erstmals am 1. Februar 2001 veröffentlicht. [vgl. 7]

Im Xamarin.Forms-Projekt wird die zum Zeitpunkt der App-Entwicklung aktuelle Version 3.4.2 der LibVLC Sharp Bindings verwendet. Bindings erlauben es dem Programmierer, auf Programmbibliotheken zuzugreifen, welche für eine andere Sprache geschrieben wurden. Man kann sich Bindings wie Kleber vorstellen, der die Funktionen der einen Programmiersprache mit denen der anderen Sprache verbindet. Die Version 3.4.2 ist inzwischen nicht mehr die neueste Version, jedoch ist sie in Hinsicht auf Funktionalität deckungsgleich mit der neuesten Version.

3.2.1 Erwähnenswerte Funktionen

Die LibVLC-Sharp-Bindings ermöglichen die Verwendung des Model-View-ViewModel (MVVM)-Modells, welches vom Windows Presentation Foundation (WPF)- und Silver-

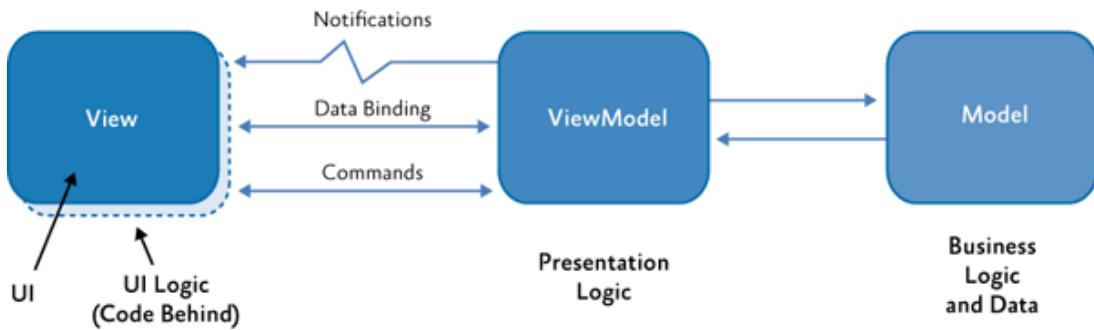


Abbildung 3.1: Datenfluss MVVM [9]

light-Architekten John Gossman im Jahr 2005 auf seinem Blog vorgestellt wurde. [vlg. 8, The Evolution of Model-View-ViewModel]

Die Philosophie hinter diesem Modell ist, die Oberfläche (View) bis auf wenige Schnittstellen komplett vom eigentlichen Code zu trennen. Dies ermöglicht es ohne größeren Aufwand, die Oberfläche zu ändern oder gar auszutauschen. Folgende Terminologie wird in diesem Zusammenhang verwendet:

- Models ... beinhalten die Programmdaten. Meist werden einfache Klassen oder Strukturen hierfür verwendet.
- Views ... Die grafische Oberfläche und deren Elemente wie Buttons, TextBoxen, und ähnliche
- ViewModels ... wandeln die Daten der Models so um, dass sie von der Oberfläche dargestellt werden können. ViewModels sind meist als Klasse ausgeführt.

Die Daten der Oberfläche und die Daten des Models bleiben zueinander konsistent. Sobald sich ein Wert auf einer Seite ändert, wird er sofort auf der anderen aktualisiert. Die hier beschriebenen Vorgänge sind in Abbildung 3.1 nochmals dargestellt.

Das MVVM-Modell bietet sich auch an, wenn die Daten des Programms oft in eine andere Form gebracht werden müssen, bevor sie dargestellt werden können. Zum Beispiel wird vom Model die aktuelle Zeit als Ganzzahl abgespeichert, welche nur die vergangenen Millisekunden seit einem bestimmten Zeitpunkt zählt; um die Zeit allerdings darstellen zu können, muss diese Ganzzahl zuerst in ein lesbare Datumsformat gebracht werden. Diese Aufgabe würde das ViewModel übernehmen.

3.2.2 Nachteile

Sowohl die ursprüngliche LibVLC-Bibliothek, als auch die entsprechenden C#-Bindings sind auf der offiziellen Dokumentationsseite nur spärlich dokumentiert. [vgl. 10]

Beispielsweise verwenden alle Beispielprojekte der LibVLC Sharp Bindings das MVVM-Modell, daher ist nicht bekannt, ob eine Implementierung ohne dieses Modell überhaupt möglich ist. Weiters sind die einzelnen Funktionen zwar auf der Dokumentationsseite aufgelistet, allerdings nicht sehr ausführlich beschrieben. [vgl. 10]

3.2.3 Alternativen

Während der Recherche wurden wenige Alternativen gefunden. Diese Alternativen sind jedoch seit langem nicht mehr in Entwicklung und sind daher nicht für die Verwendung empfohlen. Die LibVLC-Bibliothek ist momentan die einzige kostenfreie Variante, RTSP-Streams mit Xamarin auf dem Mobilgerät abspielen zu können. Es gibt Bibliotheken von VASTreaming, jedoch sind diese kostenpflichtig zu erwerben, was für diese Diplomarbeit keine Option ist. [vgl. 11, Pricing]

3.3 GStreamer

3.3.1 Modularität

GStreamer ist ein extrem leistungsfähiges und vielseitiges Framework zur Erstellung von Medien-Streaming-Anwendungen. Viele der Vorteile des GStreamer-Frameworks liegen in seiner Modularität: GStreamer kann neue Plugin-Module nahtlos integrieren. Aber da Modularität und Leistungsfähigkeit oft mit einem Preis für eine größere Komplexität einhergehen, ist das Schreiben neuer Anwendungen nicht immer einfach. [12, aus dem Englischen übersetzt]

GStreamer ist Pipeline-basiert, d.h. die Funktion wird mit einer Vielzahl von hintereinander geschalteten Filtern bestimmt. Eine Pipeline hat immer einen Eingang (Source) und einen oder mehrere Ausgänge. Zwischen diesen können sich beliebig viele Filter befinden, die unterschiedlichste Funktionen haben können:

- Signale de- und encodieren (Codec, z.B. x264enc)
- Größe und Position von Videos verändern (Caps, z.B. video/x-raw)
- die Pipeline in zwei Teile auftrennen, z.B. in Video und Audio (Demux)

- Daten zu einem Netzwerk-Paket zusammenbündeln (z.B. rtph264pay)
- etc.

Ein Filter hat bestimmte Ein- und Ausgangsformate, die er unterstützt. Als Beispiel nehmen wir den H264-Encoder: dieser unterstützt für das Eingangssignal Raw-Video, also decodierte Binärdaten. Am Ausgang kommt logischerweise ein H264-codierter Video-Stream heraus.

3.3.2 Version

Das GStreamer-Rahmenwerk wird aktuell vom GStreamer-Team fortlaufend upgedatet und verbessert. Die erste Version des GStreamer-Rahmenwerks wurde im Jahr 2001 mit der Versionsnummer 0.1.0 veröffentlicht. Im Jahr 2012 kam eine große Umstellung, bei der grundlegende Teile des Rahmenwerks ausgetauscht wurden, wodurch einige Teile nicht mehr kompatibel waren. Ab diesem Zeitpunkt begann die Versionsnummer mit 1.x.x, um die neuen Versionen deutlich von den Vorgängern abzutrennen. Die aktuelle Version des GStreamer-Rahmenwerks zum Verfassungszeitpunkt dieser Arbeit ist die Version 1.16.2.

Aktuell wird die Version 1.16.2 verwendet. Da aber alle 1.x.x-Versionen mehr oder weniger miteinander kompatibel sind, kann sich das ändern, sobald ein neues Update veröffentlicht wird. Es gibt keinen besonderen Grund auf genau dieser Version zu bleiben.

3.3.3 Alternativen

Es wurden während der Recherche einige mögliche Alternativen für das GStreamer-Rahmenwerk gefunden:

- FFmpeg, ein Multimedia-Rahmenwerk, das vor allem im Bereich Transcodierung Verwendung findet. Es bietet alle Funktionen des GStreamer-Rahmenwerks an.
- LibVLC, die Grundlage des VLC Media Players, welche vor allem zur Wiedergabe verwendet wird. Diese Bibliothek kann zur Transcodierung und Live-Übertragung genutzt werden, ist allerdings nicht dafür optimiert.

3.3.4 Vorteile

GStreamer wird in diesem Projekt verwendet, da bei Live-Video-Übertragung möglichst wenig Latenz vorkommen soll. Das GStreamer-Rahmenwerk ist in dieser Hinsicht den anderen Multimedia-Rahmenwerken weit voraus. Zusätzlich gibt es von GStreamer keine einschränkenden Vorgaben, was Ein- und Ausgangsformat betrifft. Es kann jede beliebige Quelle mit jedem beliebigen Output verknüpft werden, solange die richtige Pipeline verwendet wird. Weiters ist das Projekt quelloffen, d.h. man ist in Bezug auf die Installationsdateien nicht auf den Hersteller angewiesen, sondern kann das Programm selbst für die jeweilige Plattform kompilieren, wenn der Hersteller das noch nicht getan hat. Außerdem kann jeder Entwickler Vorschläge für Änderungen am Code sowie Bugfixes vorbringen.

3.3.5 Funktionsweise

Die detaillierte Funktionsweise des GStreamer-Rahmenwerks lässt sich am besten anhand eines realistischen Beispiels erklären. In diesem Beispiel wird ein in Echtzeit generiertes Testvideosignal zuerst auf 1280x720 Pixel vergrößert und danach als H264-encodierter Stream via Real-time Transport Protocol (RTP) verschickt. Mit dem Kommandozeilen-Programm `gst-launch-1.0.exe` kann ohne großen Aufwand eine Pipeline aufgebaut werden:

```
1  gst-launch-1.0 videotestsrc !
   ↳  video/x-raw, width=1260, height=720, framerate=20/1 !
   ↳  autovideoconvert ! queue ! x264enc tune=zerolatency bitrate=4096
   ↳  speed-preset=superfast ! queue ! rtph264pay config-interval=1
   ↳  mtu=1300 ! udpsink host=127.0.0.1 port=5000
```

Der Filter „queue“ wird hier verwendet, um vor und nach dem Encoding-Vorgang ein kurzes Video-Segment in einem Zwischenspeicher zu behalten. Dies ist notwendig, da der H264-Codec mehrere Video-Frames auf einmal komprimiert.

3.3.6 Zusatzmodul

`gst-rtsp-server` ist ein Zusatzmodul für das GStreamer- Rahmenwerk, welches die Verwendung des RTSP-Protokolls für Server-Applikationen bereitstellt. Mit diesem Modul können Medien-Streams, welche mit GStreamer verarbeitet wurden, mittels dem RTSP-Protokoll den Netzwerk bereitzustellen.

Das GitHub-Repository des Zusatzmoduls beinhaltet einige Beispielapplikationen, darunter auch `test-launch`. Dieses Anwendungsbeispiel nimmt ähnlich wie das vorhin beschriebene `gst-launch-1.0` als Kommandozeilenargument eine GStreamer-Pipeline an. Der resultierende Medienstream wird von der Anwendung in einen RTSP-Stream verpackt und dem Netzwerk bereitgestellt.

3.4 Live555 Proxy

Ein RTSP-Stream ist grundsätzlich eine Unicast-Verbindung zwischen Server und Client. Falls mehrere Clients den gleichen Medien-Stream abrufen möchten, muss der Server diesen für jeden Client einzeln transcodieren und verschicken. Dies stellt eine große Belastung für den Server und dessen Internetanbindung dar. Mit Hilfe eines Proxy-Servers muss der Medien-Stream nur einmal verarbeitet werden, bevor er dann an die einzelnen Clients verteilt wird.

Der Live555 Proxy Server ist eine quelloffene Applikation, welche für genau diese Aufgabe von Live Networks Inc. entwickelt wurde. Mit Hilfe des Live555 Proxy Servers wird ein RTSP-Stream einmalig eingelesen und an alle verbundenen Clients verteilt. Dadurch bleibt die CPU-Belastung des Servers unabhängig von der Anzahl der verbundenen Clients, wodurch er unter Umständen mehrere verschiedene Streams bereitstellen könnte. Für jeden Medien-Stream, den der Server anbietet, wird eine weitere Live555-Proxy-Server-Instanz benötigt.

Das ursprüngliche Veröffentlichungsdatum des Live555 Proxy Servers konnte nicht gefunden werden. Bei Analyse der Google-Such-Trends zum Begriff Live555 wurde festgestellt, dass Suchanfragen mit Oktober 2005 gestartet haben. Daher lässt sich vermuten, dass das Programm kurz vor diesem Zeitpunkt veröffentlicht wurde. [vgl. 13, Interest over time]

Neben dem Live555 Proxy Server entwickelt das Unternehmen mehrere Projekte:

- Live555 Media Server, mit dem lokale Medien-Dateien per RTSP dem Netzwerk bereitgestellt werden können.
- liveCaster, zur Multicast-Übertragung von MP3-Dateien
- diverse Kommandozeilenprogramme zum Senden und Empfangen von Echtzeit-Streams

Die von Live Networks Inc. entwickelten Programme und Bibliotheken finden in vielen bekannten Applikationen Anwendung, darunter auch der VLC Media Player mit der dazugehörigen LibVLC. In diesem Programm werden die Live555-Bibliotheken zum Empfangen und Entschlüsseln von RTSP-Streams genutzt.

4 Push-Benachrichtigung

4.1 Push Notification Service

Eine Notification dient dazu den Benutzer über etwas zu informieren. Dies kann ein Kalender-Eintrag, eine eingetroffene SMS-Nachricht oder ein anderes Ereignis sein. Diese Art der Benachrichtigung wird meist über eine Local Notification erzeugt. Eine Local Notification wird aufgrund eines lokal am Gerät auftretenden Ereignisses ausgelöst. Ein Beispiel ist in Abbildung 4.1 zu sehen.

Im Gegensatz dazu dient eine Push Notification dazu, ein externes Ereignis auf einem anderen Gerät zu verarbeiten. Das Empfängergerät zeigt in den meisten Fällen eine zusätzliche Local Notification dem Benutzer an. Beispiele für Push Notifications sind Katastrophenwarnungen, neue Instant-Messenger-Nachrichten, oder ähnliche Ereignisse, die nicht auf dem Empfängergerät ihren Ursprung haben.

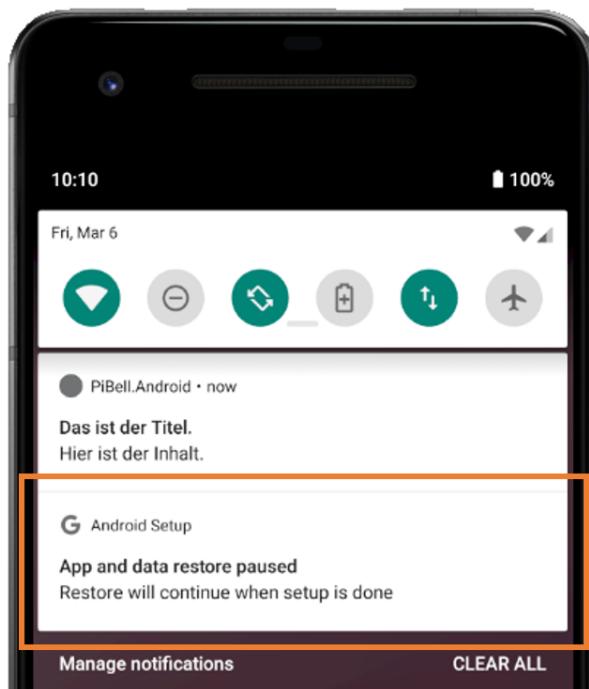


Abbildung 4.1: Lokale Benachrichtigung

4.1.1 Funktionsweise

Eine Push Notification wird immer von einem sogenannten Push Notification Service an alle Zielgeräte geschickt. Dieser Service wird vom jeweiligen Betriebssystem-Hersteller bereitgestellt. Im Fall der hier entwickelten App sind das Google und Apple, mit ihren Push Notification Services Google Cloud Messaging (GCM) und Apple Push Notification Service (APNs).

Der Service ist dafür verantwortlich, die Benachrichtigung an alle registrierten Zielgeräte zu schicken. Wenn das Zustellen nicht möglich ist, bricht der Service die Zustellung nach mehreren Versuchen für dieses Gerät ab. Alle anderen Geräte erhalten die Benachrichtigung. Daher ist es nicht sichergestellt, dass eine Push-Benachrichtigung bei jedem Gerät ankommt.

Zusätzlich zum Nachrichtentext können bei einer Push Notification zusätzliche Daten mitgeschickt werden. Diese können zum Beispiel die App über die genaue Benachrichtigungsursache informieren.

4.2 AppCenter Push

AppCenter Push hat in erster Linie die Funktion, den Zugriff auf die verschiedenen Push Notification Services zu vereinheitlichen und damit einfacher zu gestalten. Microsoft AppCenter verwaltet alle Server-Application Programming Interface (API)-Schlüssel, Zielgerätelisten und ähnliche Einstellungen, um den Prozess des Benachrichtigungssendens zu vereinfachen. AppCenter Push ist entweder über das Web-Interface unter <https://appcenter.ms/> oder über die AppCenter API erreichbar und verwendbar.

Die AppCenter Push SDK abstrahiert den ganzen Prozess der Geräte-Registration, des Registrieren des Eingangs-Events und versteckt alles hinter einer einzelnen simplen Funktion `AppCenter.Start("App Secret", typeof(Push))`. Das App Secret ist der von AppCenter zugewiesene Schlüssel mit dem sichergestellt wird, dass das Gerät für das richtige AppCenter Projekt registriert wird. Ohne diese SDK müsste in der App viel mehr konfiguriert werden, was einen höheren Programmieraufwand und größere Fehlerrate darstellt. Außerdem würde das Rad neu erfunden werden.

In der entwickelten App wird die AppCenter-SDK-Version 2.6.4 verwendet. Zu beachten ist, dass AppCenter Push nicht mehr weiterentwickelt und irgendwann dieses Jahr der Service abgeschaltet wird, wie es John Wargo in seinem Blog-Post am 3. Februar 2020 schrieb. Bevor AppCenter Push endgültig terminiert wird, will Microsoft detaillierte Anleitungen zum Umstieg auf Azure bereitstellen.

5 Programm-Dokumentation

5.1 Übersicht

Das im Zuge dieser Diplomarbeit entwickelte Software-Projekt basiert auf einer der von Microsoft bereitgestellten Xamarin.Forms-Vorlagen, welche als Teil von Visual Studio bereitgestellt werden. Es gibt mehrere Auswahlmöglichkeiten, wobei alle Vorlagen bis auf „Blank“ Beispielcode beinhalten. Da der Beispielcode der anderen Vorlagen für dieses Projekt nicht relevant ist und ohnehin gelöscht werden müsste, wird die „Blank“-Vorlage verwendet. Diese Vorlage beinhaltet nur eine Haupt-Oberflächenseite ohne jeglichen Beispielcode.

Der immer wieder vorkommende Name PiBell ist eine freigeistliche Erfindung. PiBell ist der Projektnname, der sich aus Raspberry Pi und „Bell“, dem englischen Wort für Klingel oder Glocke, zusammensetzt.

Visual Studio organisiert alle Programmteile in einer sogenannten Solution, welche diesen Projektnamen trägt. In diesem Fall beinhaltet die Solution „PiBell“ drei Programmteile oder „Projects“. Dieser Aufbau wird in Abbildung 5.1 dargestellt.

PiBell ist das portable Projekt, auch .NET-Standard-Projekt genannt. Dieses beinhaltet den ganzen plattformunabhängigen Code, sowie die Definition der User interface (UI)-Oberfläche mittels Extensible Application Markup Language (XAML)-Dateien. In diesem Projekt befindet sich der größte entwickelte Code-Anteil, da die meisten Funktionen, wie das Abspielen von Videos mit Hilfe der LibVLC-Bindings, plattformunabhängig funktionieren und deswegen geteilt werden können.

PiBell.Android beinhaltet allen Code, der plattformspezifisch für Android geschrieben wurde. Unter anderem ist hier die Android-Implementation des Mikrofon-Aufnahmeservices, sowie das Berechtigungs-Management enthalten. Die Mikrofon-Aufnahme erfolgt auf einer sehr hardwarenahen Ebene, was die Plattform-Abhängigkeit erklärt.

PiBell.iOS beinhaltet wie PiBell.Android den plattformspezifischen Code für die iOS-Plattform. Aufgrund mangelnder Entwicklungswerkzeuge wird dieser Teil nicht großartig behandelt.

Abbildung 5.1: Aufbau der Solution

5.2 Portable Project

Im folgenden Teil werden die wesentlichen Teile des portablen Projektes PiBell beschrieben. In den einzelnen Programmteilen gibt es immer wiederkehrende Abläufe, wie Initialisierung, die nur einmal beschrieben werden. Die von Microsoft vorgefertigten Teile werden nicht behandelt. Der Fokus liegt damit auf den selbst erstellten Programmteilen. Die Programmteile werden generell in derselben Reihenfolge behandelt, wie sie vom Programm aufgerufen werden.

Textsegmente mit vorgestellter Zeilenummer sind direkt vom Programm eingefügt. Der beschreibende Text bezieht sich auf den Programmteil mit Hilfe der Zeilenummern.

5.2.1 App.xaml.cs

Im diesem Programmteil wird die Applikation grundlegend initialisiert. Die Initialisierung beinhaltet unter anderem das Laden aller benötigten Programm-Bibliotheken und NuGet-Pakete.

```

17 public App()
18 {
19     InitializeComponent();
20     Core.Initialize();
21     MainPage = new MainPage();
22
23
24     AppCenter.LogLevel = Microsoft.AppCenter.LogLevel.Verbose;
25     AppCenter.Start("android={Your Android App secret here};"
26                     "uwp={Your UWP App secret here};"
27                     "ios={Your iOS App secret here}",

```

```

28     typeof(Analytics), typeof(Crashes), typeof(Push));
29 }
```

28
29

19-20: Die Funktionen InitializeComponent() und Core.Initialize sind vorgefertigte Funktionen; diese sind zu verwenden, um die Applikation und alle benötigten Bibliotheken zu initialisieren.

21: Die Anweisung MainPage = new MainPage() erzeugt eine neue Oberfläche, mit der der Benutzer mit der App interagiert. Diese Oberfläche erscheint nach dem App-Start am Bildschirm und passt sich automatisch der physikalischen Bildschirmgröße an.

24-28: Mit AppCenter.LogLevel kann festgelegt werden, welche Log-Nachrichten angezeigt werden. In diesem Fall werden Alle Nachrichten aktiviert. Anschließend wird das AppCenter SDK mit AppCenter.Start(..) gestartet. Am Ende dieser Anweisung werden alle Module (Analytics, Crashes und Push), die dazu gestartet werden sollen, angegeben.

5.2.2 MainPage.xaml

In einer XAML-Datei wird das Aussehen der Oberfläche nach dem What You See Is What You Mean (WYSIWYM)-Prinzip beschrieben. Das Format ähnelt sehr einer Extensible Markup Language (XML)-Datei, in der Hinsicht, dass ein Element mit einer spitzen Klammer (<) beginnt und mit einem Schrägstrich und einer spitzen Klammer (/>) endet. Einige Elemente, wie zum Beispiel Listen oder ein Grid, können sogenannte Kind-Elemente beinhalten.

```

3 <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"           3
4   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"             4
5   xmlns:local="clr-namespace:PiBell"                                     5
6   xmlns:piBell="clr-namespace:PiBell;assembly=PiBell"                  6
7   xmlns:shared="clr-namespace:LibVLCSharp.Forms.Shared;"                7
8     ↳ assembly=LibVLCSharp.Forms"                                         8
9   x:Class="PiBell.MainPage"                                              9
10  Title="Main Page"                                                 9
11  BackgroundColor="#ff1b1b1b">
```

3-8: Am Anfang der XAML-Datei wird der allgemeine Seiten-Typ, gemeinsam mit einigen Quell-Abkürzungen (Namenspräfix für Xamarin-fremde Elemente) definiert. In diesem Fall handelt es sich um eine simple ContentPage mit den Standard-Abkürzungen. Zur Verwendung der LibVLC wurde in Zeile 7 eine zusätzliche Quelle definiert. Die neu hinzugefügte Quelle erlaubt den Zugriff auf Elemente wie die VideoView, welche ein Video als Teil der Oberfläche darstellt.

9: Mit x:Class wird die Programmklasse angegeben, die den Code der Oberfläche beinhaltet. In dieser Klasse finden sich alle Event-Handler für jegliche Oberflächenereignisse, wie das Drücken eines Buttons.

10: Das Festlegen der Title-Variable ist in diesem Fall optional. Falls eine Master-Detail-Page-Struktur verwendet werden würde, wäre der Titel oben in der Kopfzeile der App zu sehen. Da es sich hier aber um ein Single-Page-Layout handelt, ist der Titel nicht zwingend notwendig.

11: Mit der Variable BackgroundColor wird die Hintergrundfarbe der Oberfläche definiert. Die gewünschte Farbe wird als Hexadezimal-Zahl mit dem Format Alpha-Rot-Grün-Blau angegeben. Der Alpha-Wert legt die Durchsichtigkeit der Farbe fest. Es wird ein dunkles Design verwendet, da dieses in einer wenig beleuchteten Umgebung die Augen besser schont.

```

12 <ContentPage.Content>                                12
13   <Grid Margin="10" x:Name="MainGrid">                13
14     <Grid.RowDefinitions>                            14
15       <RowDefinition Height=".7*" />                  15
16       <RowDefinition Height="3*" />                   16
17       <RowDefinition Height="2*" />                   17
18     </Grid.RowDefinitions>                          18
19     <Grid.ColumnDefinitions>                        19
20       <ColumnDefinition Width="*" />                 20
21     </Grid.ColumnDefinitions>                      21

```

12: Mit dieser Zeile wird das Element ContentPage.Content geöffnet. Content steht hier für den gesamten Oberflächeninhalt, also alle Schaltflächen und Anzeigen.

13-21: Als erstes und einziges Kind-Element von ContentPage.Content wird die Komponente Grid gewählt. Diese ermöglicht es, mehrere Kind-Elemente in einer schach-

brettartigen Anordnung zu organisieren. Hierfür werden mit Hilfe der Row- und ColumnDefinitions die Höhen und Breiten der einzelnen Zeilen und Spalten festgelegt. Der Stern (*) in der Längenangabe bedeutet relatives Maß bezogen auf die verfügbare Breite bzw. Höhe.

```

23 <Entry x:Name="EditMrl" Text="{Binding Mrl, Mode=TwoWay}" Grid.Row="0"      23
    ↳ Grid.Column="0" TextColor="White" />
24
25 <shared:VideoView x:Name="VideoView" Grid.Column="0" Grid.Row="1"           25
    ↳ MediaPlayer="{Binding MediaPlayer}"/>
```

23: Hier wird das erste eigenständige Oberflächen-Element erzeugt. Es handelt sich um ein Text-Eingabe-Feld (Entry), mit dem zu Testzwecken die Serveradresse eingegeben werden kann. Dies ist notwendig, da die finale Serveradresse noch nicht bekannt ist und sich diese im Zuge der Entwicklung ständig ändert. Für den späteren Betrieb könnte dieses Problem umgangen werden, indem die aktuelle Server-Adresse per Push-Benachrichtigung mitgeschickt wird. Hier ist auch eine Anwendung des MVVM-Modells erkennbar: es wird mit Hilfe des Binding-Schlüsselwortes der Speicherort des angezeigten Textes festgelegt. Mode=TwoWay beschreibt die Richtung der Synchronisierung der Daten. Egal, ob sich der Wert im Speicher oder der Wert in der Anzeige ändert, wird die jeweils andere Seite aktualisiert. Mit Grid.Row und Grid.Column wird die Position im vorhin definierten Grid gesetzt.

25: Das zweite Oberflächen-Element, eine VideoView der LibVLC, wird hier erzeugt. Da die Komponente VideoView nicht standardmäßig in Xamarin enthalten ist, wird das am Anfang der Datei definierte Namenspräfix „shared:“ benötigt.

```

36 <ImageButton x:Name="BtConnect" Source="call.png"                         36
    ↳ BackgroundColor="LimeGreen" Grid.Row="0"
        Grid.Column="0" Margin="10"
37
38     Clicked="BtConnect_Clicked"/>                                         38
```

36: Ein ImageButton kann im Vergleich zu einem normalen Button als Inhalt ein Bild anzeigen. Die Bildquelle wird mit der „Source“-Property gesetzt. Es gibt mehrere mögliche Bildformate, wobei hier PNG aufgrund von Transparenz-Unterstützung gewählt wurde. Ohne Definition der „BackgroundColor“ würde der ImageButton mit der Default-Farbe grau gefüllt werden.

38: Mit dem EventHandler „Clicked“ kann die Funktion, die aufgerufen wird, wenn der ImageButton gedrückt wird, spezifiziert werden.

```

43 <ImageButton x:Name="BtToggleSpeaker" Source="SpeakerMute.png"           43
  ↳ BackgroundColor="Transparent"
44     Grid.Row="1" Grid.Column="0" Margin="10"                                44
45     Clicked="BtToggleSpeaker_Clicked">                                     45
46     <VisualStateManager.VisualStateGroups>                                 46
47         <VisualStateGroup x:Name="SpeakerStates">                         47
48             <VisualState Name="Mute">                                         48
49                 <VisualState.Setters>                                       49
50                     <Setter Property="Source"                                50
51                         Value="SpeakerMute.png" />                         51
52                 </VisualState.Setters>                                       52
53             </VisualState>                                              53
54             <VisualState Name="Unmute">                                         54
55                 <VisualState.Setters>                                       55
56                     <Setter Property="Source"                                56
57                         Value="SpeakerUnmute.png" />                         57
58                 </VisualState.Setters>                                       58
59             </VisualState>                                              59
60         </VisualStateGroup>                                         60
61     </VisualStateManager.VisualStateGroups>                           61
62 </ImageButton>                                                       62

```

47-62: Mit dem VisualStateManager können für die meisten Oberflächen-Elemente bestimmte Zustände definiert werden (z.B. wenn sie gedrückt sind). Hier werden zwei Zustände definiert, die das angezeigte Bild dem Lautsprecher-Zustand anpassen. Über den Code-Behind werden die richtigen VisualStates aufgerufen, was in dem in Abbildung 5.2 gezeigten Verhalten resultiert.

Die hier beschriebenen Konzepte werden mehrmals verwendet, um die Oberfläche aufzubauen. Die Datei als Gesamtheit ergibt die in Bild 5.3 gezeigte Oberfläche.

5.2.3 MainPage.xaml.cs

Dieser Teil beinhaltet den C#-Code, der für die Steuerung und Verwaltung der Oberfläche notwendig ist. Unter anderem wird hier das Verhalten beim Wechseln zwischen

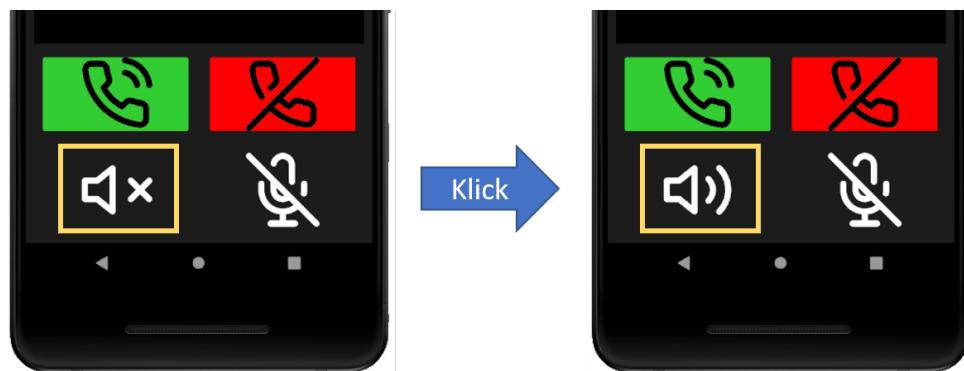


Abbildung 5.2: VisualStateManager

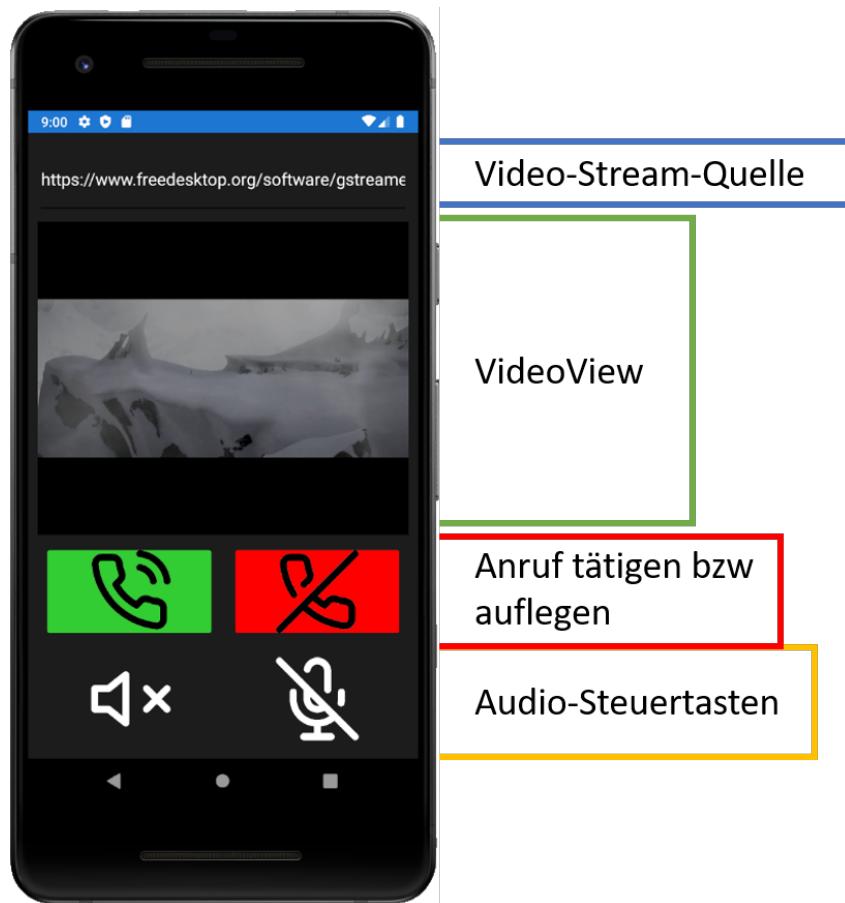


Abbildung 5.3: App-Oberfläche

Hoch- und Querformat festgelegt.

```

36     if (!AppCenter.Configured)          36
37         Push.PushNotificationReceived += async (sender, e) =>      37
38     {
39 #if DEBUG                         39
40     Console.WriteLine("DEBUG - Push-Notification received");        40
41 #endif                           41
42     foreach (string key in e.CustomData.Keys)                      42
43     {
44 #if DEBUG                         44
45     Console.WriteLine($"DEBUG - Custom Data:           45
46     ↳ {key}:{e.CustomData[key]}");
47 #endif                           46
48     if (key == "mrl")               47
49         _player.Mrl = e.CustomData[key].ToString();                  48
50     }
51
52     if (await DisplayAlert("Incoming Call", "Connect now?", "Yes", 51
53     ↳ "No"))
54     {
55         _player.StartCall();                     53
56         _streamer.StartCall();                  54
57     }
58 };

```

Hier wird das Verhalten bei eingehender Push Benachrichtigung festgelegt.

39,41,44,46: Die mit „#“ beginnenden Anweisungen sind sogenannte Preprocessor-Statements, mit denen dem Compiler spezielle Anweisungen gegeben werden können. Hier soll der Programmteil, der sich zwischen `#if DEBUG` und `#endif` befindet, nur im DEBUG-Modus verarbeitet werden. Wenn ein anderer Compiler-Modus ausgewählt ist, werden diese Teile ignoriert.

42-49: Mit einer Schleife werden alle Zusatzdaten der Benachrichtigung überprüft. Wenn der Datensatz `mrl` enthalten ist, wird dessen Wert in `_player.Mrl` gespeichert.

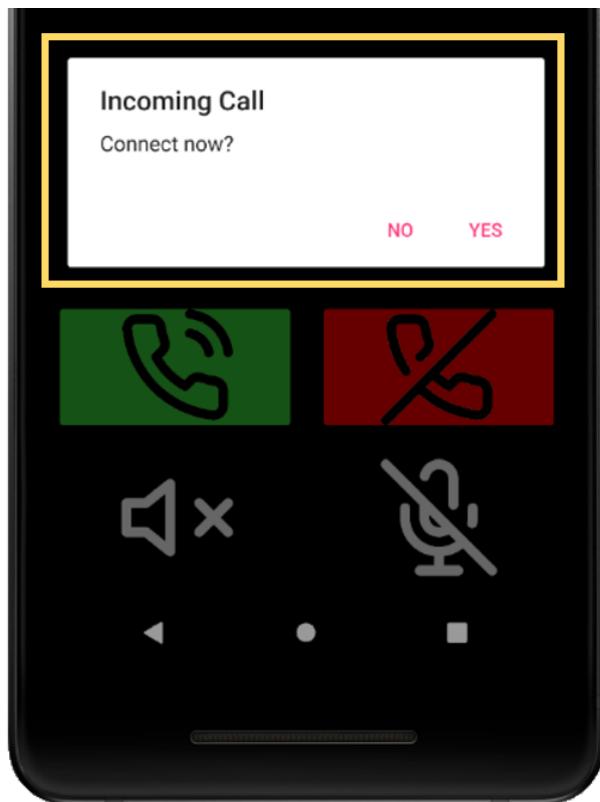


Abbildung 5.4: Abfrage Anrufannahme

Der Wert stellt die Server-Adresse dar, von der der Video-Stream abgerufen werden soll.

51-55: Der Benutzer wird per Popup gefragt, ob der Anruf entgegengenommen werden soll (Abb. 5.4). Wenn die Applikation in den Hintergrund geht, verliert die LibVLC die Referenz auf die VideoView und kann den Video-Stream nicht mehr darstellen. Da her muss die Wiedergabe gestoppt werden, sobald die App in den Hintergrund geht und neu gestartet werden, wenn die App wieder im Vordergrund ist. Da die Benachrichtigung über den Vorder- bzw. Hintergrundstatus der Applikation nur über das native Projekt funktioniert, muss diese Information vom nativen Projekt zum portablen Projekt übertragen werden. Dies geschieht mit der MessagingCenter-Komponente.

```
57 MessagingCenter.Subscribe<string>(this, "OnPause", app =>          57
58 {                                         58
59     _wasConnected = _player.MediaPlayer.isPlaying;           59
60     _player.MediaPlayer.Pause();                           60
61     _prevPosition = _player.MediaPlayer.Position;        61
62     _player.MediaPlayer.Stop();                         62
```

```

63
64     MainGrid.Children.Remove(VideoView);
65 }
66
67 MessagingCenter.Subscribe<string>(this, "OnRestart", app =>
68 {
69     RegenerateVideoView();
70     VideoView.MediaPlayer = _player.MediaPlayer;
71     if (_wasConnected)
72     {
73         _player.MediaPlayer.Play();
74         _player.MediaPlayer.Position = _prevPosition;
75     }
76
77     _prevPosition = 0;
78 });

```

57-65: Dieses Ereignis wird aufgerufen, wenn die Applikation in den Hintergrund wechselt. Bevor dies geschieht, wird die aktuelle Wiedergabe-Position in einer Variablen gespeichert und die Wiedergabe gestoppt. Anschließend wird die VideoView von der Oberfläche entfernt, damit sie später neu hinzugefügt werden kann.

67-78: Das OnRestart-Event wird aufgerufen, wenn die App vom Hintergrund wieder in den Vordergrund geht. Hier geschieht derselbe Vorgang wie davor, jedoch in sinngemäß umgekehrter Reihenfolge. Als erstes wird die VideoView der Oberfläche wieder hinzugefügt, damit die LibVLC eine Referenz dazu bilden kann. Falls davor ein Video gespielt wurde, wird dieses fortgesetzt und die Variable für die vorherige Wiedergabe-position wird gelöscht.

```
84 BindingContext = _player;
```

84: Der BindingContext gibt für das MVVM-Modell an, wo die Variablen, die mit der Oberfläche verknüpft sind, sich befinden. In diesem Fall sind alle Variablen in der Player-Instanz `_player` zu finden.

An folgender Funktion wird das Verhältnis von Oberflächeneignissen mit dem entsprechenden Code-Behind erklärt.

```

109 private void BtConnect_Clicked(object sender, EventArgs e)
110 {
111     _player.StartCall();
112     _streamer.StartCall("192.168.43.78");
113 }

```

109: Eine Funktion für ein Button-Event hat grundsätzlich als Rückgabe-Typ **void**, also nichts. Auf die Funktion kann nur aus der selben Klasse zugegriffen werden, was mit dem **private**-Schlüsselwort bekanntgegeben wird. Als Parameter werden Informationen zum Event-Auslöser und zum Event selbst mitgegeben.

111-112: Wenn der grüne „Anrufen“-Button gedrückt wird, beginnen **_player** und **_streamer** damit, die Daten abzurufen bzw. bereitzustellen. Diese Funktionen werden im Abschnitt 5.2.4 genauer beschrieben.

Im letzten Abschnitt dieser Datei wird das Verhalten beim Wechseln zwischen Hoch- und Querformat festgelegt.

```

121 protected override void OnSizeAllocated(double width, double height)
122 {
123     base.OnSizeAllocated(width, height);
124     if (_width != width || _height != height)
125     {
126         _width = width;
127         _height = height;
128         if (width < height) //Portrait
129         {
130             MainGrid.ColumnDefinitions.Clear();
131             MainGrid.ColumnDefinitions.Add(new ColumnDefinition
132                 {Width = new GridLength(1, GridUnitType.Star)});
133
134             MainGrid.RowDefinitions.Clear();
135             MainGrid.RowDefinitions.Add(new RowDefinition
136                 {Height = new GridLength(0.7, GridUnitType.Star)});
137             MainGrid.RowDefinitions.Add(new RowDefinition
138                 {Height = new GridLength(3, GridUnitType.Star)});
139             MainGrid.RowDefinitions.Add(new RowDefinition
140                 {Height = new GridLength(2, GridUnitType.Star)});

```

```
141  
142     Grid.SetColumnSpan(EditMrl, 1);  
143     Grid.SetColumn(ButtonGrid, 0);  
144     Grid.SetRow(ButtonGrid, 2);  
145 }  
  
[...]  
  
164 }  
165 }
```

121: Die `OnSizeAllocated()`-Funktion ist eine vorgefertigte Funktion, die aufgerufen wird, wenn sich die Abmessungen der Oberfläche geändert haben. Die neuen Abmessungen sind über die Parameter `width` und `height` verfügbar.

124-164: Wenn die Abmessungen anders sind, als die Abmessungen beim letzten Aufruf wird das Oberflächen-Layout entsprechend angepasst. Falls die Höhe größer als die Breite ist, greift der erste Teil der If-Else-Struktur, ansonsten der zweite.

130-144: Zuerst werden die Reihen- und Spalten-Definitionen gelöscht, um sie neu zuweisen zu können. Für das Hochformat werden eine Spalte und drei Reihen benötigt. Diese werden mit den richtigen relativen Maßen hinzugefügt. Wenn dies geschehen ist, werden die Oberflächenelemente den richtigen Plätzen im Grid zugewiesen.

146-163: Das Verändern der Oberfläche für Querformat wird hier ausgelassen, da es sinngemäß äquivalent zum Hochformat ist.

5.2.4 MediaClasses.cs

In dieser Source-Datei sind sämtliche Hilfsklassen zum Abrufen und Bereitstellen von Medieninformationen definiert. Grundsätzlich finden sich hier zwei Klassen:

- Player, die für das Abrufen des Medien-Streams vom Server und Darstellen von diesem zuständig ist
- Streamer, mit der das Zurückschicken der Mikrofon-Rohdaten erfolgen soll

5.2.4.1 Player

Die Player-Klasse implementiert das MVVM-Modell, um die MediaPlayer-Komponente mit der Oberfläche zu verknüpfen. Um die Anzeige über eine Änderung der Variablen zu informieren, muss der PropertyChanged-EventHandler mit dem Variablenamen als Argument aufgerufen werden. Um die Lesbarkeit den Codes zu vereinfachen, wird dieser Aufruf meist in einem Unterprogramm versteckt, welches von Visual Studio automatisch generiert werden kann. Daher wird auf den genauen Aufbau dieses Unterprogrammes nicht genauer eingegangen. Die Initialisierung beinhaltet keine besonderen Funktionen und wird daher ebenfalls nicht behandelt.

```

36 public bool ToggleSpeaker()
37 {
38     MediaPlayer.Volume = _speakerActive ? 0 : 100;
39     return _speakerActive = !_speakerActive;
40 }
```

36
37
38
39
40

38-39: Über die Volume-Eigenschaft des MediaPlayer-Elements kann die Audio-Ausgabe in der Lautstärke verändert oder ganz abgeschaltet werden. Mit der Funktion ToggleSpeaker() wird der aktuelle Zustand der Ausgabe invertiert. Das heißt, wenn die Ausgabe aktiviert ist, wird sie durch einen Aufruf des Unterprogrammes deaktiviert und umgekehrt. Der neue Zustand wird als Wahrheitswert an die aufrufende Funktion zurückgegeben, wobei true bedeutet, dass die Audio-Ausgabe aktiv ist. Als Abkürzung für ein mehrzeiliges If-Else-Statement wird der Ternary-Operator verwendet. Diese Anweisung nimmt abhängig von einem Wahrheitswert zwei verschiedene Werte an. Die Syntax dieser Abkürzung sieht so aus:

```
1 variable = <true/false> ? <Wert wenn true> : <Wert wenn false>;
```

1

Die StartCall()-Funktion wird aufgerufen, wenn der Benutzer auf den grünen Knopf drückt.

```

42 public void StartCall()
43 {
44     MediaPlayer.Media = new Media(LibVlc, Mrl, FromType.FromLocation);
45     MediaPlayer.Play();
46     MediaPlayer.Volume = _speakerActive ? 100 : 0;
47 }
```

42
43
44
45
46
47

44: Bevor die MediaPlayer-Komponente ein Video abrufen kann, muss erst definiert werden, wo dieses Video zum Abruf bereitsteht. Über die Variable `Media` kann die Mediainformation des Ziel-Videos festgelegt werden. Die Variable `Mrl` stellt hier die aktuelle Adresse des Video-Streams dar.

45-46: Mit der `Play()`-Funktion kann anschließend, wie der Name vermuten lässt, die Wiedergabe gestartet werden. Jedoch wird im gleichen Zug die Wiedergabe-Lautstärke zurückgesetzt, weshalb sie nach dem Aufruf der `Play()`-Funktion wieder dem aktuell gewünschten Zustand der Audio-Wiedergabe angepasst werden muss.

```
49 public void EndCall() 49
50 { 50
51     MediaPlayer.Stop(); 51
52 } 52
```

49: Die Wiedergabe des Video-Streams kann am Ende des Gesprächs mit der `Stop()`-Funktion beendet werden.

Die C#-Sprache bietet aufgrund der strengen Objektorientierung das Sprachkonstrukt „Property“ an. Eine Property kann mit einer gewöhnlichen Variable verglichen werden, allerdings bietet sie zusätzlich `get`- und `set`-Funktionen an. Diese Zusatzfunktionen werden aufgerufen, wenn auf den Wert der Property zugegriffen bzw. dieser verändert wird. Dies wird anhand eines Beispiels erläutert:

```
54 private string _mrl; 54
55 public string Mrl 55
56 { 56
57     get => _mrl; 57
58     set 58
59     { 59
60         _mrl = value; 60
61         OnPropertyChanged(); 61
62     } 62
63 }
```

54: Eine Property mit benutzerdefinierten `get`- und `set`-Funktionen benötigt eine zusätzliche Variable, die den eigentlichen Wert der Property speichert. Diese kann auch

dann private und somit nach außen versteckt sein, wenn die Property selbst public, also von außen zugänglich ist.

57: Die get-Funktion ist bei den meisten Properties so definiert, dass der Wert der internen Variablen einfach weitergegeben wird, da beim Auslesen in den meisten Fällen keine zusätzliche Aktion erforderlich ist.

58-62: In der set-Funktion wird hier nicht nur der Wert der internen Variable mit dem neuen Wert überschrieben, sondern auch das System über eine Änderung informiert. Dies ist für die Funktion des MVVM-Modells essenziell.

5.2.4.2 Streamer

Die Streamer-Klasse ist deutlich kürzer, als die Player-Klasse, da die Streamer-Klasse selbst keine wirkliche Funktionalität beinhaltet. Vielmehr dient diese Klasse als Interface zu den plattformspezifischen Softwaremodulen, die für die Audioaufnahme zuständig sind. Dies wird mit Hilfe eines DependencyService erreicht.

```

90  private IAudioRecordingService Service { get; set; }          90
91
92  public Streamer()
93  {
94      Service = DependencyService.Get<IAudioRecordingService>();  94
95 }
```

94: Der DependencyService dient als „Fernsteuerung“, mit dem vom portable Projekt ausgehend Ereignisse und Funktionen im nativen Code aufgerufen werden können.

```

112 public interface IAudioRecordingService           112
113 {
114     void Start(IPEndPoint target);                113
115     void Stop();                                114
116     bool ToggleMic();                           115
117 }
```

112-117: Mit einem interface wird festgelegt, welche Funktionen eine Klasse, die von diesem abgeleitet wird, haben muss. In diesem Fall muss der plattformspezifische Teil der Audioaufnahme die Funktionen Start(), Stop() und ToggleMic() definieren,

um mit dem Interface kompatibel zu sein. Über ein Interface können nicht nur Funktionen, sondern auch Variablen und Properties vorgegeben werden.

5.3 PiBell.Android

Dieses Projekt beinhaltet den gesamten nativen Code für die Android-Plattform. Dazu gehören das Berechtigungs-Management, sowie die Audio-Aufnahme.

5.3.1 MainActivity.cs

Ähnlich wie bei Abschnitt 5.2.1 wird hier das native Android-Projekt initialisiert. Dazu gehört das Laden aller Runtime-Ressourcen und des benötigten Programm-Codes, sowie Überprüfen bzw. Anfragen der Berechtigungen.

```
33 if (CheckSelfPermission(Manifest.Permission.RecordAudio) !=           33
     → Permission.Granted)
34 {
35     RequestPermissions(new[] { Manifest.Permission.RecordAudio }, 1);   34
36 }
```

33-36: Es wird überprüft, ob die Berechtigung für die Audio-Aufnahme bereits erteilt ist. Wenn das nicht der Fall ist, wird diese beim System angefragt. Der Benutzer erhält dann ein PopUp, mit dem er gefragt wird, ob die Berechtigung erteilt werden soll. Der Benutzer kann anschließen entweder, wie in Abbildung 5.5 gezeigt, die Berechtigung erteilen oder verweigern. In letzterem Fall kann die Applikation nicht richtig funktionieren.

```
39 protected override void OnPause()                                39
40 {                                                               40
41     base.OnPause();                                         41
42     MessagingCenter.Send("app", "OnPause");                  42
43 }                                                               43
44                                                               44
45 protected override void OnRestart()                            45
46 {                                                               46
47     base.OnRestart();                                         47
48     MessagingCenter.Send("app", "OnRestart");                48
49 }                                                               49
```

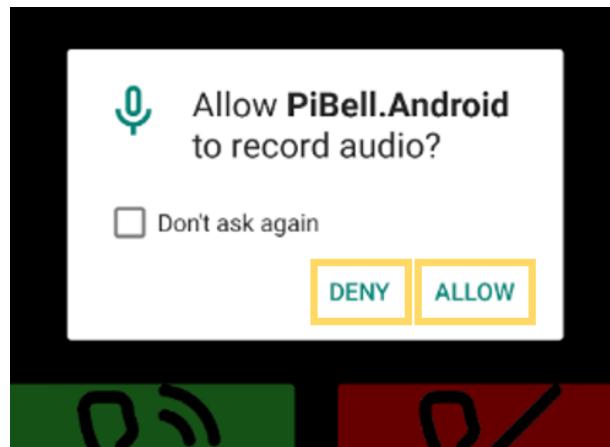


Abbildung 5.5: Anfrage für Berechtigung

39-49: Das in Abschnitt 5.2.3 beschriebene Verhalten beim Wechsel der App von und in den Hintergrund benötigt ein vom nativen Projekt ausgelöstes Ereignis. Über das MessagingCenter wird eine Nachricht an das portable Projekt geschickt, damit dieses den Zustandswechsel mitbekommt und darauf reagieren kann.

6 Software-Ausblick

6.1 Gesicherte Verbindung

Die im Zuge dieser Arbeit entwickelte Applikation verschickt bzw. empfängt alle Daten-Streams unverschlüsselt und ohne Benutzerverifizierung. Bevor die Applikation im Feld verwendet werden kann, sollte eine gesicherte Verbindung zum Server konfiguriert werden. Dies kann zum Beispiel per Virtual Private Network (VPN) erfolgen. Hier wird ein verschlüsselter Tunnel zu einem gemeinsamen Server aufgebaut, in dem alle Daten verschickt werden. Dieses neu erzeugte Netzwerk ist virtuell, d.h. es existiert nur in der Softwarewelt und benötigt keine zusätzliche physikalische Verbindung.

Man könnte entweder auf kommerzielle VPN-Anbieter zurückgreifen oder mit Open-VPN und einem eigenen Server eine gesicherte Verbindung ermöglichen. Im letzteren Fall kann auch der Medienserver die Rolle des VPN-Servers übernehmen.

6.2 Weitere mobile Plattformen

Neben dem ausprogrammierten Android-System gibt es noch das weit verbreitete iOS der Apple-Geräte. Mit Xamarin.Forms lässt sich dieses verglichen mit anderen Entwicklungsmethoden einfach unterstützen. Es muss lediglich der plattformspezifische Teil auf die neue Plattform portiert werden, während das portable Projekt unverändert bleibt. Dies bedeutet, dass analog zum PiBell.Android ein Projekt PiBell.iOS geschrieben werden muss, um die Plattform zu unterstützen. In diesem Projekt müssen alle Funktionen des Android-Teils repliziert werden. Wichtig für die Entwicklung ist auch, dass die Endgeräte zum Übertragen und Testen verfügbar sind. Im Fall der iOS-Plattform wird ein Apple Mac für die Übertragung und ein Apple iPhone zum Testen benötigt.

Für Testzwecke kann die Applikation auf bis zu fünf iPhones installiert und getestet werden. Wenn die App schlussendlich veröffentlicht werden soll, ist eine kostenpflichtige Mitgliedschaft beim Apple Developer Program notwendig. Diese beläuft sich auf etwa USD 99 pro Jahr [vgl. 14, Integrated Development Environment Availability]

Teil II

Hardware - Matthias Mair

7 Ausgangssituation

7.1 Hardware-Konfiguration

Die aktuelle Hardware der Station setzt sich aus mehreren Einzel-Platinen mit unterschiedlichen Betriebsspannungen zusammen:

- Mikrofon-Vorverstärker-Platine, die das Signal des Kondensator-Mikrofons verstärkt, bevor es zum RPi weitergeleitet wird. Diese Platine wird mit einer Spannung von 5 V betrieben.
- Lautsprecher-Verstärker-Platine, die für die Verstärkung der wiederzugebenden Audiosignale des RPi zuständig ist. Die verstärkten Signale werden direkt an den Lautsprecher der Station übergeben. Diese Platine benötigt eine Versorgungsspannung von 24 V.

Die elektrische Versorgung der Station erfolgt mittels Power over Ethernet (PoE) bei einer Speisespannung von etwa 30 V. Alle Platinen sind mit Heißkleber in der Station montiert.

7.2 Problemdefinition

Systemstabilität: In der aktuellen Konfiguration kommt es immer wieder zu unvorhersehbaren und unkontrollierten Systemabstürzen. Die betroffene Station reagiert weder auf eingehende Signale noch erfolgt irgendeine Form der Informationsausgabe. Dieser Zustand lässt sich nur durch ein Zurücksetzen des Prozessors beenden. Die Abstürze sind bei der am weitesten von der Spannungsversorgung entfernten Station am häufigsten zu beobachten. Dieser Fehler tritt in unregelmäßigen Abständen von bis zu mehreren Monaten auf. Die betroffene Station weist keinen Hardware-Unterschied zu den anderen Stationen auf. Daher lässt sich vermuten, dass die Länge und Art der Verkabelung mit diesem Phänomen in Zusammenhang steht.

Komplexer Aufbau: Die aktuelle Hardware-Konfiguration verwendet einige Einzelplatinen, die miteinander über eine fliegende Verdrahtung verbunden sind. Dies reduziert die Übersichtlichkeit des Stationsinneren und birgt die Gefahr von fehlerhafter Ver-

drahtung in sich. Durch die fliegende Verdrahtung ist kein einheitliches Erscheinungsbild des Stationsinneren gewährleistet. Das Aussehen des Stationsinneren wird in Abbildung 7.1 festgehalten.

Brumm-Schleife: Die aktuelle Verdrahtung führt zu einer Masseschleife. Diese erzeugt ein konstantes Störgeräusch. Über die Verstärkerschaltung führt dies zu einem unerträglichen Brumm-Ton. In der aktuellen Hardware-Konfiguration wird das Problem mit einer Masse trennung nach dem Ausgang des RPi umgangen. Dieser zusätzliche Baustein wurde fliegend verdrahtet und mit Heißkleber befestigt.

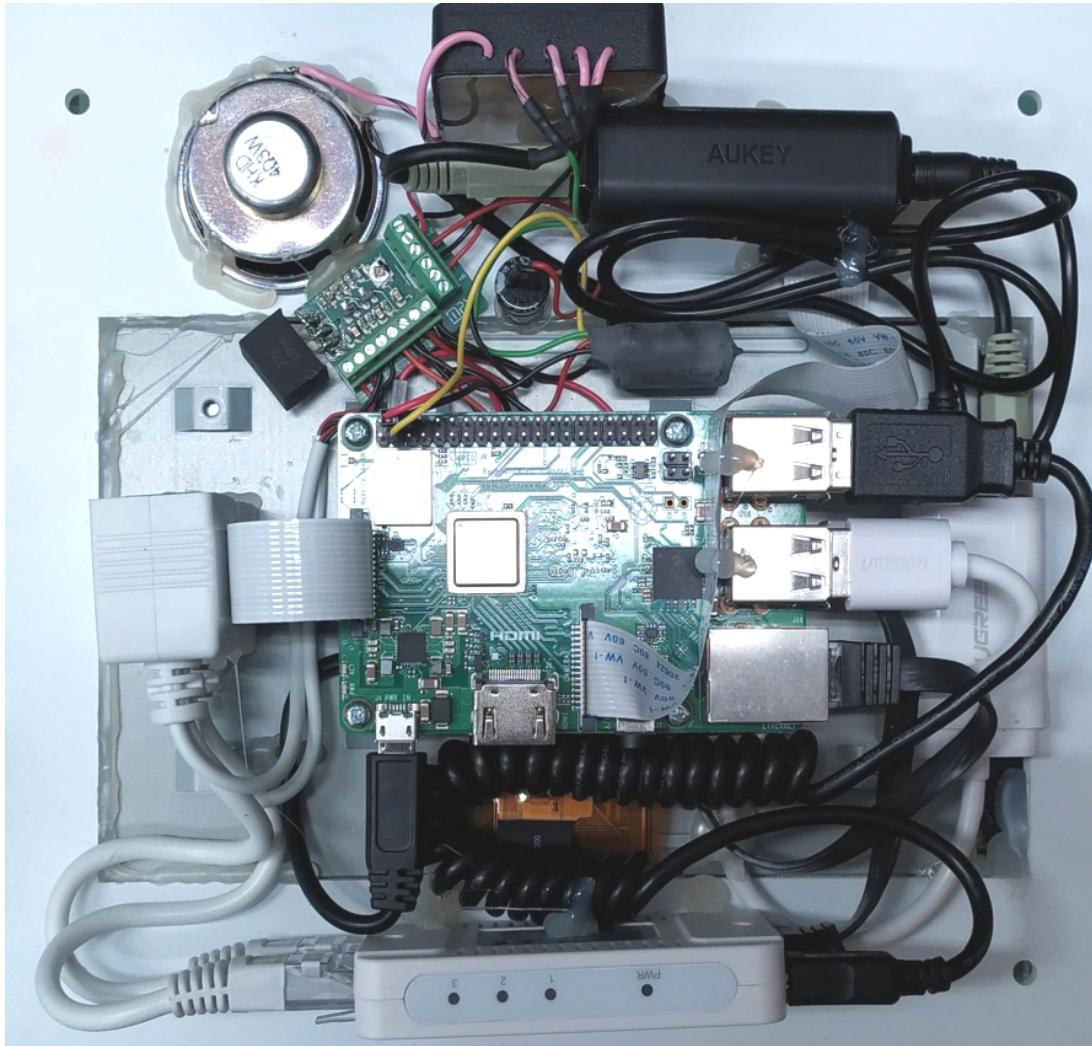


Abbildung 7.1: Unübersichtlicher Aufbau

8 Technische Grundlagen

8.1 Power over Ethernet

Bei Power over Ethernet wird die Energieversorgung über eine herkömmliche Cat5-Ethernetleitung mitgeführt. Dadurch wird eine weitere Versorgungsleitung eingespart. Eine Ethernet Leitung besteht aus acht Adern, wobei je zwei Adern zu einem Adernpaar verdrillt sind. Je nach Geschwindigkeit der Datenübertragung werden nur vier oder alle acht Adern für die Datenübertragung verwendet. Das bedeutet, bei geringeren Übertragungsraten können die freien Adern zur Versorgung genutzt werden. Wenn alle acht zur Datenübermittlung verwendet werden, müssen vier Adern für Datenübertragung und Energieversorgung genutzt werden.

Grundsätzlich wird zwischen vier verschiedenen Typen unterschieden:

Type 1, PoE: Der Standard IEEE 802.3af gilt nur für 10 Mbit/s und 100 Mbit/s. Das bedeutet, dass nur die Adernpaare 1/2 und 3/6 für die Datenübertragung genutzt werden und die beiden anderen Adernpaare unbenutzt sind. Je Adernpaar ist ein Strom von maximal 175 mA vorgesehen. Bei zwei Adernpaaren ist das in Summe ein Strom von 350 mA. Beim Einschalten sind kurzzeitig 400 mA erlaubt. Die maximale Leistungsaufnahme beträgt 15,4 Watt pro Switch-Port.

Type 2, Power over Ethernet Plus (PoE+): Mit IEEE 802.3at eignet sich Power-over-Ethernet auch für 1000 Mbit/s (Gigabit). Gleichzeitig wird die Leistung fast verdoppelt. IEEE 802.3at verspricht eine Leistung bis 25,5 Watt pro Port. Dabei wird die Minimalspannung von 44 auf 50 Volt erhöht. Der maximale Strom wurde von 350 mA auf 600 mA erhöht. Bei diesen hohen Leistungen wird ein Cat5e/6-Kabel empfohlen.

Type 3 & 4, Four Pair Power over Ethernet (4PPoE): Den Standard IEEE 802.3bt bezeichnet man als Four-Pair-Power-over-Ethernet. Die Kurzschreibweise 4PPoE oder PoE++. Bisher nutzt Power-over-Ethernet nur zwei der vier Aderpaare eines Twisted-Pair-Kabels. Mit 4PPoE werden alle Adern der vorhandenen Kabel zur Leistungsübertragung verwendet. Damit steigt die Leistung auf 70 bis 100 Watt.

Aus den verschiedenen Typen ergeben sich zwei Arten der Energiespeisung:

Spare-Pairs-Verfahren: Das Spare-Pairs-Verfahren verwendet die beiden unbenutzten Adernpaare im Kabel (4/5 und 7/8) für die Stromversorgung. Die Übertragung von Strom und Daten sind voneinander getrennt. Diese Art kann nur bei Type 1 und Type 2 verwendet werden.

Phantom-Speisung: Bei der Phantom-Speisung werden die datenführenden Adern im Kabel verwendet. Phantom-Speisung bedeutet, dass der Strom für die Energieversorgung dem Datensignal überlagert wird. Das Power-Device muss die Entkopplung übernehmen, was fehleranfällig, aufwendig und teuer ist. Wenn alle Adernpaare des Kabels für die Datenübertragung verwendet werden, d.h. es wird Type 3 oder Type 4 angewendet, dann ist man zwangsläufig auf die Phantom-Speisung angewiesen. Hierbei ist der Strom pro Adernpaar und somit die Gesamtleistung begrenzt.

Type	PoE	PoE+	4PPoE
Ausgangsspannung	36-57 V	42.5-57 V	42.5-57 V
Ausgangstrom Betrieb	350 mA	600 mA	2x 960 mA
Ausgangstrom Startmodus	400 mA	400 mA	?*
Leistung der (PSE)-Versorgung	max. 15.4 W	max. 30 W	45/60/75/90 W
Leistung am Endgerät (PD)	max. 12.95 W	max. 25.5 W	40/51/62/71 W
PSE-Klasse	1, 2, 3	4	5, 6, 7, 8
unterstützte Endgeräte (PD-Type)	1	1 und 2	1, 2, 3, 4
Benutzte Adernpaare	2	2	2 und 4

*in Fachartikel *Schlüsseltechnologie Power over Ethernet (PoE)* nicht behandelt

Tabelle 8.1: Vergleich PoE-Typen [vgl. 15]

8.2 Spannungsregler

Ein Spannungsregler stabilisiert eine elektrische Spannung. Er wird eingesetzt, wenn eine konstante Betriebsspannung benötigt wird. Grundsätzlich arbeitet ein Spannungsregler mit einem Leistungstransistor, der so angesteuert wird, dass die gewünschte Ausgangsspannung zu Stande kommt.

Es gibt zwei Arten von Spannungsregler:

Linearregler: Beim Linearregler wird der Leistungstransistor als veränderbarer Widerstand genutzt. Über einen Regelkreis wird eine Veränderung der Ausgangsspan-

nung, z.B. Laständerung, festgestellt. Daraufhin ändert der Transistor seinen Innenwiderstand, um die Veränderung auszugleichen.

Schaltregler: Der Schaltregler hingegen nutzt den Leistungstransistor als Schalter. Er schaltet mit einer Frequenz von mehreren kHz bis MHz und erzeugt somit ein Pulse Width Modulation (PWM) Signal. Dieses Signal wird anschließend mit einer Induktivität geglättet, um einen konstanten Strom zu erzielen. Mit der Pulsbreite wird der Effektivwert des PWM-Signals und damit die Höhe der geglätteten Spannung variiert.

8.3 Surface Mounting Technology

Bei der klassischen THT-Technologie werden die Bauteilanschlüsse durch Löcher in der Platine gesteckt und mit kreisförmigen Lötpads auf der gegenüberliegenden Seite verlötet. Im Gegensatz dazu sind bei SMD-Platinen Bauteile und Lötpads auf derselben Platinenseite, weshalb keine Bohrungen mehr erforderlich sind. Die Bauteilanschlüsse werden flach mit dem rechteckigen Lötpad verbunden. Die Verlötzung findet mithilfe von Heißluft statt. Die fertig bestückten Platinen werden in einen Reflow-Ofen gelegt und auf die Schmelztemperatur der Lötpaste erhitzt.

8.3.1 Entwicklung

In den 1960ern wurde die Oberflächenmontagetechnik (Surface Mounting Technology (SMT)) von IBM entwickelt und fand ihre ersten Anwendungen in den Saturn- und Apollo-Missionen. In den 1970ern wurde die SMT erstmals genormt, wodurch sich die Technologie im kommerziellen Sektor verbreiten konnte. Die SMT bietet gegenüber der herkömmlichen THT einige Vorteile:

Vorteile:

- Miniaturisierung, aufgrund der geringeren Größe der Bauteile können Platinen und somit auch Endgeräte kleiner und kompakter gebaut werden.
- Auch das Gewicht wird durch die Verkleinerung der Bauelemente stark reduziert.
- Der Bauteilabstand kann verkleinert werden, dadurch verbessert sich der Platzverbrauch weiter.
- Die Hochfrequenzeigenschaft wird durch die kürzeren Übertragungswege zwischen den Bauteilen verbessert.

- Die Fertigung der Platinen kann mit SMD besser automatisiert werden, da die Positionierung der Bauelemente nicht so genau sein muss, wie bei THT. Die Bau- teile werden beim Verlöten durch die Oberflächenspannung des Lötzinns in die korrekte Position gezogen.
- Einfache doppelseitige Bestückung von Platinen, da kein Platz von Anschlüssen von Bauteilen auf der anderen Platinenseite verbraucht wird.

Nachteile:

- Anschlüsse an der Unterseite von Bauteilen können nur durch Röntgen überprüft werden.
- Das Heißluftlötverfahren erhitzt nicht nur die Lötstelle, sondern das gesamte Bau- teil. Dadurch besteht ein höheres Risiko, dass dieses beim Anlöten zerstört wird.
- Große Bauelemente benötigen zusätzliche Fixierung, da die Lötstelle der mechanischen Belastung nicht standhält. Aufgrund der vielen Vorteile gegenüber THT fand die SMT in der Digital- und Computertechnik anfänglich häufige Anwendung. Anfang der 1980er begannen die ersten Unternehmen mit der automatischen Fer- tigung von SMD-Platinen, wodurch die Herstellungskosten weiter gesenkt wur- den.

8.3.2 Bauteilcodes

Grundsätzlich werden alle elektronischen Bauteile in die beiden Untergruppen aktive und passive Bauelemente unterteilt.

Aktive Bauelemente können Signale verstärken oder erzeugen. Die benötigte En- ergie kommt aus einer zusätzlichen Versorgung durch elektrische oder einer anderen Energieform. Beispiele für aktive Bauteile sind Transistoren, ICs, Photovoltaikzellen, ...

Passive Bauelemente können im Gegensatz zu aktiven Bauteilen selbst keine Si- gnale erzeugen oder verstärken. Sie benötigen keine zusätzliche Energieversorgung, um zu funktionieren. Beispiele für diese Art sind Widerstände, Kondensatoren und Spu- len.

8.3.2.1 Passive Bauteile

Für passive Bauelemente gibt es vier wichtige Bezeichnungen:

Chip: Dies ist die Standard Bauform für unter anderem Keramik- und Tantal-Kondensatoren, Induktivitäten sowie nichtlineare und lineare Widerstände. Ein Chip ist ein quaderförmiges Bauteil, das auf der Platine aufliegt. In die Bezeichnung fließt nur die bauliche Größe ein. Die Größe wird mit einem metrischen Code angegeben, z.B. „0603“, die Ziffer „06“ steht für die Länge und „03“ für die Breite. In diesem Beispiel bedeutet das, dass das Bauteil 0,6 mm lang und 0,3 mm breit ist.

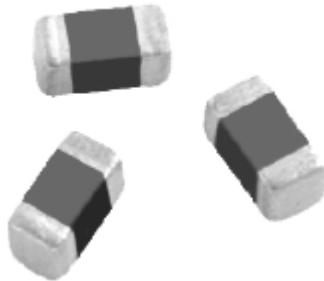


Abbildung 8.1: Chip-Bauform [vgl. 16]

Metal Electrode Faces (MELF): Die MELF Bauform ist im Gegensatz zur Chip-Bauform zylinderförmig, liegt aber ebenfalls flach auf der Platine auf. Diese Bauform ist deutlich größer als die Chip Variante, dafür bietet sie elektrische Vorteile. Die MELF Bauteile haben eine bessere Spannungsfestigkeit, höhere Impulsstrombelastbarkeit und eine bessere Temperaturstabilität. Im Fehlerfall sind in dieser Bauweise die Widerstände so ausgelegt, dass sie in jedem Fall hochohmig werden und dadurch Schäden an anderen Bauteilen verhindern können. Beim Lötvorgang hingegen kann es bei MELF Bauteilen dazu kommen, dass sie wegrollen. Daher werden sie meist vor dem Lötvorgang festgeklebt.

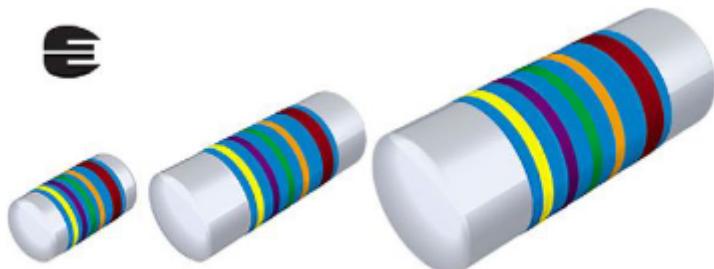


Abbildung 8.2: MELF-Bauform [vgl. 17]

Small Outline Diode (SOD): Wie der Name schon sagt ist das eine Gehäuseform, welche ausschließlich für Dioden verwendet wird. Die Bauteile sind entweder wie bei MELF zylindrisch (z.B. SOD-80 oder MiniMELF) oder quaderförmig mit Anschlussfahnen (z.B. SOD-123) ausgeführt.



Abbildung 8.3: SOD-123-Bauform [vgl. 18]

Vertical Chip (V-Chip): Ein V-Chip ist ebenfalls zylindrisch, jedoch werden diese Bauteile stehend montiert. Diese Bauform wird häufig bei Aluminium-Elektrolytkondensatoren verwendet und können dabei große Abmessungen erreichen.



Abbildung 8.4: V-Chip-Bauform [vgl. 19]

8.3.2.2 Aktive Bauteile

Für aktive Bauelemente (ICs) gibt es wesentlich mehr Bauformen und Bezeichnungen. Daher werden hier nur einige wichtige aufgeführt.

Small Outline Transistor (SOT): Diese Bauform wird ausschließlich für Transistoren verwendet. Sie hat drei oder vier Anschlüsse, wobei der vierte nur zum Abführen entstandener Wärme dient. Das Bauteil ist in dieser Bauform in einem quaderförmigen Kunststoff-Gehäuse verschweißt.

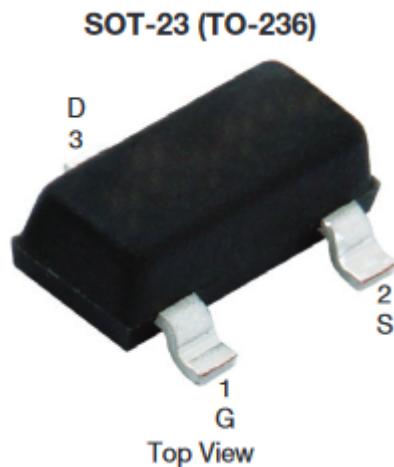


Abbildung 8.5: SOT-23-Bauform [vgl. 20]

Small Outline Integrated Circuit (SOIC): Bauteile in dieser Ausführung ähneln einem herkömmlichen Dual In-Line Package (DIP)-Gehäuse, da der Anschlussabstand bei beiden Formen gleich groß ist. Dieser Gehäusetyp wird vor allem für ICs verwendet.

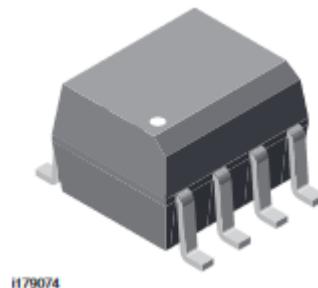


Abbildung 8.6: SOIC-Bauform [vgl. 21]

Small Outline Package (SOP): Dies ist eine kleinere Form der SOIC-Bauform und wird ebenso hauptsächlich für ICs verwendet. Diese Ausführung bekommt je nach Baugröße einen oder mehrere zusätzliche Buchstaben vorangestellt.

8.3.3 Mögliche Fertigungsfehler

8.3.3.1 Grabstein-Effekt:

Bei dem sogenannten Grabsteineffekt handelt es sich um einen Fehler, der bei der Verlötzung der Bauteile mit der Platine auftreten kann. Ein Bauteil mit zwei Anschlüssen wird auf einer Seite angelötet. Aufgrund der Oberflächenspannung des Lötzinns hebt



Abbildung 8.7: SOP-Bauform [vgl. 22]

sich das Bauelement auf einer Seite der Platine ab. Diese Position lässt das Bauteil wie einen kleinen Grabstein erscheinen.

Um diesen Effekt entgegenwirken zu können, wird darauf geachtet, dass beide Lötstellen zeitgleich erhitzt werden.

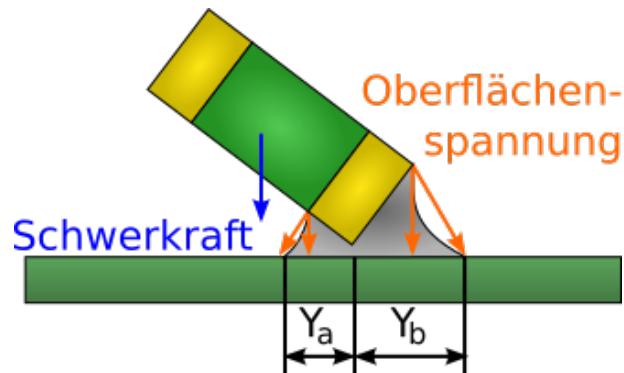


Abbildung 8.8: Grabstein-Effekt [23]

8.3.3.2 Popcorn-Effekt:

Ein feuchteempfindliches Bauteil, wie zum Beispiel eine Rohplatine oder das Trägermaterial eines Prozessors, welches außerhalb der vor Feuchte schützenden Verpackung gelagert wird, neigt zum Popcorn Effekt. Das Bauteil nimmt die Feuchtigkeit aus der Umgebung auf und speichert diese. Beim Löten verdampft das Wasser und bläht das Bauteil aufgrund der Ausdehnung auf. Das Ausdehnen führt u. a. zu Rissen im Gehäuse oder Delaminierung der Kupferschicht.

Diese Fehler führen meist zur Zerstörung des Bauteils und können erst nach der Fertigung diagnostiziert werden. Vorbeugend ist auf eine korrekte Lagerung der Komponenten zu achten.

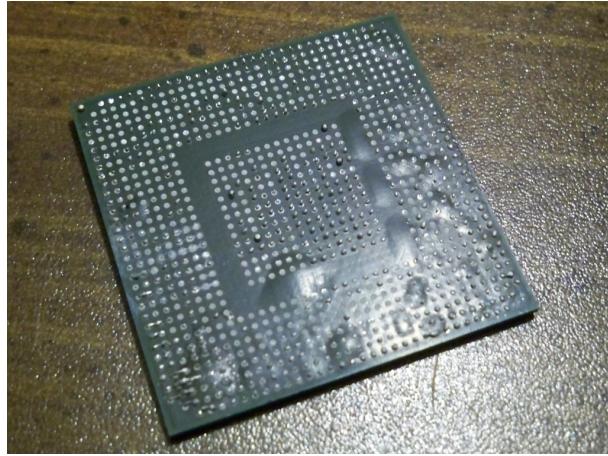


Abbildung 8.9: Popcorn-Effekt [24]

8.4 Serielle Schnittstellen

8.4.1 Universal Asynchronous Receiver/Transmitter

Universal Asynchronous Receiver Transmitter (UART) ist ein universell einsetzbarer elektronischer Baustein, der Systemeinheiten mit parallelem Übertragungsmodus an serielle Übertragungswege anpasst. Mit dem UART werden in Personal Computern (PC) die parallel anfallenden Daten in serielle Datenströme umgewandelt, die dann an den seriellen Schnittstellen (COM, LPT, IDE, usw.) für die angeschlossenen Peripheriegeräte und für die Datenübertragung über Modems zur Verfügung stehen. [25]

Der UART-Datensatz besteht aus 11 Bit, einem Startbit, acht Datenbits, einem Paritätsbit und einem Stoppsbit. Der UART selbst besteht aus zwei First In First Out (FIFO)-Puffern, die sende- und empfangsseitig die ein- und ausgehenden Datensignale zwischenspeichern. Je nach Breite des Datenbusses gibt es Wandler für 8-Bit und 16-Bit mit Übertragungsraten von 19,2 kbit/s, 56 kbit/s (UART16450) und 128 kbit/s (UART16650D). [25]

UART wurde für diese Anwendung ausgewählt, wegen der Einfachheit der Verdrahtung. Die Kommunikation der Geräte findet über nur zwei Datenleitungen statt, dadurch werden Verbindungen zwischen der Platine und dem RPi eingespart. UART wird vom RPi und Mikrocontroller unterstützt.

8.4.2 Serial Peripheral Interface

Das Serial Peripheral Interface, kurz Serial Peripheral Interface (SPI) oder auch Micro Wire genannt, ist ein Bussystem bestehend aus drei Leitungen für eine serielle synchro-

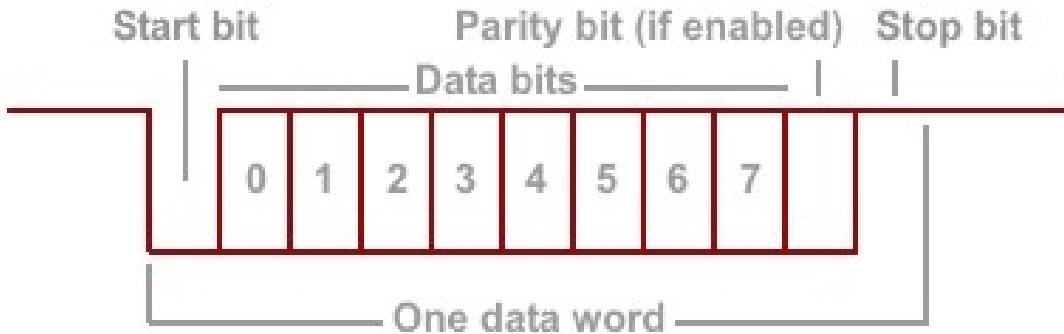


Abbildung 8.10: Aufbau eines UART-Frames [26]

ne Datenübertragung zwischen verschiedenen ICs. [27]

SPI-Bausteine kommunizieren im Vollduplex-Modus. Das System verwendet eine Master-Slave-Architektur mit einem einzigen Master. Der Bus besteht aus folgenden Leitungen

- Master Out Slave In (MOSI)
- Master In Slave Out (MISO)
- Serial Clock (SCK) - Schiebetakt

Zusätzlich zu diesen drei Leitungen wird für jeden Slave eine Slave Select (SS) genannte Leitung benötigt, durch die der Master den Slave zur aktuellen Kommunikation selektiert. Dies geschieht dadurch, dass der Master die SS-Leitung von High nach Low zieht. Oft ist mit dieser Aktivierung durch den Master auch eine Benachrichtigung für den Slave verbunden, mit der ihm mitgeteilt wird, dass jetzt eine Nachricht beginnt, das nächste Byte also zum Beispiel als Kommando aufzufassen ist. [27]

Die Übertragung geschieht so, dass der Master seine Datenleitung (MOSI) auf den Pegel des nächsten Bits bringt und dann an der SCK Leitung einen Puls ausgibt. Gleichzeitig wird vom Master der Pegel an der Datenleitung vom Slave zum Master überwacht und ihr Zustand als nächstes einzulesendes Bit aufgefasst. Üblicherweise gibt es zumindest beim Master mehrere Einstellungen, die festlegen, welches der Grundzustand dieser SCK Leitung sein soll und welche Flanke des Taktes zur Datenübernahme herzunehmen ist (die steigende oder die fallende). Bei einigen Slaves ist diese Einstellung ebenfalls möglich, oft ist es aber so, dass per SPI anzusprechende IC eine feste Einstellung benutzen, an die sich der Master anpassen muss. [27]

Für den SPI-Bus gibt es kein festgelegtes Protokoll. Die Taktpolarität und Phase können ebenfalls von Slave zu Slave unterschiedlich sein. Der SPI-Bus kann mit einer Taktfrequenz von vielen Megahertz betrieben werden. Es gibt viele verschiedene ICs, die als Slave an dem SPI-Bus betrieben werden können. Diese gehen von einfachen Schieberegistern bis hin zu Real-Time Clocks (RTCs) oder Displaytreibern mit vorgegebenem Protokoll. Unter anderem werden die meisten AVR-Microcontroller von Atmel über SPI programmiert. [27]

8.5 Audio-Verstärker Class D

Mit Class-D wird ein bestimmtes Schaltungsdesign von Audio-Verstärkern bezeichnet, die ein PWM-Signal erzeugen und daher auch als PWM-Verstärker bekannt sind. [28]

Bei einem herkömmlichen Transistorverstärker enthält die Ausgangsstufe Transistoren, die den momentanen kontinuierlichen Ausgangstrom liefern. Zu den vielen möglichen Implementierungen für Audiosysteme gehören die Klassen A, AB und B. Im Vergleich zu Designs der Klasse D ist die Verlustleistung der Ausgangsstufe selbst in den effizientesten linearen Ausgangsstufen groß. Dieser Unterschied verleiht der Klasse D in vielen Anwendungen erhebliche Vorteile, da die geringere Verlustleistung weniger Wärme erzeugt, Platz und Kosten auf der Leiterplatte spart und die Batterielebensdauer in tragbaren Systemen verlängert.

Ein symmetrisch arbeitender Dreiecksgenerator schwingt mit einer typischen Frequenz von ca. 250 kHz bis zu einigen MHz, dessen Pegel von einem Komparator mit dem Pegel des zu verstärkenden Eingangssignals verglichen wird. Der Komparator verwandelt das analoge Tonsignal in eine Rechteckschwingung, wie in dem Blockdiagramm zu erkennen ist.

Ist das Dreiecksignal größer als das Tonsignal, springt der Ausgang auf low, ist es kleiner, springt er auf high. Die maximale Impulsbreite ist dabei kleiner als die Zykluszeit der Arbeitsfrequenz und kann somit nie länger als ein Taktzyklus ein- oder ausgeschaltet (high oder low) sein. Das Tonsignal liegt nun im Tastverhältnis des PWM-Signals vor. Der Mittelwert ist dadurch exakt proportional zum Tonsignal.

Dieses PWM-Signal wird der Endstufe zugeführt, in welcher die eigentliche Verstärkung stattfindet, bestehend aus zwei Leistungstransistoren im Schaltbetrieb für je eine positive und eine negative Halbwelle. Ein Class-D-Verstärker ist entweder als Halbbrücke mit zwei Transistoren aufgebaut oder mit vier Transistoren als Vollbrücke (H-Brücke) ausgeführt.

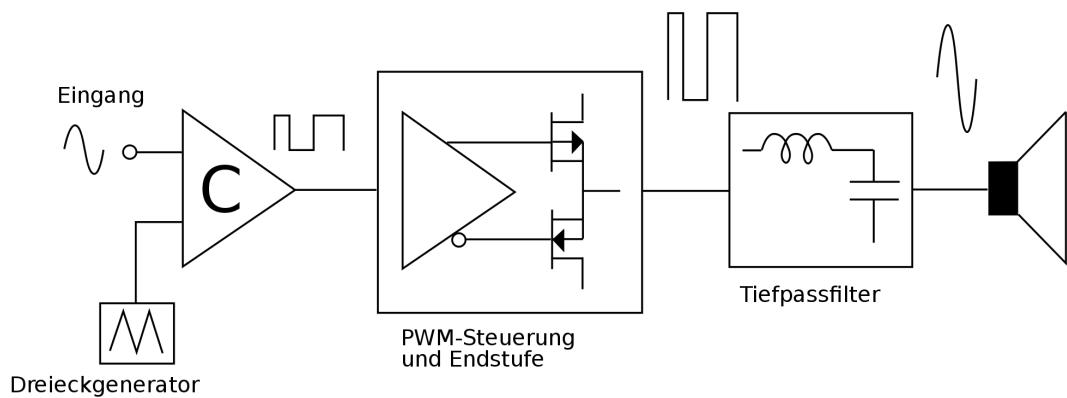


Abbildung 8.11: Schematische Darstellung Class-D [29]

9 Berechnung

9.1 Spannungsabfall

Herkömmliche Cat5-Ethernetkabel wurden ursprünglich nicht für die Übertragung großer elektrischer Leistungen entwickelt und besitzen daher Leitungen mit einem sehr geringen Ader-Durchmesser. Aufgrund dieser Gegebenheit kann es bei weiten Entfernung oder großen übertragenen Leistungen zu abnormal hohen Spannungsabfällen kommen. Um die Anlage in dieser Hinsicht zu überprüfen, werden sämtliche Spannungsabfälle und die damit verbundenen Verlustleistungen bei maximaler Belastung durch die Stationen berechnet. Zur Berechnung der Spannungsabfälle sind folgende Angaben verwendet worden.

- Es werden vier Stationen über PoE versorgt. Die Stationen sind nach Abbildung 9.1 miteinander verdrahtet.
- Jede Station hat einen maximalen Leistungsverbrauch von 12 W.
- Das Ethernetkabel zwischen den Stationen hat jeweils eine Länge von 10 m.
- Der Querschnitt einer Ader des Ethernet-Kabels beträgt 0.128 mm^2 [vgl. 30].
- Am PSE wird eine Spannung von 30 V in das Netz eingespeist.

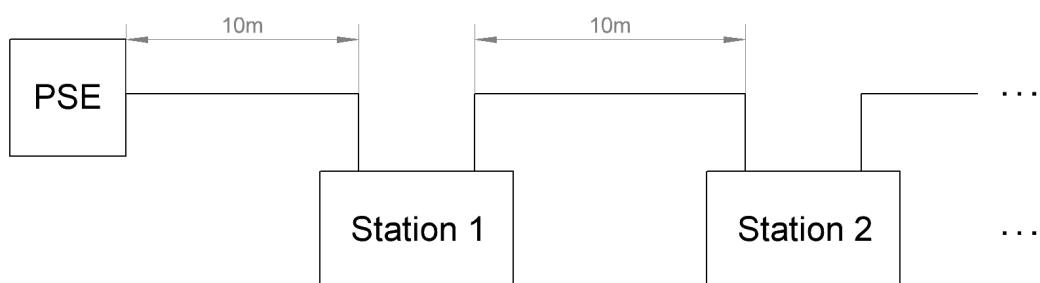


Abbildung 9.1: Verdrahtungsschema

Der Leitungswiderstand eines Adernpaars des Ethernetkabels beträgt nach der allgemeinen Drahtwiderstandsformel $R_{Ltg} = \frac{l}{\gamma \cdot A} = 0.686 \Omega$. Bei den PoE-Typen 1 und 2 wird

je ein Adernpaar für die Zu- und Rückleitung verwendet. Bei den PoE-Typen 3 und 4 werden je zwei Adernpaare parallel geschaltet.

Daraus ergeben sich die in den Abbildungen 9.2 und 9.3 dargestellten Ersatzschaltungen. Die Stationen (Verbraucher) sind als Widerstand gezeichnet, wobei sie sich nicht wie ein solcher verhalten.

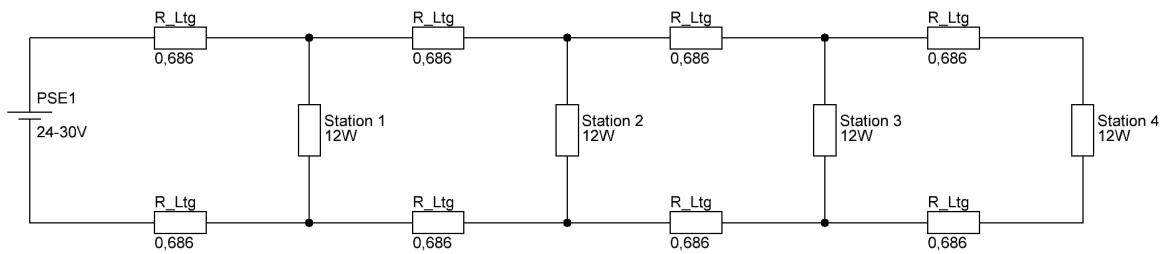


Abbildung 9.2: Ersatzschaltung PoE Type 1 und 2

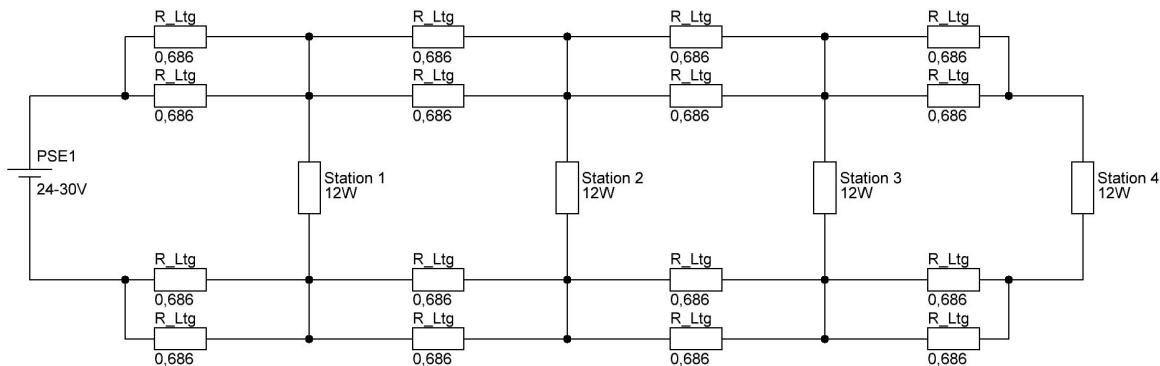


Abbildung 9.3: Ersatzschaltung PoE Type 3 und 4

Anhand der in den Abbildungen 9.2 dargestellten Ersatzschaltung ergibt sich ein Gleichungssystem in 8 Variablen. Für die in Abbildung 9.3 dargestellte Ersatzschaltung halbiert sich der Leitungswiderstand, wodurch der Faktor 2 vor diesem in den Gleichungen verschwindet. Die Quellspannung ist mit U_0 bezeichnet, der Strom und die

Spannung der Station n mit I_n bzw. U_n .

$$U_1 = U_0 - 2 \cdot R_{Ltg} \cdot (I_1 + I_2 + I_3 + I_4) \quad (9.1)$$

$$U_2 = U_1 - 2 \cdot R_{Ltg} \cdot (I_2 + I_3 + I_4) \quad (9.2)$$

$$U_3 = U_2 - 2 \cdot R_{Ltg} \cdot (I_3 + I_4) \quad (9.3)$$

$$U_4 = U_3 - 2 \cdot R_{Ltg} \cdot I_4 \quad (9.4)$$

$$12W = U_1 \cdot I_1 \quad (9.5)$$

$$12W = U_2 \cdot I_2 \quad (9.6)$$

$$12W = U_3 \cdot I_3 \quad (9.7)$$

$$12W = U_4 \cdot I_4 \quad (9.8)$$

Dieses Gleichungssystem von Hand zu lösen, würde viel Zeit in Anspruch nehmen. Daher wurde für die Lösung auf das Computerprogramm wxMaxima zurückgegriffen. Die Eingabe der Gleichungen erfolgt in Textform (siehe Abbildung 9.4)

wxMaxima ist eine dokumentenbasierte Schnittstelle für das Computeralgebra system Maxima. wxMaxima bietet Menüs und Dialoge für viele gängige Maxima-Befehle, Auto vervollständigung, Inline-Plots und einfache Animationen. wxMaxima wird unter der General Public License (GPL) vertrieben. [31, aus dem Englischen übersetzt]

```
(%i16) u_0: 30$  
r_ltg: 0.686$  
p: 12$  
eqn1: u_1=u_0-2·r_ltg·(i_1+i_2+i_3+i_4)$  
eqn2: u_2=u_1-2·r_ltg·(i_2+i_3+i_4)$  
eqn3: u_3=u_2-2·r_ltg·(i_3+i_4)$  
eqn4: u_4=u_3-2·r_ltg·i_4$  
eqn5: p=u_1·i_1$  
eqn6: p=u_2·i_2$  
eqn7: p=u_3·i_3$  
eqn8: p=u_4·i_4$  
algsys([eqn1,eqn2,eqn3,eqn4,eqn5,eqn6,eqn7,eqn8],[u_1,u_2,u_3,u_4,i_1,i_2,i_3,i_4]);
```

Abbildung 9.4: Eingabe wxMaxima Type 1 & 2

Die Lösungen für das Gleichungssystem werden vom Programm in Textform zurückgegeben. Das Programm findet neben der relevanten Lösung noch einige komplexe Lösungen für das Gleichungssystem. Diese werden hier aus Platzgründen und geringer Relevanz weggelassen. Bei einer Quellenspannung von 30 V und einem Gesamtstrom von 1.931 A (Type 1 & 2) bzw. 1.729 A (Type 3 & 4) ergibt sich eine Gesamtleistung von 57.93 bzw. 51.87 W. Bei Type 3 & 4 ist die Belastung der Quelle um 10 % niedriger als bei Type 1 & 2.

	Type 1 & 2			Type 3 & 4		
	U	ΔU	I	U	ΔU	I
Quelle	30.00 V		1931 mA	30.00 V		1729 mA
Station 1	27.35 V	8.83%	438 mA	28.81 V	3.97%	416 mA
Station 2	25.30 V	15.67%	474 mA	27.91 V	6.97%	430 mA
Station 3	23.90 V	20.33%	502 mA	27.31 V	8.97%	439 mA
Station 4	23.19 V	22.70%	517 mA	27.00 V	10.00%	444 mA

Tabelle 9.1: Berechnung Ergebnisse

Man sieht: Der Spannungsabfall ist wie erwartet bei Type 3 und 4 weniger als halb so groß als bei Type 1 und 2. Dadurch ist auch die gesamte Verlustleistung der Leitungen geringer, was eine niedrigere Belastung der Quelle bedeutet.

Aus der obigen Rechnung ergibt sich, dass für die gegebene Anlage eine Versorgung nach Type 3 bzw. 4 empfehlenswert ist.

10 Verbesserte Hardwarelösung

10.1 Konzept

Die neue Lösung sollte den Aufbau einer Station vereinfachen und die Ausfallsrate einer Station auf gänzlich null reduzieren. Um dieses Ziel zu erreichen, werden folgende Ansätze definiert:

Zusammenfassen der Platinen zu einer einzigen: Die Reduktion der Anzahl der Platinen auf eine Platine ermöglicht die Zusammenfassung von verschiedenen Modulen. Damit werden die Produktionskosten gesenkt und die Montage in den Stationen vereinfacht.

Reduzierung der Bauteile: Die Zusammenfassung der Module auf eine einzelne Platine ermöglicht die Reduktion von Bauteilen. Insbesondere werden Bauteile für die Verbindungstechnik eingespart. Dies führt zu einer Kostenreduktion bei gleichzeitiger Senkung der Fehlermöglichkeiten.

Entfall der Verdrahtung zwischen den Einzelplatinen: Der Entfall der Verdrahtung führt zu einer Steigerung der Übersichtlichkeit und reduziert zusätzlich den Platzbedarf. Aufwände für die händische Verdrahtung und die Prüfung der Verbindungen der einzelnen Platinen entfällt. Fehlverdrahtung wird konstruktiv durch die Einzelplatine verhindert.

Verbesserung der Spannungsversorgung: Leistungsbeschränkungen sowie die Verwendung eines neuen verbesserten Spannungsreglers verbessern die Spannungsqualität. Die verstärkte Spannungsversorgung ermöglicht ein späteres Update auf den leistungsstärkeren RPi Model 4.

Einführung WatchDogTimer: Zusätzlich wird ein Watchdog mittels eines Mikroprozessors verbaut. Dieser wird benötigt, um eventuelle Ausfälle des RPi und somit der gesamten Station zu erkennen. Falls ein Fehler auftritt, setzt der Mikroprozessor den RPi zurück und die Station startet neu. Nachdem die Station neugestartet ist, wird der Normalbetrieb wieder aufgenommen und der Watchdog nimmt den Zustand des Überwachens wieder ein.

10.2 Spannungsversorgung

10.2.1 Anforderungen

Um einen möglichst reibungslosen Betrieb zu garantieren, werden an die Spannungsversorgung mehrere Anforderungen gestellt:

- Der Eingangsspannungsbereich muss mindestens 24 bis 35 V betragen.
- Die gewünschte Ausgangsspannung beträgt 5 V.
- Der Oberwellenanteil (ripple) der Ausgangsspannung soll unter 1% liegen.
- Die Strombelastbarkeit muss mindestens 2.5 A betragen.

Die Versorgungsspannung wird über PoE (Power over Ethernet) zur Verfügung gestellt. Der Spannungsregler muss diese Art der Versorgungsspannung unterstützen. Der Spannungsregler sollte die 24 V auf 5 V möglichst verlustarm reduzieren. Alle versorgten Komponenten auf der Platine arbeiten mit einer Spannung von 5 V. Diese Komponenten sind z.B. der RPi, der Mikrocontroller und die Verstärkerschaltung für Mikrofon und Lautsprecher.

Die Spannungsversorgung soll bei Spannungsschwankungen kurzzeitig die Versorgung aufrechterhalten, um dem Versagen einer Station vorzubeugen. Zur Stützung der Spannungsversorgung werden Kondensatoren auf der Eingangsspannungsseite und auf der Ausgangsspannungsseite benötigt.

10.2.2 Auswahl Spannungsregler

Aufgrund der oben angeführten Anforderungen wird der Spannungsregler LM33630 ausgewählt. Dieser Spannungsregler kann im Bereich von 3.8 V bis 36 V betrieben werden. Die Ausgangsspannung ist immer kleiner als die Eingangsspannung und liegt im Bereich von 1 V bis 24 V. Der LM33630 ist ein Step-down Spannungsregler, das bedeutet er kann nur von einer höheren auf eine niedrigere Spannung regeln.

Um eine Schwingung, die wenig Aufwand zur Glättung benötigt, zu erzeugen, verwendet der Spannungsregler eine hohe Schaltfrequenz von 2.1 MHz. Der LM33630 hat standardmäßig einen maximalen Ausgangstrom von 3 A, dieser wird auch nahezu vollständig benötigt für den RPi, der allein bereits ca. 1.5 A bezieht. Der Lautsprecher mit der Vorschaltung benötigt ebenfalls über 0.5 A.

Vom LM33630 sind mehrere Varianten verfügbar, es gibt die DDA und RNX Variante. Die DDA Variante hat 8 Pins und einen eingebauten Kühlkörper, der mit einem großen Massefeld verbunden werden kann, damit der IC nicht überhitzt. Die RNX Variante hat 12 Pins und anstatt eines größeren Kühlflächen mehrere Ground Pins, die ebenfalls zur Kühlung des IC dienen. Zusätzlich gibt es eine weitere Unterteilung der zwei Varianten: es gibt verschiedene Schaltfrequenzen, nämlich 400, 1400 und 2100 kHz.

Die technischen Daten dieses Spannungsreglers sind in vollem Umfang im Datenblatt ersichtlich. [vgl. 32]

10.2.3 Funktion des Spannungsreglers

Der hier gewählte Spannungsregler ist ein Schaltregler, d.h. er verändert den Effektivwert der Spannung mit PWM. Über einen Spannungsteiler kann die gewünschte Ausgangsspannung eingestellt werden. Weiters kann der Spannungsregler abgeschaltet werden, wenn 0 V am Enable Pin angelegt werden. Dies sorgt für eine komplette Abschaltung der Steuerlogik. Die Verwendung dieser Funktion ist jedoch nicht vorgesehen.

10.2.4 Zusatzbeschaltung des Spannungsreglers

Der Spannungsregler ist universell einsetzbar, weshalb die genauen Betriebseigenschaften über die externe Beschaltung festgelegt werden.

Eingangskondensator: Auf der Eingangsseite, d.h. der 24 V Seite wird ein 820 μF Kondensator verbaut. Dieser dient in erster Linie zum Abfangen und Ausgleichen von Spannungsschwankungen. Diese Spannungsschwankungen können aufgrund von sich schlagartig änderndem Leistungsverbrauch entstehen.

Ausgangskondensator: Nach dem Spannungsregler auf der 5 V Seite werden zwei Kondensatoren mit jeweils der Kapazität von 1 mF verbaut, das ergibt eine Gesamtkapazität auf der 5 V Seite von 2 mF. Diese werden zur nahezu vollständigen Glättung der Ausgangsspannung und weiteres Abfangen von Spannungseinbrüchen aufgrund von Leistungsspitzen verwendet.

Spannungsteiler: Der Spannungsteiler (100 kOhm; 23,9 kOhm) am FB Pin sorgt für die richtige Ausgangsspannung von 5 V. Dieser Spannungsteiler kann frei über die

folgende Formel konfiguriert werden:

$$R_{FBT} = \frac{V_{OUT}}{V_{REF}} - 1$$

Im Datenblatt des Spannungsreglers sind bereits Werte für die am häufigsten verwendeten Spannungen vorgegeben.

Kühlung: Der Spannungsregler muss gekühlt werden, da bei ihm eine nicht vernachlässigbare Verlustleistung auftritt. Unter dem IC ist eine große Massenfläche bereitgestellt, die zur Kühlung dient. Die Kühlfläche des Spannungsreglers wird mit Durchkontaktierungen (Vias) mit der Massefläche auf der Unterseite der Platine verbunden und somit gekühlt.

10.3 Lautsprecher-Verstärker

Ein Audio-Leistungsverstärker (oder Leistungsverstärker) ist ein elektronischer Verstärker, der elektronische Audiosignale mit geringer Leistung, wie z.B. das Signal von einem Radioempfänger oder einem E-Gitarren-Tonabnehmer, auf einen Pegel verstärkt, der hoch genug ist, um Lautsprecher oder Kopfhörer anzutreiben. Während das Eingangssignal eines Audio-Leistungsverstärkers, wie z.B. das Signal einer elektrischen Gitarre, vielleicht nur einige hundert Mikrowatt misst, kann seine Ausgangsleistung bei kleinen Geräten der Unterhaltungselektronik, wie z.B. Radiowecker, einige zehn oder hundert Watt für eine Heimstereoanlage, mehrere tausend Watt für die Beschallungsanlage eines Nachtclubs oder Zehntausende von Watt für ein großes Rockkonzert-Soundsystem betragen. [vgl. 33]

10.3.1 Anforderungen

Wichtige Entwurfsparameter für Audio-Leistungsverstärker sind Frequenzgang, Verstärkung, Rauschen und Verzerrung. Diese sind voneinander abhängig; eine Erhöhung der Verstärkung führt oft zu einer unerwünschten Zunahme von Rauschen und Verzerrung. Eine negative Rückkopplung reduziert zwar die Verstärkung, aber auch die Verzerrung. Der Verstärkerchip soll einen Lautsprecher mit $8\ \Omega$ betreiben können. Er sollte eine Leistung aufweisen, damit die Töne beim Lautsprecher ausreichend laut abgespielt werden. Der Lautsprecherverstärker soll einen möglichst hohen Wirkungsgrad besitzen, um die Verlustleistung möglichst niedrig zu halten und damit zusätzliche Kühlung zu vermeiden.

10.3.2 Auswahl

Der PAM8403 ist ein 3W, Klasse-D-Audioverstärker. Er bietet einen niedrigen THD+N-Wert (bedeutet „total harmonic distortion plus noise“. Sie wird normalerweise, durch Eingabe einer Sinuswelle, Bandsperrfilterung des Ausgangs und Vergleich des Verhältnisses zwischen dem Ausgangssignal mit und ohne Sinuswelle, gemessen), wodurch eine hochwertige Klangwiedergabe erreicht wird. Die neue filterlose Architektur ermöglicht es dem Gerät, den Lautsprecher direkt anzusteuern, so dass keine Tiefpassausgangsfilter erforderlich sind, wodurch Systemkosten und Leiterplattenfläche eingespart werden. Der PAM8403 weist eine Effizienz von bis zu 90 % auf, deutlich höher als Verstärker aus den anderen Verstärkerklassen. Diese weisen in den niederen Leistungsbereich meist eine Effizienz von 50 % auf. Bei einem $8\ \Omega$ Lautsprecher ist eine maximale Leistung von 1.5 W möglich und ein Wirkungsgrad von 90 %.

10.3.3 Funktionen

Maximale Verstärkung: Der PAM8403 hat zwei interne Verstärkerstufen. Die Verstärkung der ersten Stufe ist extern konfigurierbar, während die der zweiten Stufe intern fest eingestellt ist. Die Regelverstärkung der ersten Stufe wird durch die Wahl des Verhältnisses von RF zu RI eingestellt, während die Verstärkung der zweiten Stufe auf 2x festgelegt ist. Der Ausgang von Verstärker 1 dient als Eingang zu Verstärker 2, sodass die beiden Verstärker Signale erzeugen, die in der Größe identisch sind, sich aber in der Phase um 180° unterscheiden.

Mute Funktion: Der MUTE-Pin ist ein Eingang zur Steuerung des Ausgangszustandes des PAM8403. Ein logisches Low an diesem Pin deaktiviert die Ausgänge, und ein logisches High an diesem Pin aktiviert die Ausgänge. Dieser Pin kann als schnelle Deaktivierung oder Aktivierung der Ausgänge ohne Lautstärkeverlust verwendet werden. Der MUTE-Pin kann aufgrund des internen Pull-ups potentialfrei bleiben.

Shutdown Funktion: Um den Stromverbrauch bei Nichtbenutzung zu reduzieren, enthält der PAM8403 eine Abschaltung, mit der die Vorspannungsschaltung des Verstärkers abgeschaltet werden kann. Diese Abschaltfunktion schaltet den Verstärker ab, wenn am SHDN-Pin ein logischer Low-Wert anliegt. Durch Schalten des mit GND verbundenen SHDN-Pins wird die Stromaufnahme des PAM8403 im Leerlauf minimiert. Der SHDN-Pin kann aufgrund des internen Pull-ups potentialfrei bleiben.

10.3.4 Zusatzbeschaltung

Der Verstärker benötigt nur wenige Komponenten als Zusatzbeschaltung. Die Spannungsversorgung wird mit Kondensatoren parallelgeschaltet, um mögliche Spannungsschwankungen auszugleichen. Der Eingang des Audiosignales vom RPi wird zuerst über einen Kondensator geführt um Störungen zu vermeiden und Gleichanteile herauszufiltern.

Der Audioverstärker hat insgesamt 3 Masseanschlüsse, darunter befindet sich ein Masseanschluss für die interne Steuerschaltung. Die zwei anderen Pins sind für die zwei Lautsprecher selbst, über sie fließt der Strom des Lautsprechers ab.

10.3.5 Kühlung

Der Audioverstärker muss nicht aktiv gekühlt werden, da ein Wirkungsgrad von rund 90 % vorliegt. Das bedeutet bei einer Leistung von 1.5 W tritt nur eine Verlustleistung 0.15 W auf.

10.4 Mikrocontroller

Ein Mikrocontroller ist ein Halbleiterchip, der einen Prozessor und zugleich auch Peripheriefunktionen enthält. Der Arbeits- und Programmspeicher befindet sich vollständig auf demselben Chip, deshalb ist der Mikrocontroller ein Ein-Chip-Computersystem. Auf modernen Controllern finden sich auch komplexe Peripheriefunktionen, sowie die Schnittstellen Universal Serial Bus (USB), Inter-IC (I²C) und SPI, PWM-Ausgänge oder Analog-Digital-Umsetzer. Es gibt eine breite Menge an verschiedenen Mikrocontrollern. Angefangen bei Controllern mit einem Takt von 4 kHz und Leistung von wenigen Milliwatt oder Mikrowatt für einfache Aufgaben, wie Warten auf Betätigen eines Tasters. Bis zu Controllern mit einem Takt von mehrere MHz und einen Programmspeicher von mehreren kBytes für größere und komplexere Aufgaben.

Der AVR-Mikrocontroller ATmega328P wurde aus mehreren Gründen ausgewählt:

- Das Programmieren dieses Chips wird einerseits durch die Unterstützung der Arduino IDE, mit der wir bereits sehr gut vertraut sind, sehr stark vereinfacht.
- Das Hochladen des Programmes in den ATmega328P stellt ebenfalls kein Problem dar, ein Arduino kann als In-Circuit Serial Programmer (ICSP) verwendet werden.

- Der Controller benötigt nur eine minimale Beschaltung, um ordnungsgemäß zu funktionieren.

10.4.1 Atmel AVR

Die AVR-Mikrocontroller von Atmel (jetzt Microchip Technology Inc.) sind wegen ihrer übersichtlichen internen Struktur, der In-System-Programmierbarkeit, und der Vielzahl von kostenlosen Programmen zur Softwareentwicklung (Assembler, Compiler) beliebt. Diese Eigenschaften und der Umstand, dass viele Typen in einfach handhabbaren DIP-Gehäusen verfügbar sind, machen den AVR zum idealen Mikrocontroller für Anfänger. Über die Bedeutung des Namens „AVR“ gibt es verschiedene Ansichten; manche meinen es sei eine Abkürzung für Advanced Virtual Reduced Instruction Set Computer (RISC), andere vermuten dass der Name aus den Anfangsbuchstaben der Namen der Entwickler (Alf Egil Bogen und Vegard Wollan RISC) zusammengesetzt wurde. Laut Atmel ist der Name bedeutungslos.

10.4.2 Architektur

Die Architektur ist eine 8-Bit-Harvard-Architektur, das heißt, es gibt getrennte Busse zum Programmspeicher (Flash-ROM, dieser ist 16 bit breit) und Schreib-Lese-Speicher (RAM). Der Programmcode kann ausschließlich aus dem Programmspeicher ausgeführt werden. Weiterhin sind die Adressräume unabhängig (d.h. beide Speicher besitzen eigene Adressbereiche, die sich wertmäßig überschneiden können). Bei der Programmierung in Assembler und einigen C-Compilern bedeutet dies, dass sich Konstanten aus dem ROM nicht mit dem gleichen Code laden lassen wie Daten aus dem RAM. Abgesehen davon ist der Aufbau des Controllers recht übersichtlich und birgt wenige Fallstricke.

- 32 größtenteils gleichwertige Register
- davon 1–3 16-bit-Zeigerregister (paarweise)
- ca. 110 Befehle, die meist 1–2 Taktzyklen dauern
- Taktfrequenz bis 32 MHz
- Betriebsspannung von 1,8 – 5,5 V
- Speicher (1-256 kB Flash-ROM, 0-4 kB EEPROM, 0-16 kB RAM)

- Peripherie: Analog-Digital-Wandler 10 Bit, 8- und 16-Bit-Timer mit PWM, SPI, I²C, UART, Analog-Komparator, Watchdog
- 64 kB externer SRAM (ATmega128, ATmega64, ATmega8515/162)
- JTAG bei den größeren ATmegas
- debugWire bei den neueren AVRs

[34, Architektur]

10.4.3 Funktion

Der Mikrocontroller übernimmt in diesem Projekt mehrere Aufgaben.

10.4.3.1 Watchdog-Timer für Raspberry Pi

Ein Watchdog-Timer überwacht, ob ein System noch funktionsfähig ist, oder ob es irgendwo festhängt. In letzterem Fall wird vom Watchdog-Timer in den meisten Anwendungen ein Hardware-Reset ausgelöst.

Der Mikrocontroller kommuniziert periodisch mit dem RPi über UART, d.h. der Mikrocontroller sendet eine Nachricht und der RPi schickt eine Antwort zurück. Wenn die Antwort des RPi ausbleibt, hat sich dieser offensichtlich aufgehängt und muss zurückgesetzt werden. Der Mikrocontroller setzt den Reset-Pin des RPi auf Low und erzwingt damit einen Neustart der Station. Nachdem der RPi neugestartet ist, meldet er dies dem Mikrocontroller. Der Mikrocontroller geht wieder in den Zustand des Überwachens über und der RPi setzt seine normale Funktion fort.

10.4.3.2 IO-Handling

Über den Mikrocontroller werden sämtliche digitale Eingänge aufgenommen und an den RPi weitergegeben. Dies ist für die Diebstahlsicherung der Außenstation essenziell. Weiters steuert der RPi über die Digital-Ausgänge des Mikrocontrollers den Audio-Verstärker-IC hinsichtlich Betrieb an.

10.4.4 Zusatzbeschaltung

Reset-Pin: Der Reset Pin des Mikrocontroller ist bei einer Spannung von 0 V aktiv. Damit er im normalen Betrieb nicht aktiviert wird, wird er über einen Widerstand auf 5 V gezogen. Um den Reset Pin nun zu aktivieren, muss nur zwischen dem Widerstand

und dem Reset Pin 0 V angelegt werden. Damit verändert sich das Potential des Reset Pin auf 0 V und wird aktiviert.

Clock: Die Clock des Mikrocontrollers wird durch einen Schwingquarz erzeugt. Ein Schwingquarz, häufig vereinfachend abgekürzt als Quarz bezeichnet, ist ein elektronisches Bauelement, welches zur Erzeugung von elektrischen Schwingungen mit einer bestimmten Frequenz dient. Über die Schwingfrequenz wird die Arbeitsgeschwindigkeit des Mikrocontrollers festgelegt. In diesem Fall wird ein Quartz mit 6 MHz verwendet.

11 Hardware-Programmierung

11.1 Programm-Entwicklung

Die für diese Diplomarbeit entwickelte Hardware enthält intelligente Komponenten, die auch programmiert werden müssen. Diese sind zum einen der bereits vorher existierende RPi und der neu hinzugefügte ATmega328P Mikrocontroller.

11.1.1 Mikrocontroller

11.1.1.1 Entwicklungsumgebung

Die IDE von Arduino ist eine plattformübergreifende Anwendung (für Windows, MacOS, Linux), die in Funktionen aus C und C++ geschrieben ist. Sie wird zum Schreiben und Hochladen von Programmen auf Arduino-kompatible Boards, aber auch, mit Hilfe von 3rd-Party-Cores, auf Entwicklungsboards anderer Hersteller verwendet.

Der Quellcode für die IDE wurde unter der GNU GPL Version 2, veröffentlicht. Die Arduino IDE unterstützt die Sprachen C und C++ unter Verwendung spezieller Regeln der Code-Strukturierung. Die Arduino IDE liefert eine Software-Bibliothek aus dem Wiring-Projekt, die viele gängige Ein- und Ausgabeprozeduren zur Verfügung stellt. Der benutzergeschriebene Code benötigt nur zwei grundlegende Funktionen, `setup()` zum Starten des Sketch und der Hauptprogrammschleife `loop()`, die kompiliert und mit einem Programmstub `main()` zu einem ausführbaren zyklischen Ausführungsprogramm mit der GNU-Toolchain, die ebenfalls in der IDE-Distribution enthalten ist, verknüpft werden.

11.1.1.2 Programmierung

Der in dieser Diplomarbeit verwendete Mikrocontroller besitzt keine USB-Schnittstelle, sondern muss über ein externes Programmiergerät mittels SPI-Schnittstelle programmiert werden.

Als Programmiergerät wurde ein Arduino-Mega-Entwicklungsboard gewählt. Dieses verfügt über eine ICSP-Schnittstelle, mit der ein anderer Mikrocontroller über SPI programmiert werden kann. Diese Schnittstelle und deren Pinbelegung ist in Abbildung 11.1 dargestellt.



Abbildung 11.1: ICSP Schnittstelle [35, How to wire your boards]

Das Arduino-Board, welches als Programmiergerät verwendet wird, muss mit Hilfe des mitgelieferten Programmes als In-System Programming (ISP)-Programmiergerät konfiguriert werden. Das Programm ist in der Arduino IDE, wie in Abbildung 11.2 dargestellt, zu finden.

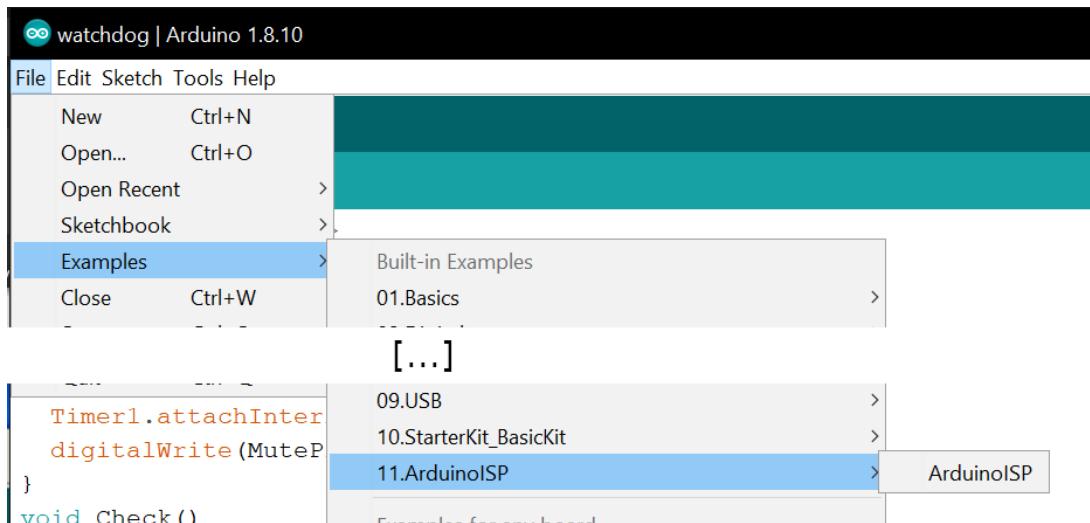


Abbildung 11.2: ArduinolISP Sketch

Bevor das eigentliche Programm auf den Mikrocontroller der Station hochgeladen werden kann, müssen noch ein paar Einstellungen in der Arduino IDE getroffen werden (siehe auch Abbildung 11.3):

- Arduino/Genuino Uno als Board auswählen (Dieses Board wird standardmäßig mit dem hier verwendeten Mikrocontroller ausgeführt)
- Auswahl der Schnittstelle, an welche das Programmiergerät angeschlossen ist.
- Die Option „Programmer“ muss von „AVRISP mkII“ auf „Arduino as ISP“ umgestellt werden.

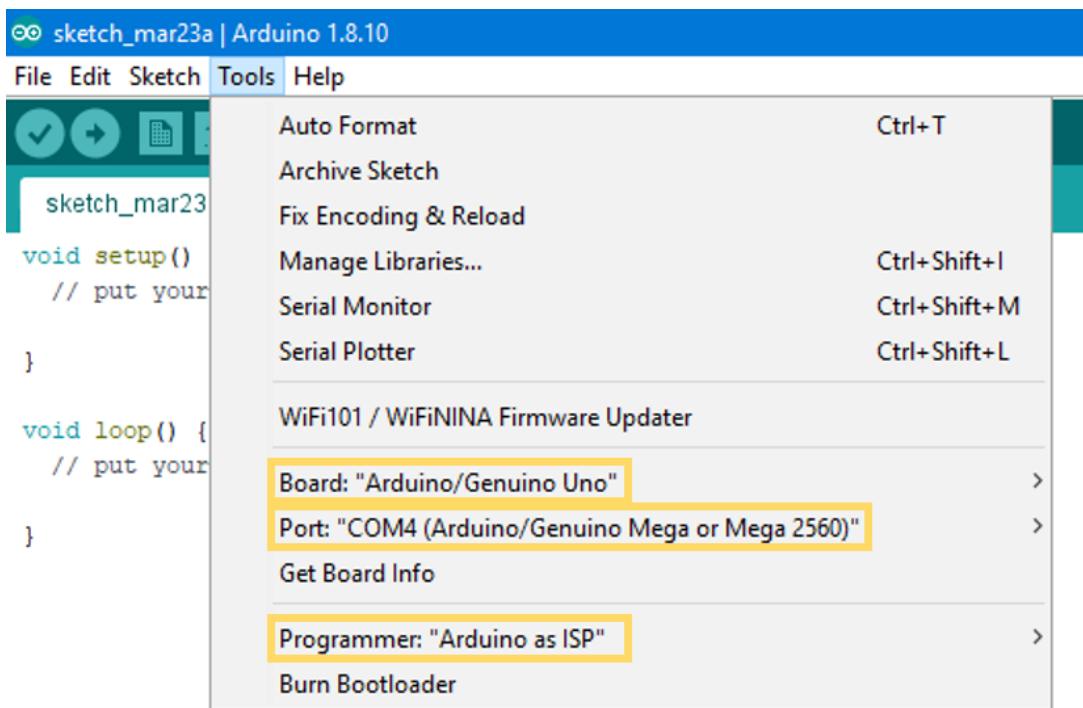


Abbildung 11.3: Arduino IDE Einstellungen

Um nun den Upload-Prozess zu starten muss in der Arduino IDE „Upload using Programmer“ ausgewählt werden (siehe Abbildung 11.4). Das Mikrocontroller-Programm wird nun kompiliert und mit Hilfe des Programmiergerätes auf den Mikrocontroller geladen.

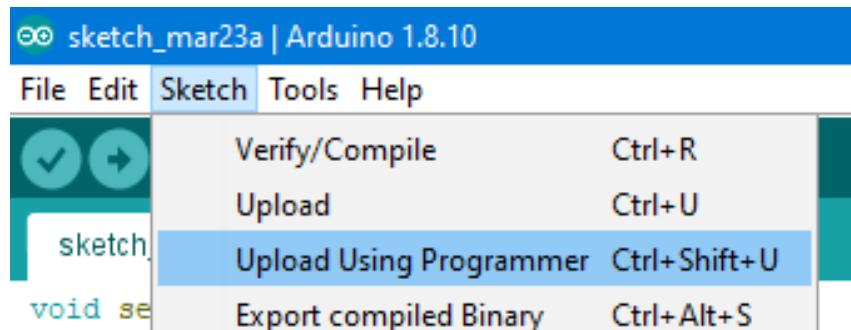


Abbildung 11.4: Hochladen des Programmes

11.1.1.3 Programm-Dokumentation

Der Mikrocontroller wird in C++ mit vorgefertigten Funktionen von Arduino programmiert.

```

1 #include <TimerOne.h>                                1
2 const int ResetPin = 2, MutePin = 3, SecurityPin = 4;  2
3 volatile bool FindEnable = false;                      3
4 bool Reboot = true;                                    4

```

1: Die TimerOne Bibliothek ermöglicht die Verwendung der Interrupt-Funktion des Hardware-Timers des Mikrocontrollers, wodurch das periodische Aufrufen einer Funktion vereinfacht wird. Dies dient dem ständigen Abfragen des Betriebszustandes des RPi.

2: Hier werden die verwendeten Pin-Nummern des Mikrocontrollers in Konstanten gespeichert, um das Programm lesbar zu halten. Zu beachten ist, dass die Nummern des Programms nicht mit den physikalischen Pin-Nummern übereinstimmen. Das liegt daran, dass die Zuordnung auf einem Arduino-Entwicklungs-Board basiert.

3-4: Weiters werden alle globalen Variablen definiert. Das volatile-Schlüsselwort wird gebraucht, wenn die Variable in einer Interrupt Service Routine verändert wird. Das Schlüsselwort hat zur Folge, dass der Wert der Variable nicht zwischengespeichert, sondern immer der aktuelle Wert abgerufen wird. Dies ist notwendig, da die Interrupt Service Routine (ISR) den Programmablauf jederzeit unterbrechen und den Wert zu einem ungünstigen Zeitpunkt verändern kann.

```

6 void setup()                                         6
7 {                                                 7
8     pinMode(ResetPin, OUTPUT);                     8
9     pinMode(MutePin, OUTPUT);                     9
10    pinMode(SecurityPin, INPUT);                  10
11
12    Serial.begin(9600);                           11
13    Serial.setTimeout(1000);                      12
14    Timer1.initialize(10000000);                 13
15    Timer1.attachInterrupt(Check);               14
16    digitalWrite(MutePin, HIGH);                  15
17 }                                                 16

```

Die Setup-Funktion wird einmalig nach dem Start bzw. Reset des Mikrocontrollers aufgerufen. In dieser werden meist Initialisierungen durchgeführt.

8-10: Mit `pinMode()` wird festgelegt, ob ein IO-Pin als Ausgang oder als Eingang konfiguriert ist.

12-13: Die serielle UART-Schnittstelle kann mit dem `Serial`-Objekt manipuliert werden. Zum Initialisieren muss die Übertragungsrate in Bit/s angegeben werden, da UART keine Takteleitung hat. Weiters wird das Timeout auf 1 Sekunde gesetzt. Die `Serial.find()`-Funktion, welche im Hauptteil verwendet wird, versucht so lange eine Zeichensequenz in der Datenübertragung zu finden, bis das Timeout abgelaufen ist.

14-15: An dieser Stelle wird die TimerOne-Bibliothek initialisiert. Zuerst wird die Periodendauer zwischen den Interrupts in `us` definiert. 10 Sekunden sind für diese Anwendung ausreichend schnell, während die Schnittstelle nicht unnötig viel verwendet wird. Mit `attachInterrupt()` kann die Funktion für die ISR definiert werden.

16: Mit `digitalWrite()` kann der Zustand eines digitalen Ausgangs festgelegt werden. Hier wird der Ausgang, welcher den Lautsprecher-Verstärker abschaltet, auf logisch 1 gesetzt. Dies verhindert etwaige Störgeräusche am Lautsprecher, wenn dieser nicht gebraucht wird. Der RPi kann mit einem Befehl über die UART-Schnittstelle den Lautsprecher freigeben. Dies ist allerdings noch nicht vollständig implementiert.

```
19 void Check()                                19
20 {                                         20
21     FindEnable = true;                      21
22 }                                         22
```

19-22: Die ISR sollte generell sehr kurz gehalten werden, da sie den normalen Programmablauf unterbricht. Funktionen der seriellen Kommunikation werden deswegen hier nicht verwendet. Stattdessen wird eine Variable gesetzt, welche im Hauptprogramm einen alternativen Ausführungszweig auslöst. Wichtig ist, dass die Variable wie oben bereits beschrieben als `volatile` deklariert wurde.

```
24 void loop()                                 24
25 {                                         25
26     if (FindEnable && !Reboot)            26
27     {                                         27
28         Serial.write("Hello\n");           28
29         FindEnable = false;               29
```

```

30     if (Serial.find("Imthere"))
31     {
32         digitalWrite(ResetPin, LOW);
33     }
34     else
35     {
36         Reboot = true;
37         digitalWrite(ResetPin, HIGH);
38         delay(1000);
39         digitalWrite(ResetPin, LOW);
40     }
41 }
42 else if (Reboot)
43 {
44     if (Serial.find("Rebooted"))
45     {
46         Reboot = false;
47         FindEnable = false;
48     }
49 }
```

26: Die erste `if`-Abfrage stellt fest, ob der Watchdog seine Überprüfung durchführen soll. Als Bedingungen für eine Überprüfung des RPi wird hier definiert, dass der RPi nicht gerade neustartet und die Watchdog-Zeit abgelaufen ist.

28-40: Die Überprüfung wird durchgeführt. Der Mikrocontroller sendet „Hello\n“ (\n signalisiert das Ende der Übertragung) und wartet auf die Antwort „Imthere“ des RPi. Nach 1000 ms ohne Antwort setzt der Mikrocontroller den RPi mit Hilfe des Reset-Pins zurück. Zusätzlich wird die Variable `Reboot` gesetzt, um den RPi nicht während des Neustarts erneut zurückzusetzen.

42-49: Die `else if`-Anweisung sorgt beim Neustart des RPi für Schutz vor ungewolltem Rücksetzen. Die `Rebooted`-Variable wird erst dann wieder zurückgesetzt, wenn der RPi die Nachricht „Rebooted“ schickt. Diese signalisiert, dass der RPi vollständig hochgefahren ist und auf weitere Anfragen reagieren kann.

```

51     if (Serial.available())
52     {
53         if (Serial.find("Imclosing"))
54             Reboot = true;
55         if (Serial.find("Enable"))
56             digitalWrite(MutePin, HIGH);
57         if (Serial.find("Disable"))
58             digitalWrite(MutePin, LOW);
59     }
60 //-----
61     if (digitalRead(SecurityPin))
62         Serial.write("Securitybreached");
63 }
```

51-59: Innerhalb dieser if-Abfrage wird auf sonstige Ereignisse, die der RPi auslösen kann reagiert. Wenn der RPi neustartet, sendet er die Nachricht „Imclosing“ an den Mikrocontroller, damit dieser den Watchdog-Timer temporär ausschaltet. Mit „Enable“ und „Disable“ kann der RPi den Lautsprecher-Verstärker ein- bzw. ausschalten.

61-62: Als letztes wird die Diebstahlsicherung für eine Station über den Mikrocontroller gesteuert. Falls die Diebstahlsicherung anschlägt wird zum RPi eine Nachricht übermittelt, dass dieser entsprechend reagiert.

11.1.2 Raspberry Pi

Als Gegenstück zum Mikrocontroller muss auf dem RPi ein weiteres Programm laufen, welches auf die Anfragen des Watchdogs antwortet und Befehle zur Steuerung des Audio-Chips an den Mikrocontroller schickt. Dieses Programm soll automatisch gestartet werden und sich mit dem Mikrocontroller verbinden.

Das Programm wird wie die eigentliche Oberfläche der Station in C# geschrieben. Als Entwicklungsumgebung hat man sich daher für Visual Studio 2019 entschieden. Die kompilierte Anwendung wird mit Hilfe des Secure Shell (SSH)-Protokolls auf den RPi übertragen und dort mit Hilfe der Mono-Runtime ausgeführt.

```

13 static SerialPort port = new SerialPort("/dev/ttys0", 9600, Parity.None,
14                                     8, StopBits.One);
14 static void Main(string[] args)
```

```

15  {
16      AppDomain.CurrentDomain.ProcessExit += new
17          EventHandler(CurrentDomain_ProcessExit);
18

```

13: Das .NET-Framework stellt Klassen für die serielle Kommunikation via UART zur Verfügung. Die `SerialPort`-Klasse benötigt für die Initialisierung alle wichtigen Informationen über die Verbindung, wie die Adresse des Ports, die Übertragungsgeschwindigkeit und wieviele Daten-Bits in einem Frame enthalten sind.

14-16: Eine .NET-Konsolenapplikation hat als Standard-Einspringpunkt die `Main`-Funktion, welche die optionalen Kommandozeilen-Argumente über ein Array annimmt. Um die Verbindung mit dem Mikrocontroller kontrolliert abzubrechen, wenn der RPi manuell heruntergefahren wird, wird das `ProcessExit`-Ereignis verwendet. Dieses wird unmittelbar vor dem Beenden des Programms aufgerufen.

```

17  try
18  {
19      port.Open();
20      port.WriteLine("Rebooted");
21      while (true)
22      {
23          if (port.ReadTo("\n").ToString().Contains("Hello"))
24              port.WriteLine("Imthere");
25      }
26  }
27  catch (Exception exc)
28  {
29      Console.WriteLine(exc);
30  }
31 }

```

17: Mit der `try`-Funktion wird ein kritischer Runtime-Fehler abgefangen, damit das Programm nicht sofort abstürzt. In diesem Fall wird der auftretende Fehler vor dem Beenden in der Konsole ausgegeben. Ein solcher Runtime-Fehler kann in diesem Fall bei allen Operationen mit dem `SerialPort`-Objekt auftreten, wenn z.B. der angegebene Port bereits vergeben ist oder garnicht existiert.

19-20: Als erstes wird die Verbindung mit dem Mikrocontroller aufgebaut. Der RPi sendet die Nachricht „Rebooted“, um dem Mikrocontroller zu signalisieren, dass er auf Watchdog-Anfragen reagieren kann.

21-25: Der RPi wechselt nun in die Endlosschleife und überprüft bei jedem Durchlauf, ob der Mikrocontroller eine Watchdog-Anfrage geschickt hat. Sollte das der Fall sein, antwortet der RPi auf diese.

```
32 static void CurrentDomain_ProcessExit(object sender, EventArgs e)      32
33 {                                         33
34     if (port.IsOpen)                      34
35     {                                     35
36         port.WriteLine("Imclosing");       36
37         port.Close();                     37
38     }                                     38
39 }
```

34-38: Bevor das Programm beendet wird, überprüft der RPi, ob er noch mit dem Mikrocontroller verbunden ist, um einen Runtime-Fehler zu umgehen. Sollte das der Fall sein, wird der Mikrocontroller über das Abschalten informiert, und die Verbindung wird getrennt. Danach wird das Programm vollständig beendet.

Um das hier beschriebene Programm automatisch nach dem Hochfahren des RPi zu starten, wird „systemd“ verwendet. „systemd“ ist ein Linux-Programm für die Verwaltung und das automatische Starten von Systemprozessen und Diensten. Die genaue Vorgehensweise kann dem in der Linux-Welt bekannten Arch Linux Wiki entnommen werden. [vgl. 36, Writing unit files]

12 Platinen-Entwicklung

12.1 EAGLE

Die Software EAGLE von Autodesk ist eine Computer-aided Design (CAD) Software zur Entwicklung von Leiterplatten (PCB). Bis 2016 hat das deutsche Unternehmen CadSoft Computer die Software entwickelt, angefangen im Jahr 1988 mit dem Erscheinen von Eagle, damals noch eine 16 Bit Anwendung für DOS. Ab 2000 wurde die Unterstützung für DOS niedergelegt und die Software wurde für Windows und Linux weiterentwickelt. Bis 2016 wurde die Software für 64 Bit entwickelt und neue Funktionen wurden hinzugefügt. Ab 2016 ist die Software im Besitz von Autodesk, die den einmaligen Kauf der Software auf ein Abo Modell umgestellt haben.

Eagle besteht grundsätzlich aus zwei Fenstern, zum einen die schematische bzw. Schaltplan-Darstellung und zum anderen der Leiterplattendesigner. In der Schaltplan-Darstellung wird dieser erstellt, die einzelnen Bauteile definiert und deren Eigenschaften festgelegt. Im Leiterplattendesigner wird die eigentliche Platine erstellt, die Bauteile auf der Platine platziert und mit Leiterbahnen verbunden.

12.1.1 Versionsvergleich

Version	Premium	Student & educator	Standard	Free
Schaltpläne pro Projekt	999	999	99	2
Zahl der Schichten	16	16	4	2
Größe der Leiterplatte	4 m ²	4 m ²	160 cm ²	80 cm ²
Nutzung	beliebig	Ausbildung	beliebig	privat & nicht-kommerziell
Kosten pro Monat	USD 65	gratis	USD 15	gratis
Kosten pro Jahr	USD 500	gratis	USD 100	gratis

Tabelle 12.1: Verfügbare EAGLE-Versionen ab 2017 [vgl. 37]

12.1.2 Entwicklung mit EAGLE

12.1.2.1 Schaltplan

Zuerst wird in EAGLE der Schaltplan erstellt. Dabei werden alle benötigten Komponenten eingefügt und miteinander verbunden. Eagle verfügt bereits über eine große Bibliothek an Komponenten von verschiedenen Herstellern, jedoch müssen für speziellere Komponenten Bibliotheken gesucht werden oder selbst erstellt werden. Eagle unterstützt die Erstellung und Bearbeitung von Bibliotheken. Diese Bibliotheken können anschließend direkt in ein Projekt eingebunden werden.

Nachdem der Schaltplan erstellt wurde, kann die Schaltung mittels Electrical Rule Check (ERC) überprüft werden. Diese Funktion überprüft, ob alle grundlegenden Anforderungen einer Schaltung erfüllt wurden, z.B. ob alle Anschlüsse verbunden worden sind oder ob alle Bauteile einen Wert haben.

12.1.2.2 Layout

Im Layout-Fenster wird die eigentliche Platine erstellt. Um das Platinen Design Fenster zu öffnen, wird der Befehl Board in der Schaltplanansicht eingegeben. Alle Komponenten aus dem Schaltplan werden übernommen, und die Verbindungen der Komponenten werden durch gelbe Linien gekennzeichnet.

Zuerst werden die Bauteile positioniert. Dabei werden alle Komponenten, die miteinander verbunden werden müssen, nahe beieinander platziert. Nachdem alle Teile platziert wurden, können die Leiterbahnen eingezeichnet werden. Es kann per Hand mit dem Befehl „route“ gezeichnet werden, oder mittels Autorouter erstellt werden. Der Autorouter benötigt jedoch viel Erfahrung, um korrekt genutzt werden zu können und das Ergebnis ist meist nicht ideal. Mit Hand zeichnen ist am Anfang deutlich einfacher.

Nachdem die Leiterbahnen erstellt wurden wird die Platine mit dem Design Rule Check (DRC) geprüft werden. Dieser Befehl prüft wiederrum, ob alle grundlegenden Anforderungen erfüllt wurden, z.B. Mindestabstand der Leiterbahnen eingehalten oder noch nicht vollständig geroutete Leiterbahnen vorhanden sind.

13 Hardware-Ausblick

13.1 Gehäuse für Platine

Momentan wird die Platine mit Hilfe von Heißkleber ohne Gehäuse in der Station fixiert. EAGLE bietet in der neuesten Version eine Exportfunktion in die A360-Cloud von Autodesk an, mit welcher die Platine inklusive der verwendeten Bauteile als 3D-Modell exportiert werden kann. Dieses Modell kann in dafür geeigneten Autodesk-Programmen wie Fusion 360 geöffnet und für die Planung eines Gehäuses verwendet werden.

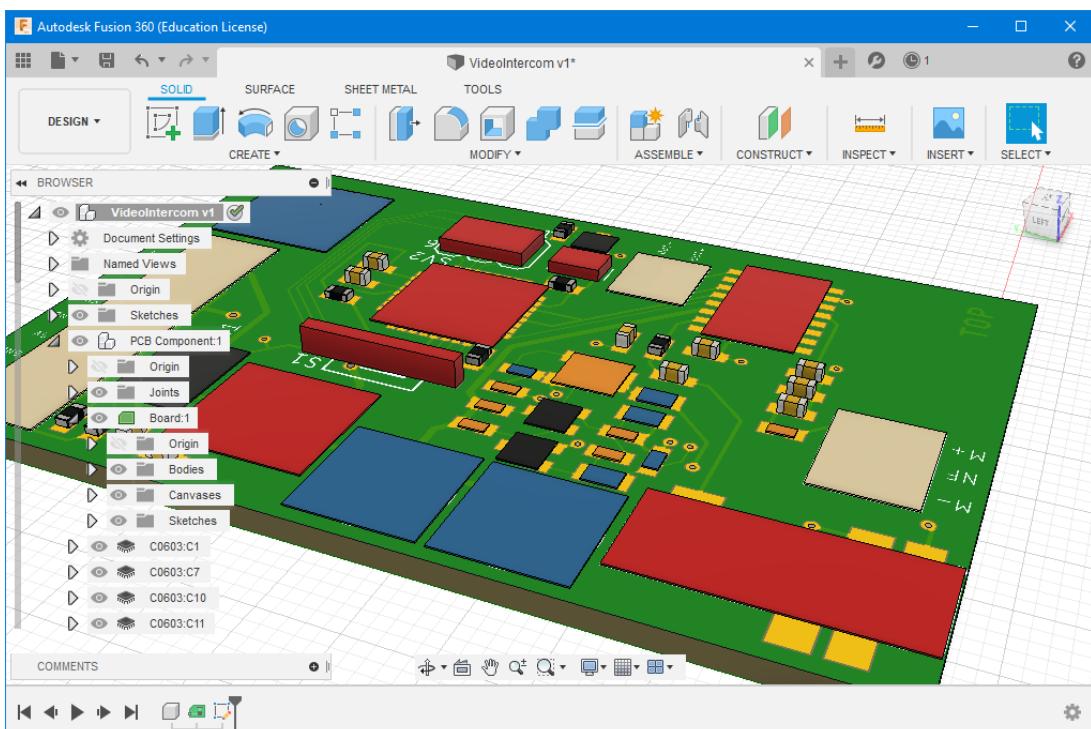


Abbildung 13.1: Exportierte Platine in Fusion 360

13.2 Platine Version 2

Die im Zuge dieser Diplomarbeit entwickelte Platine ist zwar voll funktionsfähig, jedoch bedarf es weiterer Verbesserungen. Zum einen wurden aufgrund von Miskommunikation die Anschlüsse, die zum Programmieren notwendig sind, falsch verdrahtet. Die zweite Version des Platinendesigns würde dieses Problem beheben. Zusätzlich zu den be-

reits vorhandenen Anschlüssen sollen in der zweiten Version noch ein paar zusätzliche digitale IOs hinzugefügt werden, um die Diebstahlsicherung der Außenstation auswerten zu können. Die Software würde diese Funktionalität bereits teilweise unterstützen. Die zweite Platinenversion ist bereits vollständig geplant, allerdings noch nicht gefertigt worden.

Teil III

Appendix

A Datenblätter

A.1 Mikrocontroller Atmel ATmega328P

Nachfolgend ein Auszug aus dem Datenblatt mit den wichtigsten technischen Daten, der Pinbelegung und einem Blockdiagramm. Die vollständige Dokumentation umfasst über 294 Seiten und kann unter nachfolgender Adresse abgerufen werden.

(Jan. 2015). „ATmega328P Datasheet“, Atmel, Adresse: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf



ATmega328P

**8-bit AVR Microcontroller with 32K Bytes In-System
Programmable Flash**

DATASHEET

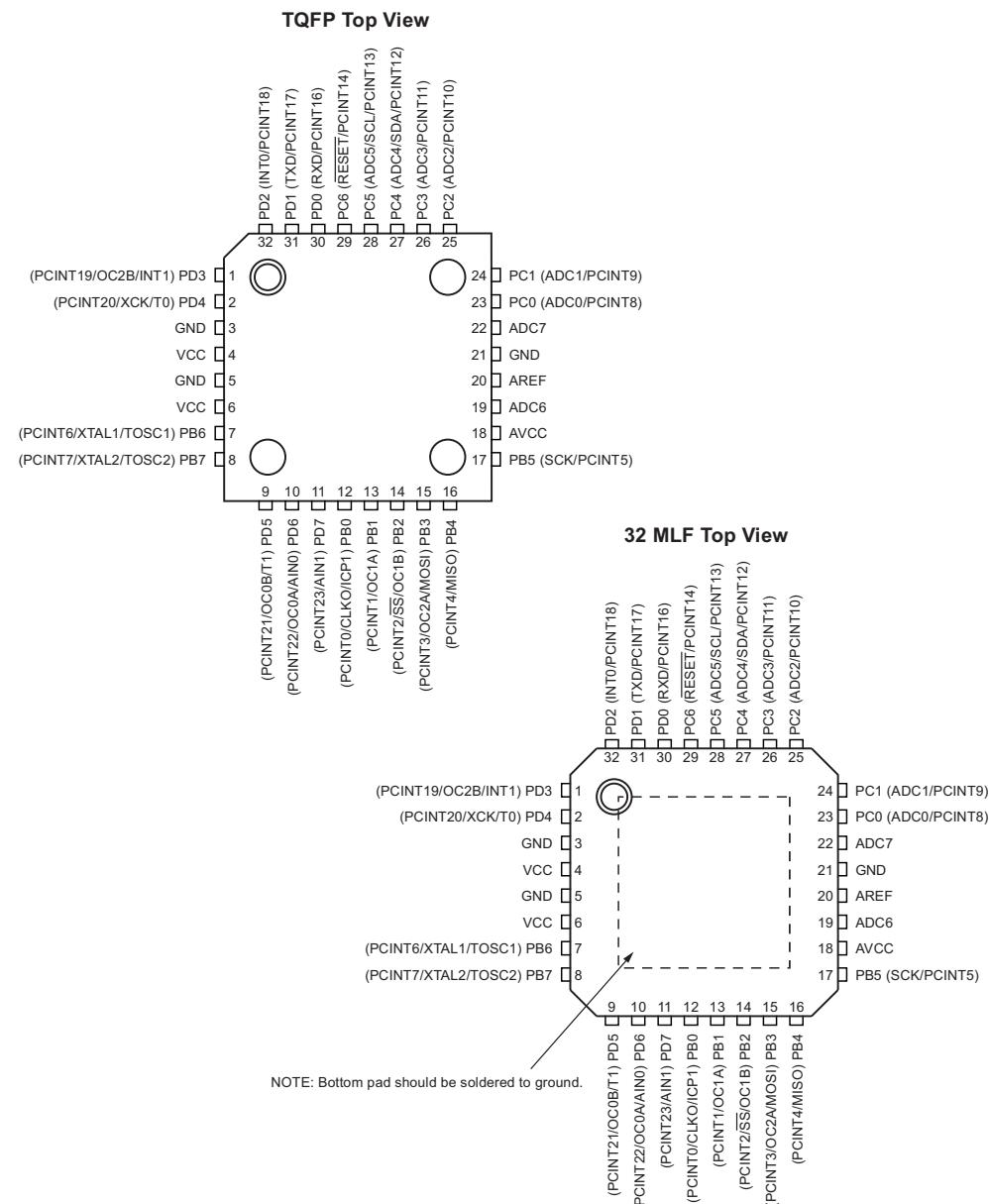
Features

- High performance, low power AVR® 8-bit microcontroller
- Advanced RISC architecture
 - 131 powerful instructions – most single clock cycle execution
 - 32×8 general purpose working registers
 - Fully static operation
 - Up to 16MIPS throughput at 16MHz
 - On-chip 2-cycle multiplier
- High endurance non-volatile memory segments
 - 32K bytes of in-system self-programmable flash program memory
 - 1Kbytes EEPROM
 - 2Kbytes internal SRAM
 - Write/erase cycles: 10,000 flash/100,000 EEPROM
 - Optional boot code section with independent lock bits
 - In-system programming by on-chip boot program
 - True read-while-write operation
 - Programming lock for software security
- Peripheral features
 - Two 8-bit Timer/Counters with separate prescaler and compare mode
 - One 16-bit Timer/Counter with separate prescaler, compare mode, and capture mode
 - Real time counter with separate oscillator
 - Six PWM channels
 - 8-channel 10-bit ADC in TQFP and QFN/MLF package
 - Temperature measurement
 - Programmable serial USART
 - Master/slave SPI serial interface
 - Byte-oriented 2-wire serial interface (Phillips I²C compatible)
 - Programmable watchdog timer with separate on-chip oscillator
 - On-chip analog comparator
 - Interrupt and wake-up on pin change
- Special microcontroller features
 - Power-on reset and programmable brown-out detection
 - Internal calibrated oscillator
 - External and internal interrupt sources
 - Six sleep modes: Idle, ADC noise reduction, power-save, power-down, standby, and extended standby

- I/O and packages
 - 23 programmable I/O lines
 - 32-lead TQFP, and 32-pad QFN/MLF
- Operating voltage:
 - 2.7V to 5.5V for ATmega328P
- Temperature range:
 - Automotive temperature range: -40°C to +125°C
- Speed grade:
 - 0 to 8MHz at 2.7 to 5.5V (automotive temperature range: -40°C to +125°C)
 - 0 to 16MHz at 4.5 to 5.5V (automotive temperature range: -40°C to +125°C)
- Low power consumption
 - Active mode: 1.5mA at 3V - 4MHz
 - Power-down mode: 1µA at 3V

1. Pin Configurations

Figure 1-1. Pinout

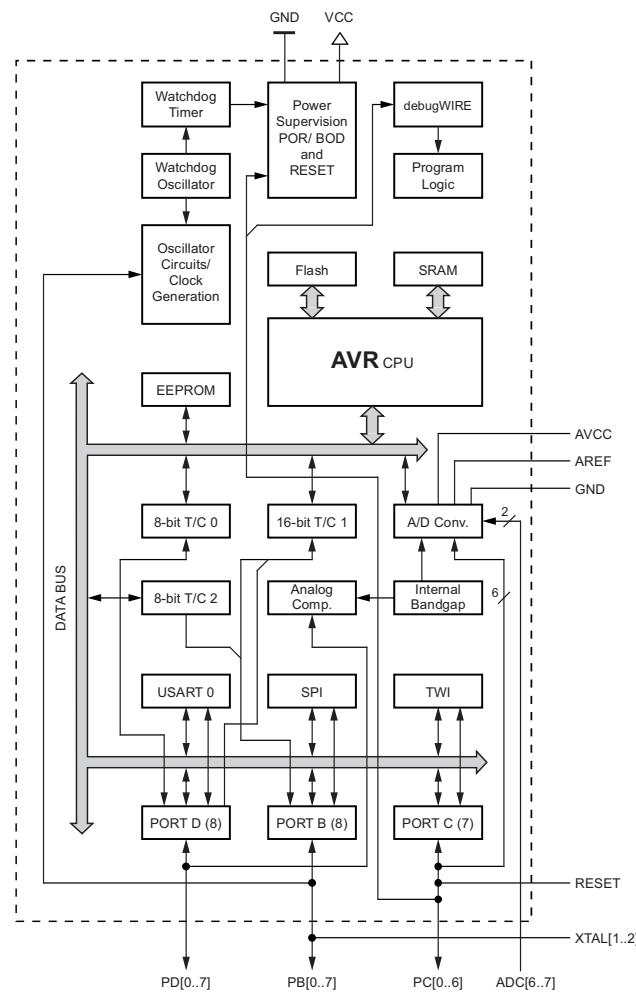


2. Overview

The Atmel® ATmega328P is a low-power CMOS 8-bit microcontroller based on the AVR® enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega328P achieves throughputs approaching 1MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

2.1 Block Diagram

Figure 2-1. Block Diagram



A.2 Audio-Verstärker PAM8403

Nachfolgend ein Auszug aus dem Datenblatt mit den wichtigsten technischen Daten, der Pinbelegung und einem Blockdiagramm. Die vollständige Dokumentation umfasst über 11 Seiten und kann unter nachfolgender Adresse abgerufen werden.

(Nov. 2012). „PAM8403 Datasheet“, Diodes, Adresse: <https://www.diodes.com/assets/Datasheets/PAM8403.pdf>

Description

The PAM8403 is a 3W, class-D audio amplifier. It offers low THD+N, allowing it to achieve high-quality sound reproduction. The new filterless architecture allows the device to drive the speaker directly, requiring no low-pass output filters, thus saving system cost and PCB area.

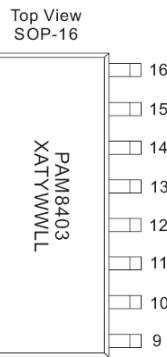
With the same numbers of external components, the efficiency of the PAM8403 is much better than that of Class-AB cousins. It can extend the battery life, which makes it well-suited for portable applications.

The PAM8403 is available in SOP-16 package.

Features

- 3W Output at 10% THD with a 4Ω Load and 5V Power Supply
- Filterless, Low Quiescent Current and Low EMI
- Low THD+N
- Superior Low Noise
- Efficiency up to 90%
- Short Circuit Protection
- Thermal Shutdown
- Few External Components to Save the Space and Cost
- Pb-Free Package

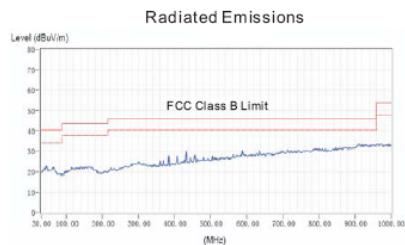
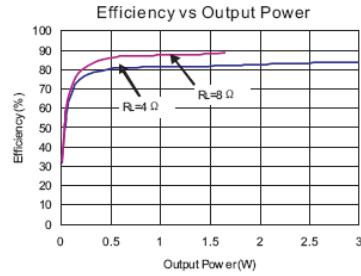
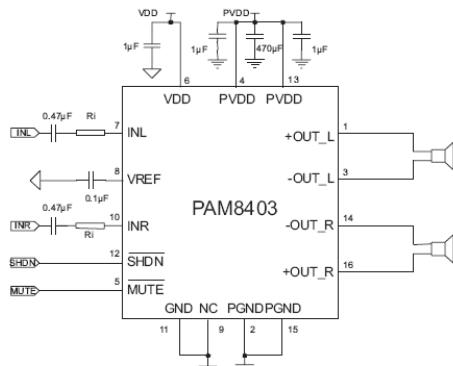
Pin Assignments



Applications

- LCD Monitors / TV Projectors
- Notebook Computers
- Portable Speakers
- Portable DVD Players, Game Machines
- Cellular Phones/Speaker Phones

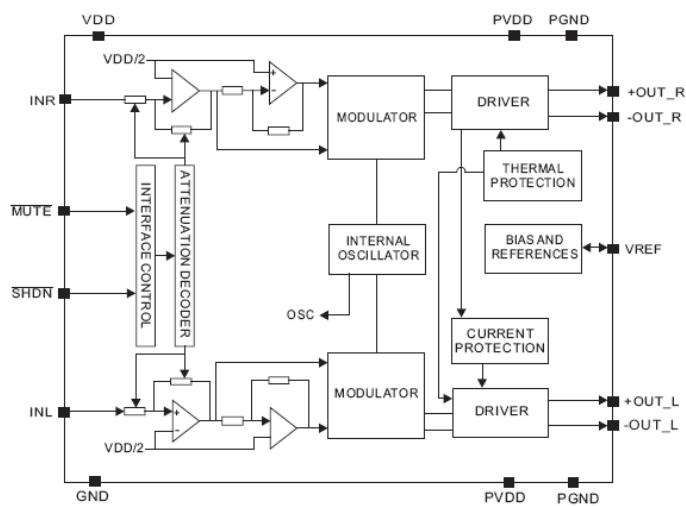
Typical Applications Circuit



Pin Descriptions

Pin Number	Pin Name	Function
1	+OUT_L	Left Channel Positive Output
2	PGND	Power GND
3	-OUT_L	Left Channel Negative Output
4	PVDD	Power VDD
5	MUTE	Mute Control Input (active low)
6	VDD	Analog VDD
7	INL	Left Channel Input
8	VREF	Internal analog reference, connect a bypass capacitor from VREF to GND.
9	NC	No Connect
10	INR	Right Channel Input
11	GND	Analog GND
12	SHDN	Shutdown Control Input (active low)
13	PVDD	Power VDD
14	-OUT_R	Right Channel Negative Output
15	PGND	Power GND
16	+OUT_R	Right Channel Positive Output

Functional Block Diagram





A Product Line of
Diodes Incorporated



PAM8403

Absolute Maximum Ratings (@ $T_A = +25^\circ\text{C}$, unless otherwise specified.)

These are stress ratings only and functional operation is not implied. Exposure to absolute maximum ratings for prolonged time periods may affect device reliability. All voltages are with respect to ground.

Parameter	Rating	Unit
Supply Voltage	6.0	V
Input Voltage	-0.3 to $V_{DD} + 0.3\text{V}$	
Operation Temperature Range	-40 to +85	
Maximum Junction Temperature	150	
Operation Junction Temperature	-40 to +125	
Storage Temperature	-65 to +150	
Soldering Temperature	300, 5 sec	

Recommended Operating Conditions (@ $T_A = +25^\circ\text{C}$, unless otherwise specified.)

Parameter	Rating	Unit
Supply Voltage Range	2.5 to 5.5	V
Operation Temperature Range	-40 to +85	°C
Junction Temperature Range	-40 to +125	°C

Thermal Information

Parameter	Package	Symbol	Max	Unit
Thermal Resistance (Junction to Ambient)	SOP-16	θ_{JA}	110	
Thermal Resistance (Junction to Case)	SOP-16	θ_{JC}	23	°C/W

A Product Line of
Diodes Incorporated

PAM8403

Electrical Characteristics (@ $T_A = +25^\circ\text{C}$, $V_{DD} = 5\text{V}$, Gain = 24dB, $R_L = 8\Omega$, unless otherwise specified.)

Symbol	Parameter	Test Conditions		Min	Typ	Max	Units
V_{DD}	Supply Voltage			2.5		5.5	V
P_o	Output Power	THD+N = 10%, f = 1KHz, $R_L = 4\Omega$	$V_{DD} = 5.0\text{V}$	3.2			W
			$V_{DD} = 3.6\text{V}$	1.6			
			$V_{DD} = 3.2\text{V}$	1.3			
		THD+N = 1%, f = 1KHz, $R_L = 4\Omega$	$V_{DD} = 5.0\text{V}$	2.5			W
			$V_{DD} = 3.6\text{V}$	1.3			
			$V_{DD} = 3.2\text{V}$	0.85			
		THD+N = 10%, f = 1KHz, $R_L = 8\Omega$	$V_{DD} = 5.0\text{V}$	1.8			W
			$V_{DD} = 3.6\text{V}$	0.9			
			$V_{DD} = 3.2\text{V}$	0.6			
		THD+N = 1%, f = 1KHz, $R_L = 8\Omega$	$V_{DD} = 5.0\text{V}$	1.4			W
			$V_{DD} = 3.6\text{V}$	0.72			
			$V_{DD} = 3.2\text{V}$	0.45			
THD+N	Total Harmonic Distortion Plus Noise	$V_{DD} = 5.0\text{V}$, $P_o = 1\text{W}$, $R_L = 8\Omega$	$f = 1\text{kHz}$	0.15			%
		$V_{DD} = 3.6\text{V}$, $P_o = 0.1\text{W}$, $R_L = 8\Omega$		0.11			
		$V_{DD} = 5.0\text{V}$, $P_o = 0.5\text{W}$, $R_L = 4\Omega$	$f = 1\text{kHz}$	0.15			%
		$V_{DD} = 3.6\text{V}$, $P_o = 0.2\text{W}$, $R_L = 4\Omega$		0.11			
G_v	Closed Loop Gain	$V_{DD} = 3\text{V}$ to 5V			24		dB
PSRR	Power Supply Ripple Rejection	$V_{DD} = 5.0\text{V}$, Inputs AC-Grounded with $C_{IN} = 0.47\mu\text{F}$	$f = 100\text{Hz}$		-59		dB
			$f = 1\text{kHz}$		-58		
C_s	Crosstalk	$V_{DD} = 5.0\text{V}$, $P_o = 0.5\text{W}$, $R_L = 8\Omega$, $G_v = 20\text{db}$	$f = 1\text{kHz}$		-95		dB
SNR	Signal-to-Noise Ratio	$V_{DD} = 5.0\text{V}$, $V_{ORMS} = 1\text{V}$, $G_v = 20\text{db}$	$f = 1\text{kHz}$		80		dB
V_N	Output Noise	$V_{DD} = 5.0\text{V}$, Inputs AC-Grounded with $C_{IN} = 0.47\mu\text{F}$	No A-Weighting	100			μV
			A-Weighting	150			
Dyn	Dynamic Range	$V_{DD} = 5.0\text{V}$, THD = 1%	$f = 1\text{kHz}$		90		dB
η	Efficiency	$R_L = 8\Omega$, THD = 10%	$f = 1\text{kHz}$		87		%
		$R_L = 4\Omega$, THD = 10%			83		
I_Q	Quiescent Current	$V_{DD} = 5.0\text{V}$	No Load		16		mA
		$V_{DD} = 3.6\text{V}$			10		
		$V_{DD} = 3.0\text{V}$			8		
I_{MUTE}	Muting Current	$V_{DD} = 5.0\text{V}$	$V_{MUTE} = 0.3\text{V}$		3.5		mA
I_{SD}	Shutdown Current	$V_{DD} = 2.5\text{V}$ to 5.5V	$V_{SD} = 0.3\text{V}$		< 1		μA
$R_{DS(ON)}$	Static Drain-to-Source On-State Resistor	$I_{DS} = 500\text{mA}$, $V_{GS} = 5\text{V}$	PMOS		180		$\text{m}\Omega$
			NMOS		140		
f_{SW}	Switching Frequency	$V_{DD} = 3.0\text{V}$ to 5.0V			260		kHz
V_{OS}	Output Offset Voltage	$V_{IN} = 0\text{V}$, $V_{DD} = 5.0\text{V}$			10		mV
V_{IH}	Enable Input High Voltage	$V_{DD} = 5.0\text{V}$		1.5	1.4		V
V_{IL}	Enable Input Low Voltage	$V_{DD} = 5.0\text{V}$			0.7	0.4	
V_{IH}	MUTE Input High Voltage	$V_{DD} = 5.0\text{V}$		1.5	1.4		V
V_{IL}	MUTE Input Low Voltage	$V_{DD} = 5.0\text{V}$			0.7	0.4	
OTP	Over Temperature Protection	No Load, Junction Temperature	$V_{DD} = 5.0\text{V}$		140		V
OTH	Over Temperature Hysteresis				30		V

A.3 Spannungsregler LMR33630CDDAR

Nachfolgend ein Auszug aus dem Datenblatt mit den wichtigsten technischen Daten, der Pinbelegung und einem Blockdiagramm. Die vollständige Dokumentation umfasst über 11 Seiten und kann unter nachfolgender Adresse abgerufen werden.

(März 2019). „LMR33630 Datasheet“, Texas Instruments, Adresse: <https://www.ti.com/lit/ds/symlink/lmr33630.pdf>

LMR33630 SIMPLE SWITCHER® 3.8-V to 36-V, 3-A synchronous step-down voltage converter

1 Features

- Configured for rugged industrial applications
 - Input voltage range: 3.8 V to 36 V
 - Output voltage range: 1 V to 24 V
 - Output current: 3 A
 - 75-mΩ/50-mΩ R_{DS-ON} power MOSFETs
 - Peak-current-mode control
 - Short minimum on-time of 68 ns
 - Frequency: 400 kHz, 1.4 MHz, 2.1 MHz
 - Junction temperature range -40°C to $+125^{\circ}\text{C}$
 - Low EMI and low switching noise
 - Integrated compensation network
- High-efficiency solution
 - Peak efficiency $> 95\%$
 - Low shutdown quiescent current of 5 μA
 - Low operating quiescent current of 25 μA
- Flexible system interface
 - Power-good flag and precision enable
- Create a custom design using the LMR33630 with the [WEBENCH® Power Designer](#)

2 Applications

- Motor drive systems: drones, AC inverters, VF drives, servos
- Factory and building automation systems: PLC CPU, HVAC control, elevator control
- General purpose wide V_{IN} power supplies

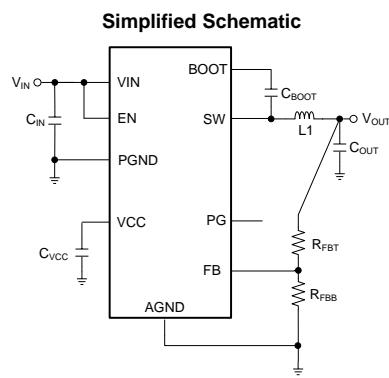
3 Description

The LMR33630 SIMPLE SWITCHER® regulator is an easy-to-use, synchronous, step-down DC/DC converter that delivers best-in-class efficiency for rugged industrial applications. The LMR33630 drives up to 3 A of load current from an input of up to 36 V. The LMR33630 provides high light load efficiency and output accuracy in a very small solution size. Features such as a power-good flag and precision enable provide both flexible and easy-to-use solutions for a wide range of applications. The LMR33630 automatically folds back frequency at light load to improve efficiency. Integration eliminates most external components and provides a pinout designed for simple PCB layout. Protection features include thermal shutdown, input undervoltage lockout, cycle-by-cycle current limit, and hiccup short-circuit protection. The LMR33630 is available in an 8-pin HSOIC package and in a 12-pin 3 mm × 2 mm VQFN package with wettable flanks. This device is also available in an AEC-Q100-qualified version.

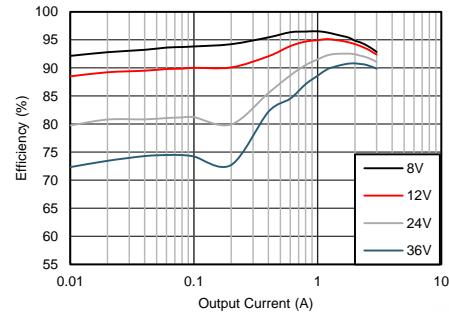
Device Information⁽¹⁾

PART NUMBER	PACKAGE	BODY SIZE (NOM)
LMR33630	HSOIC (8)	5.00 mm × 4.00 mm
LMR33630	VQFN (12)	3.00 mm × 2.00 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

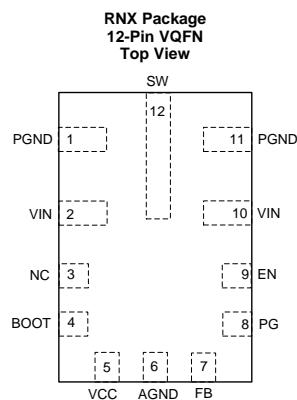
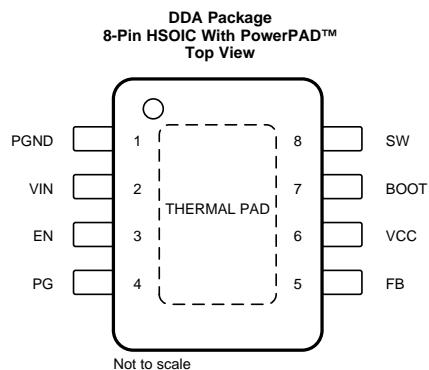


Efficiency vs Output Current
 $V_{OUT} = 5 \text{ V}, 400 \text{ kHz}, \text{VQFN}$



An IMPORTANT NOTICE at the end of this data sheet addresses availability, warranty, changes, use in safety-critical applications, intellectual property matters and other important disclaimers. PRODUCTION DATA.

6 Pin Configuration and Functions



7 Specifications

7.1 Absolute Maximum Ratings

Over the recommended operating junction temperature range⁽¹⁾

PARAMETER		MIN	MAX	UNIT
Voltages	VIN to PGND	-0.3	38	V
	EN to AGND ⁽²⁾	-0.3	$V_{IN} + 0.3$	
	FB to AGND	-0.3	5.5	
	PG to AGND ⁽²⁾	0	22	
	AGND to PGND	-0.3	0.3	
	SW to PGND	-0.3	$V_{IN} + 0.3$	
	SW to PGND less than 100-ns transients	-3.5	38	
	BOOT to SW	-0.3	5.5	
	VCC to AGND ⁽³⁾	-0.3	5.5	
	T _J	Junction temperature ⁽⁴⁾	-40	150
T _{stg}	Storage temperature	-55	150	°C

(1) Stresses beyond those listed under *Absolute Maximum Ratings* may cause permanent damage to the device. These are stress ratings only, which do not imply functional operation of the device at these or any other conditions beyond those indicated under *Recommended Operating Conditions*. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

(2) The voltage on this pin must not exceed the voltage on the VIN pin by more than 0.3 V

(3) Under some operating conditions the VCC LDO voltage may increase beyond 5.5V.

(4) Operating at junction temperatures greater than 125°C, although possible, degrades the lifetime of the device.

7.2 ESD Ratings

V _(ESD)	Electrostatic discharge	Human-body model (HBM) ⁽¹⁾	VALUE	UNIT
		Charged-device model (CDM) ⁽²⁾	±750	V

(1) JEDEC document JEP155 states that 500-V HBM allows safe manufacturing with a standard ESD control process.

(2) JEDEC document JEP157 states that 250-V CDM allows safe manufacturing with a standard ESD control process.

7.3 Recommended Operating Conditions

Over the recommended operating temperature range of -40 °C to 125 °C (unless otherwise noted)⁽¹⁾

Input voltage	VIN to PGND	MIN	MAX	UNIT
	EN ⁽²⁾	0	V_{IN}	V
	PG ⁽²⁾	0	18	
	V _{OUT} ⁽³⁾	1	24	V
Output current	I _{OUT}	0	3	A

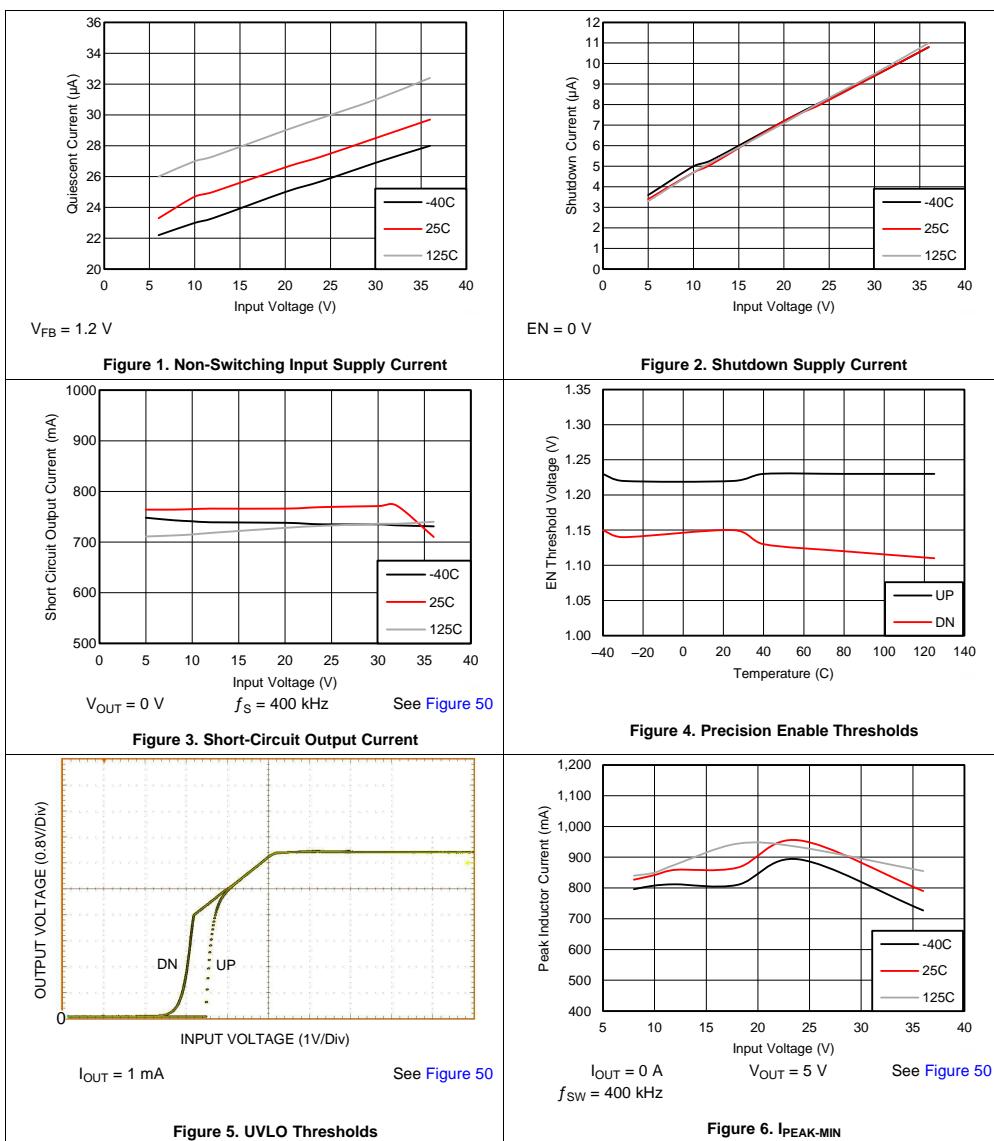
(1) Recommended operating conditions indicate conditions for which the device is intended to be functional, but do not ensure specific performance limits. For ensured specifications, see *Electrical Characteristics*.

(2) The voltage on this pin must not exceed the voltage on the VIN pin by more than 0.3 V.

(3) The maximum output voltage can be extended to 95% of V_{IN}; contact TI for details. Under no conditions should the output voltage be allowed to fall below zero volts.

7.8 Typical Characteristics

Unless otherwise specified the following conditions apply: $T_A = 25^\circ\text{C}$ and $V_{IN} = 12\text{ V}$



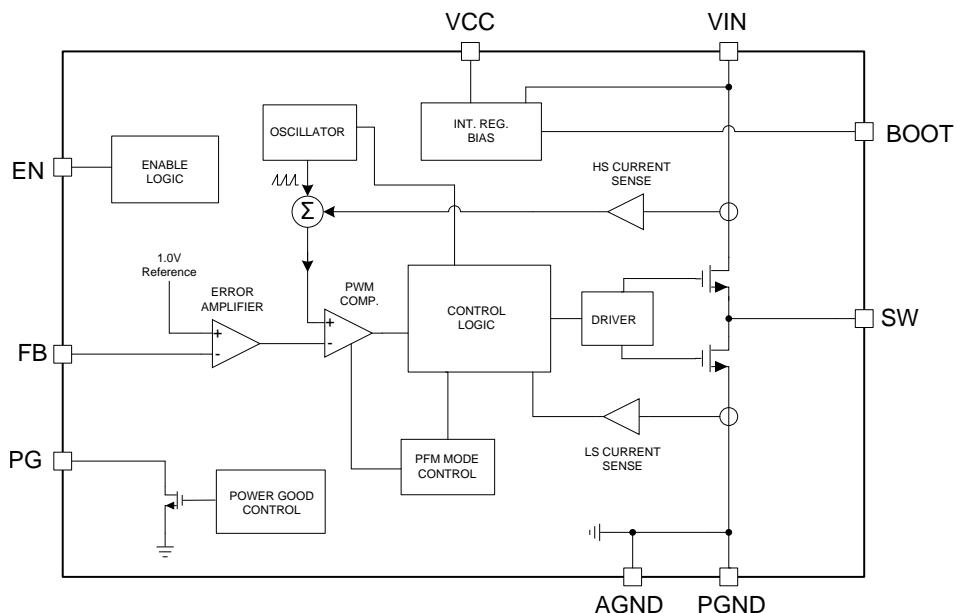
8 Detailed Description

8.1 Overview

The LMR33630 is a synchronous peak-current-mode buck regulator designed for a wide variety of industrial applications. Advanced high speed circuitry allows the device to regulate from an input voltage of 20 V, while providing an output voltage of 3.3 V at a switching frequency of 2.1 MHz. The innovative architecture allows the device to regulate a 3.3 V output from an input of only 3.8 V. The regulator automatically switches modes between PFM and PWM depending on load. At heavy load, the device operates in PWM at a constant switching frequency. At light loads the mode changes to PFM, with diode emulation allowing DCM. This reduces the input supply current and keeps efficiency high. The device features internal loop compensation which reduces design time and requires fewer external components than externally compensated regulators.

The LMR33630 is available in an ultra-miniature VQFN package with wettable flanks. This package features extremely small parasitic inductance and resistance, enabling very high efficiency while minimizing switch node ringing and dramatically reducing EMI. The VIN/PGND pin layout is symmetrical on either side of the VQFN package. This allows the input current magnetic fields to partially cancel, resulting in reduced EMI generation.

8.2 Functional Block Diagram



Copyright © 2017, Texas Instruments Incorporated

B Fertigungsdokumentation

B.1 Code-Listing

```

1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using System.Collections.ObjectModel;
5 using System.Collections.Specialized;
6 using System.ComponentModel;
7 using System.Linq;
8 using System.Text;
9 using System.Threading.Tasks;
10 using LibVLCSharp.Shared;
11 using PiBell.Classes;
12 using Xamarin.Forms;
13 using LibVLCSharp.Forms.Shared;
14 using Microsoft.AppCenter.Push;
15 using Microsoft.AppCenter;
16
17 namespace PiBell
18 {
19     public partial class MainPage : ContentPage
20     {
21         private Player _player;
22         private Streamer _streamer;
23         private double _width, _height;
24         private bool _wasConnected;
25         private float _prevPosition;
26
27         public MainPage()
28         {
29             InitializeComponent();
30             RegenerateVideoView();
31             Core.Initialize();
32
33             _player = new Player();
34             _streamer = new Streamer();
35
36             if (!AppCenter.Configured)
37                 Push.PushNotificationReceived += async (sender, e) =>
38                 {
39 #if DEBUG
40                     Console.WriteLine("DEBUG - Push-Notification received");
41 #endif
42                     foreach (string key in e.CustomData.Keys)
43                     {
44 #if DEBUG
45                         Console.WriteLine($"DEBUG - Custom Data: {key}:{e.
46                         CustomData[key]}");
47 #endif
48                         if (key == "mrl")
49                             _player.Mrl = e.CustomData[key].ToString();
50
51                         if (await DisplayAlert("Incoming Call", "Connect now?", "
52                         Yes", "No"))
53                         {
54                             _player.StartCall();
55                             _streamer.StartCall();
56                         };
57                         MessagingCenter.Subscribe<string>(this, "OnPause", app =>
58                         {

```

```

59             _wasConnected = _player.MediaPlayer.IsPlaying;
60             _player.MediaPlayer.Pause();
61             _prevPosition = _player.MediaPlayer.Position;
62             _player.MediaPlayer.Stop();
63
64             MainGrid.Children.Remove(VideoView);
65         });
66
67         MessagingCenter.Subscribe<string>(this, "OnRestart", app =>
68     {
69         RegenerateVideoView();
70         VideoView.MediaPlayer = _player.MediaPlayer;
71         if (_wasConnected)
72     {
73             _player.MediaPlayer.Play();
74             _player.MediaPlayer.Position = _prevPosition;
75         }
76
77         _prevPosition = 0;
78     });
79
80     _player = new Player();
81     _streamer = new Streamer();
82 //_player.MediaPlayer.Volume = 0;
83
84     BindingContext = _player;
85 }
86
87     void RegenerateVideoView()
88     {
89         VideoView = new VideoView();
90         MainGrid.Children.Add(VideoView, 0, 1);
91     }
92
93     protected override void OnAppearing()
94     {
95         base.OnAppearing();
96         _player.MediaPlayer = new MediaPlayer(_player.LibVlc);
97     }
98
99     private void BtToggleSpeaker_Clicked(object sender, EventArgs e)
100    {
101        VisualStateManager.GoToState((ImageButton) sender, _player.
102        ToggleSpeaker() ? "Unmute" : "Mute");
103    }
104
105     private void BtToggleMic_Clicked(object sender, EventArgs e)
106    {
107        VisualStateManager.GoToState((ImageButton) sender, _streamer.
108        ToggleMic() ? "Unmute" : "Mute");
109    }
110
111     private void BtConnect_Clicked(object sender, EventArgs e)
112    {
113        _player.StartCall();
114        _streamer.StartCall("192.168.43.78");
115    }
116
117     private void BtDisconnect_Clicked(object sender, EventArgs e)
118    {

```

```
117         _player.EndCall();
118         _streamer.EndCall();
119     }
120
121     protected override void OnSizeAllocated(double width, double height)
122     {
123         base.OnSizeAllocated(width, height);
124         if (_width != width || _height != height)
125         {
126             _width = width;
127             _height = height;
128             if (width < height) //Portrait
129             {
130                 MainGrid.ColumnDefinitions.Clear();
131                 MainGrid.ColumnDefinitions.Add(new ColumnDefinition
132                     {Width = new GridLength(1, GridUnitType.Star)} );
133
134                 MainGrid.RowDefinitions.Clear();
135                 MainGrid.RowDefinitions.Add(new RowDefinition
136                     {Height = new GridLength(0.7, GridUnitType.Star)} );
137                 MainGrid.RowDefinitions.Add(new RowDefinition
138                     {Height = new GridLength(3, GridUnitType.Star)} );
139                 MainGrid.RowDefinitions.Add(new RowDefinition
140                     {Height = new GridLength(2, GridUnitType.Star)} );
141
142                 Grid.SetColumnSpan(EditMrl, 1);
143                 Grid.SetColumn(ButtonGrid, 0);
144                 Grid.SetRow(ButtonGrid, 2);
145             }
146             else //Landscape
147             {
148                 MainGrid.ColumnDefinitions.Clear();
149                 MainGrid.ColumnDefinitions.Add(new ColumnDefinition
150                     {Width = new GridLength(2, GridUnitType.Star)} );
151                 MainGrid.ColumnDefinitions.Add(new ColumnDefinition
152                     {Width = new GridLength(1, GridUnitType.Star)} );
153
154                 MainGrid.RowDefinitions.Clear();
155                 MainGrid.RowDefinitions.Add(new RowDefinition
156                     {Height = new GridLength(0.7, GridUnitType.Star)} );
157                 MainGrid.RowDefinitions.Add(new RowDefinition
158                     {Height = new GridLength(5, GridUnitType.Star)} );
159
160                 Grid.SetColumnSpan(EditMrl, 2);
161                 Grid.SetColumn(ButtonGrid, 1);
162                 Grid.SetRow(ButtonGrid, 1);
163             }
164         }
165     }
166 }
167 }
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5
6 using UIKit;
7 using System.Net.Sockets;
8 using System.Net;
9 using static PiBell.Classes.Streamer;
10 using System.Threading;
11 using Plugin.AudioRecorder;
12
13 namespace PiBell.iOS
14 {
15     class AudioRecordingService : IAudioRecordingService
16     {
17         Socket socket;
18         NetworkStream stream;
19         Thread recordingThread;
20         AudioRecorderService recorder;
21
22         public AudioRecordingService()
23         {
24             socket = new Socket(SocketType.Dgram, ProtocolType.Udp);
25             stream = new NetworkStream(socket);
26             recorder = new AudioRecorderService();
27             recordingThread = new Thread(RecordingFunction);
28         }
29
30         private void RecordingFunction()
31         {
32             stream = new NetworkStream(socket);
33             recorder.StartRecording();
34             while(true)
35             {
36                 try
37                 {
38                     recorder.GetAudioFileStream().CopyTo(stream);
39                 }
40                 catch(ThreadAbortException)
41                 {
42                     break;
43                 }
44                 catch(Exception e)
45                 {
46                     Console.WriteLine(e.Message);
47                     break;
48                 }
49             }
50             recorder.StopRecording();
51         }
52
53         public void Start(IPEndPoint target)
54         {
55             socket.Connect(target);
56             recordingThread.Start();
57         }
58
59         public void Stop()
60         {
```

File - C:\Users\Grain\source\repos\gratux\PiBell\PiBell\PiBell.iOS\AudioRecordingService.cs

```
61         recordingThread.Abort();
62         socket.Close();
63     }
64
65     public bool ToggleMic()
66     {
67         throw new NotImplementedException();
68     }
69 }
70 }
```

```

1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.IO;
5  using System.Net;
6  using System.Net.Sockets;
7  using System.Runtime.CompilerServices;
8  using System.Runtime.InteropServices;
9  using System.Text;
10 using System.Threading;
11 using LibVLCSharp.Shared;
12 using PiBell.Annotations;
13 using Xamarin.Forms;
14
15 namespace PiBell.Classes
16 {
17     public class Player : INotifyPropertyChanged
18     {
19         public event PropertyChangedEventHandler PropertyChanged;
20
21         [NotifyPropertyChangedInvocator]
22         protected virtual void OnPropertyChanged([CallerMemberName] string
23 propertyName = null)
24         {
25             PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(
26 propertyName));
27         }
28
29         private bool _speakerActive;
30
31         public Player(string mrl = "https://www.freedesktop.org/software/gstreamer-sdk/data/media/sintel\_trailer-480p.webm")
32         {
33             LibVlc = new LibVLC();
34             MediaPlayer = new MediaPlayer(LibVlc) { NetworkCaching = 0 };
35             Mrl = mrl;
36         }
37
38         public bool ToggleSpeaker()
39         {
40             MediaPlayer.Volume = _speakerActive ? 0 : 100;
41             return _speakerActive = !_speakerActive;
42         }
43
44         public void StartCall()
45         {
46             MediaPlayer.Media = new Media(LibVlc, Mrl, FromType.FromLocation
47 );
48             MediaPlayer.Play();
49             MediaPlayer.Volume = _speakerActive ? 100 : 0;
50         }
51
52         public void EndCall()
53         {
54             MediaPlayer.Stop();
55         }
56
57         private string _mrl;
58         public string Mrl
59         {

```

```
57         get => _mrl;
58         set
59     {
60         _mrl = value;
61         OnPropertyChanged();
62     }
63 }
64
65     private LibVLC _libVlc;
66     public LibVLC LibVlc
67     {
68         get => _libVlc;
69         set
70     {
71         _libVlc = value;
72         OnPropertyChanged();
73     }
74 }
75
76     private MediaPlayer _mediaPlayer;
77     public MediaPlayer MediaPlayer
78     {
79         get => _mediaPlayer;
80         set
81     {
82         _mediaPlayer = value;
83         OnPropertyChanged();
84     }
85 }
86 }
87
88     public class Streamer
89     {
90         private IAudioRecordingService Service { get; set; }
91
92         public Streamer()
93     {
94             Service = DependencyService.Get<IAudioRecordingService>();
95         }
96
97         public void StartCall(string targetIp = "127.0.0.1", int targetPort
= 10000)
98     {
99             Service.Start(new IPEndPoint(IPAddress.Parse(targetIp),
targetPort));
100        }
101
102        public void EndCall()
103    {
104        Service.Stop();
105    }
106
107        public bool ToggleMic()
108    {
109        return Service.ToggleMic();
110    }
111
112        public interface IAudioRecordingService
113    {
114        void Start(IPEndPoint target);
```

```
115         void Stop();  
116         bool ToggleMic();  
117     }  
118 }  
119 }
```

```
1 using System;
2 using System.Diagnostics;
3 using LibVLCSharp.Shared;
4 using Microsoft.AppCenter;
5 using Microsoft.AppCenter.Analytics;
6 using Microsoft.AppCenter.Crashes;
7 using Microsoft.AppCenter.Push;
8 using Xamarin.Forms;
9 using Xamarin.Forms.Xaml;
10
11 [assembly: XamlCompilation(XamlCompilationOptions.Compile)]
12
13 namespace PiBell
14 {
15     public partial class App : Application
16     {
17         public App()
18         {
19             InitializeComponent();
20             Core.Initialize();
21             MainPage = new MainPage();
22
23
24             AppCenter.LogLevel = Microsoft.AppCenter.LogLevel.Verbose;
25             AppCenter.Start("android={Your Android App secret here};" +
26                             "uwp={Your UWP App secret here};" +
27                             "ios={Your iOS App secret here}",
28                             typeof(Analytics), typeof(Crashes), typeof(Push));
29         }
30
31         protected override void OnStart()
32         {
33             // Handle when your app starts
34         }
35
36         protected override void OnSleep()
37         {
38             // Handle when your app sleeps
39         }
40
41         protected override void OnResume()
42         {
43             // Handle when your app resumes
44         }
45     }
46 }
```

```
1 using System;
2 using Android.App;
3 using Android.Content.PM;
4 using Android.Runtime;
5 using Android.Views;
6 using Android.Widget;
7 using Android.OS;
8 using LibVLCSharp.Forms.Shared;
9 using LibVLCSharp.Shared;
10 using Xamarin.Forms;
11 using Microsoft.AppCenter.Push;
12 using Microsoft.AppCenter;
13 using Android.Util;
14 using Android;
15 using Android.Support.V4.Content;
16 using Android.Support.V4.App;
17
18 namespace PiBell.Droid
19 {
20     [Activity(Label = "PiBell", Icon = "@mipmap/icon", Theme = "@style/MainTheme", MainLauncher = true,
21             ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
22     public class MainActivity : global::Xamarin.Forms.Platform.Android.
FormsAppCompatActivity
23     {
24         protected override void OnCreate(Bundle savedInstanceState)
25         {
26             TabLayoutResource = Resource.Layout.Tabbar;
27             ToolbarResource = Resource.Layout.Toolbar;
28
29             base.OnCreate(savedInstanceState);
30             global::Xamarin.Forms.Forms.Init(this, savedInstanceState);
31             LoadApplication(new App());
32
33             if (CheckSelfPermission(Manifest.Permission.RecordAudio) != Permission.Granted)
34             {
35                 RequestPermissions(new[] { Manifest.Permission.RecordAudio
}, 1);
36             }
37         }
38
39         protected override void OnPause()
40         {
41             base.OnPause();
42             MessagingCenter.Send("app", "OnPause");
43         }
44
45         protected override void OnRestart()
46         {
47             base.OnRestart();
48             MessagingCenter.Send("app", "OnRestart");
49         }
50     }
51 }
```

```

1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4  using System.Linq;
5  using System.Text;
6  using System.Threading;
7  using Android.App;
8  using Android.Content;
9  using Android.Media;
10 using Android.OS;
11 using Android.Runtime;
12 using Android.Views;
13 using Android.Widget;
14 using Java.Nio;
15 using Xamarin.Forms;
16 using PiBell.Classes;
17 using System.Net.Sockets;
18 using System.Net;
19
20 [assembly: Dependency(typeof(PiBell.Droid.AudioRecordingService))]  

21
22 namespace PiBell.Droid
23 {
24     class AudioRecordingService : Streamer.IAudioRecordingService
25     {
26         private Thread _recordingThread;
27         private volatile byte[] _audioBuffer;
28         private UdpClient _udp;
29         private IPEndPoint _target;
30         private bool _micActive;
31
32         public AudioRecordingService()
33         {
34             _audioBuffer = new byte[AudioRecord.GetMinBufferSize(48000,
ChannelIn.Mono, Android.Media.Encoding.Pcm16bit) * 3];
35             // _audioRecord = new AudioRecord(AudioSource.Mic, 48000,
ChannelIn.Mono, Android.Media.Encoding.Pcm16bit, _audioBuffer.Length);
36             _udp = new UdpClient();
37         }
38
39         public void Start(IPEndPoint target)
40         {
41             this._target = target;
42             _recordingThread = new Thread(RecordAudio) { IsBackground = true
};  

43             _recordingThread.Start();
44 #if DEBUG
45             Toast.MakeText(Android.App.Application.Context, "Started
Recording", ToastLength.Short).Show();
46 #endif
47         }
48
49         private void RecordAudio()
50         {
51             var audioRecord = new AudioRecord(AudioSource.Mic, 48000,
ChannelIn.Mono, Android.Media.Encoding.Pcm16bit, _audioBuffer.Length);
52             audioRecord.StartRecording();
53             while (true)
54             {
55                 try

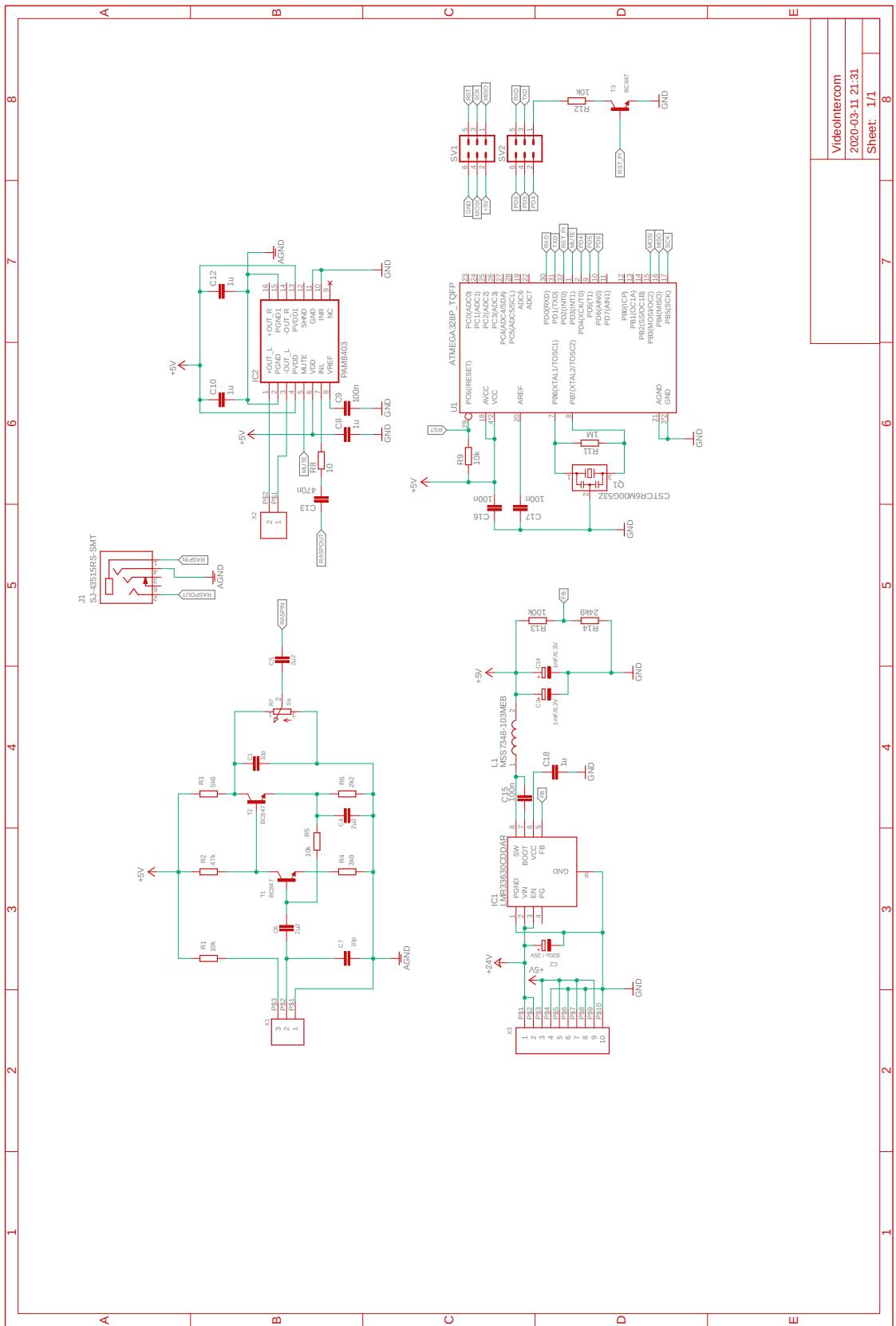
```

```
56             {
57                 audioRecord.Read(_audioBuffer, 0, _audioBuffer.Length);
58                 _udp.Send(_micActive ? _audioBuffer : new byte[
59                     _audioBuffer.Length], _audioBuffer.Length, _target);
60             }
61         catch (ThreadAbortException)
62         {
63             //before the thread stops
64             audioRecord.Stop();
65             audioRecord.Release();
66             break;
67         }
68         catch (Exception e)
69         {
70             Console.WriteLine($"AudioRecord: {e.Message}");
71             break;
72         }
73     }
74
75     public void Stop()
76     {
77         _recordingThread.Abort();
78 #if DEBUG
79         Toast.MakeText(Android.App.Application.Context, "Stopped
Recording", ToastLength.Short).Show();
80 #endif
81     }
82
83     public bool ToggleMic()
84     {
85         return _micActive = !_micActive;
86     }
87 }
88 }
```

```
1 <?xml version="1.0" encoding="utf-8"?>
2
3 <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
4     xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
5     xmlns:local="clr-namespace:PiBell"
6     xmlns:piBell="clr-namespace:PiBell;assembly=PiBell"
7     xmlns:shared="clr-namespace:LibVLCSharp.Forms.Shared;assembly=
LibVLCSharp.Forms"
8         x:Class="PiBell.MainPage"
9         Title="Main Page"
10        BackgroundColor="#ff1b1b1b">
11
12    <ContentPage.Content>
13        <Grid Margin="10" x:Name="MainGrid">
14            <Grid.RowDefinitions>
15                <RowDefinition Height=".7*" />
16                <RowDefinition Height="3*" />
17                <RowDefinition Height="2*" />
18            </Grid.RowDefinitions>
19            <Grid.ColumnDefinitions>
20                <ColumnDefinition Width="*" />
21            </Grid.ColumnDefinitions>
22
23            <Entry x:Name="EditMrl" Text="{Binding Mrl, Mode=TwoWay}" Grid.
Row="0" Grid.Column="0" TextColor="White" />
24
25            <shared:VideoView x:Name="VideoView" Grid.Column="0" Grid.Row="1
" MediaPlayer="{Binding MediaPlayer}"/>
26
27            <Grid x:Name="ButtonGrid" Grid.Row="2" Grid.Column="0">
28                <Grid.ColumnDefinitions>
29                    <ColumnDefinition Width="*" />
30                    <ColumnDefinition Width="*" />
31                </Grid.ColumnDefinitions>
32                <Grid.RowDefinitions>
33                    <RowDefinition Height="*" />
34                    <RowDefinition Height="*" />
35                </Grid.RowDefinitions>
36
37            <ImageButton x:Name="BtConnect" Source="call.png"
BackgroundColor="LimeGreen" Grid.Row="0"
                           Grid.Column="0" Margin="10"
                           Clicked="BtConnect_Clicked"/>
38            <ImageButton x:Name="BtDisconnect" Source="Hangup.png"
BackgroundColor="Red" Grid.Row="0"
                           Grid.Column="1" Margin="10"
                           Clicked="BtDisconnect_Clicked"/>
39
40            <ImageButton x:Name="BtToggleSpeaker" Source="SpeakerMute.png
" BackgroundColor="Transparent"
                           Grid.Row="1" Grid.Column="0" Margin="10"
                           Clicked="BtToggleSpeaker_Clicked">
41                <VisualStateManager.VisualStateGroups>
42                    <VisualStateGroup x:Name="SpeakerStates">
43                        <VisualState Name="Mute">
44                            <VisualState.Setters>
45                                <Setter Property="Source"
Value="SpeakerMute.png" />
46                            </VisualState.Setters>
47                        </VisualState>
48                    </VisualStateGroup>
49                </VisualStateManager.VisualStateGroups>
50            </ImageButton>
51
52
53
54
```

```
55                     <VisualState Name="Unmute">
56                         <VisualState.Setters>
57                             <Setter Property="Source"
58                                 Value="SpeakerUnmute.png" />
59                         </VisualState.Setters>
60                     </VisualState>
61                 </VisualStateManager.VisualStateGroups>
62             </ImageButton>
63
64             <ImageButton x:Name="BtToggleMic" Source="MicMute.png"
65             BackgroundColor="Transparent"
66                 Grid.Row="1" Grid.Column="1" Margin="10"
67                 Clicked="BtToggleMic_Clicked">
68                 <VisualStateManager.VisualStateGroups>
69                     <VisualStateGroup x:Name="MicStates">
70                         <VisualState Name="Mute">
71                             <VisualState.Setters>
72                                 <Setter Property="Source"
73                                     Value="MicMute.png" />
74                             </VisualState.Setters>
75                         </VisualState>
76                         <VisualState Name="Unmute">
77                             <VisualState.Setters>
78                                 <Setter Property="Source"
79                                     Value="MicUnmute.png" />
80                             </VisualState.Setters>
81                         </VisualState>
82                     </VisualStateGroup>
83                 </VisualStateManager.VisualStateGroups>
84             </ImageButton>
85         </Grid>
86     </Grid>
87 </ContentPage.Content>
88 </ContentPage>
```

B.2 Schaltpläne



B.3 Platinen-Design

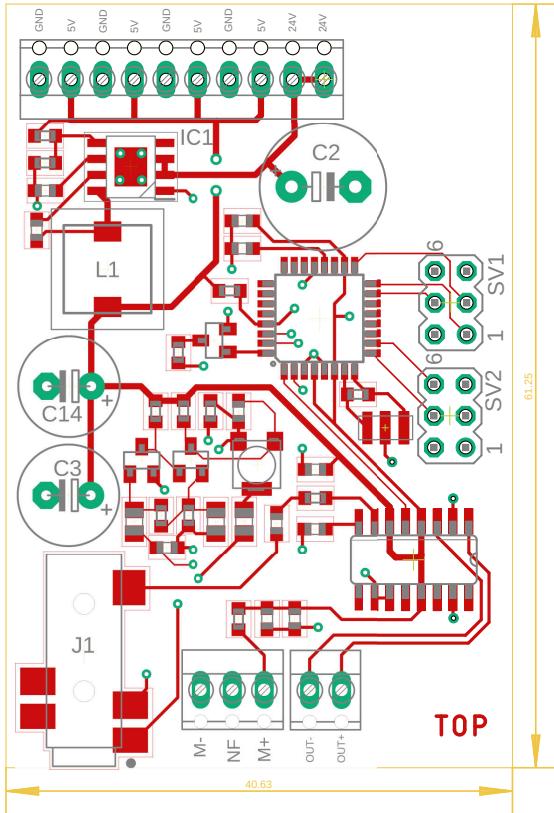


Abbildung B.1: Platine TOP-Layer

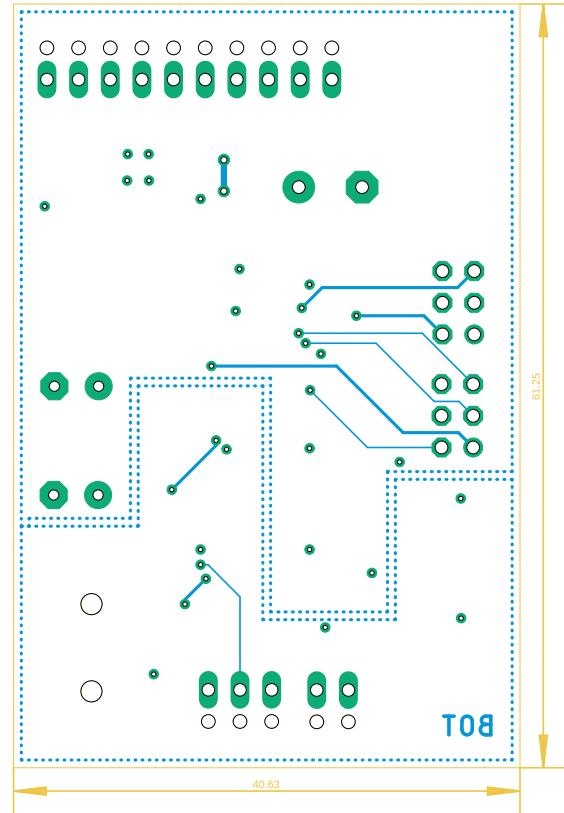


Abbildung B.2: Platine BOT-Layer

B.4 Stückliste

Qty	Value	Package	Parts	Description
1	470 nF	C0603	C13	Capacitor
1	820 μ F/35 V	E5-10,5	C2	polarized Capacitor
2	1 mF/6.3 V	E3,5-8	C3, C14	polarized Capacitor
3	2.2 μ F	C0805	C4, C5, C6	Capacitor
4	1 μ F	C0603	C8, C10, C12, C18	Capacitor
4	100 nF	C0603	C9, C15, C16, C17	Capacitor
1	LMR33630CDDAR	DDA0008J	IC1	Buck Converter
1	PAM8403	SOP-16	IC2	Dual channel audio amplifier
1	3.5 mm/4 conductor	SJ-43515RS-SMT	J1	Audio Jack
1	10 μ H/2.9 A	MSS7348-103MEB	L1	Inductor
1	6 Mhz	CSTCR6M00G53Z	Q1	Resonator
1	1 M Ω	R0603	R11	Resistor
2	10 k Ω	R0603	R1, R5	Resistor
1	100 k Ω	R0603	R13	Resistor
1	24.9 k Ω	R0603	R14	Resistor
1	47 k Ω	R0603	R2	Resistor

Qty	Value	Package	Parts	Description
1	5.6 kΩ	R0603	R3	Resistor
1	3.9 kΩ	R0603	R4	Resistor
1	2.2 kΩ	R0603	R6	Resistor
1	5 kΩ	POT_33/64 W	R7	Potentiometer
1	10Ω	R0603	R8	Resistor
2	10 kΩ	R0603	R9, R12	Resistor
2	2x3 pin	MA03-2	SV1, SV2	Pin Header
3	BC847	SOT23	T1, T2, T3	NPN Transistor
1	ATmega328P	TQFP32-08	U1	Microcontroller
1	3 pin	RS-CONN-2.54-3P	X1	Screw Terminal
1	2 pin	RS-CONN-2.54-2P	X2	Screw Terminal
1	10 pin	RS-CONN-2.54-10P	X3	Screw Terminal

Tabelle B.1: Platine Stückliste

Literatur und Quellen

- [1] (Okt. 2011). „Adobe Announces Agreement to Acquire Nitobi, Creator of PhoneGap“, Adobe Systems, Adresse: <https://news.adobe.com/press-release/adobe-creative-cloud-dps/adobe-announces-agreement-acquire-nitobi-creator-phonegap>.
- [2] (März 2020). „Compare Visual Studio 2019 Editions“, Microsoft, Adresse: <https://visualstudio.microsoft.com/vs/compare/>.
- [3] (Dez. 2019). „Install Visual Studio“, Microsoft, Adresse: <https://docs.microsoft.com/en-us/visualstudio/install/install-visual-studio?view=vs-2019>.
- [4] J. Piironen. (Feb. 2008). „Overview of the Common Language Infrastructure“, Wikimedia Commons, Adresse: https://commons.wikimedia.org/wiki/File:Overview_of_the_Common_Language_Infrastructure.svg.
- [5] H. Schulzrinne, A. Rao und R. Lanphier. (Apr. 1998). „RFC 2326 - Real Time Streaming Protocol (RTSP)“, Internet Engineering Task Force, Adresse: <https://tools.ietf.org/html/rfc2326>.
- [6] H. Schulzrinne, A. Rao, R. Lanphier, M. Westerlund und M. Stiemerling. (Dez. 2016). „RFC 7826 - Real-Time Streaming Protocol Version 2.0“, Internet Engineering Task Force, Adresse: <https://tools.ietf.org/html/rfc7826>.
- [7] J.-B. Kempf. (Feb. 2016). „15 years of VLC and VideoLAN“, Adresse: <http://www.jbkempf.com/blog/post/2016/15-years-of-VLC>.
- [8] J. Smith. (Dez. 2016). „Patterns - WPF Apps With The Model-View-ViewModel Design Pattern“, Microsoft, Adresse: <https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern>.
- [9] S. Chitrapu und E. Price. (Jan. 2014). „MVVM Model View ViewModel Part - 1“, Adresse: <https://social.technet.microsoft.com/wiki/contents/articles/13347.mvvm-model-view-viewmodel-part-1.aspx>.
- [10] M. Finkel. (Jan. 2020). „LibVLCSharp README.md“, VideoLAN, Adresse: <https://code.videolan.org/videolan/LibVLCSharp/-/blob/3.x/README.md>.

- [11] (März 2020). „highly optimized and efficient media streaming, encoding, decoding and processing libraries for developers“, VAStreaming, Adresse: <https://www.vastreaming.net/>.
 - [12] (März 2020). „GStreamer: a flexible, fast and multiplatform multimedia framework“, GStreamer, Adresse: <https://gstreamer.freedesktop.org/documentation>.
 - [13] (März 2020). „Google Trends - LIVE555“, Google, Adresse: <https://trends.google.com/trends/explore?date=all&q=%2Fm%2F0641r5r>.
 - [14] (März 2017). „Part 1 – Understanding the Xamarin Mobile Platform“, Microsoft, Adresse: <https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/building-cross-platform-applications/understanding-the-xamarin-mobile-platform>.
 - [15] R. Dold. (Juni 2018). „Schlüsseltechnologie Power over Ethernet (PoE)“, Elektropraktiker, Adresse: <https://www.elektropraktiker.de/nachricht/schlueseltechnologie-power-over-ethernet-poe/>.
 - [16] (Aug. 2012). „Datasheet Multilayer Ferrite Beads“, Vishay, Adresse: <https://www.vishay.com/docs/34024/ilbb0603.pdf>.
 - [17] (Sep. 2016). „Datasheet Precision Thin Film MELF Resistors“, Vishay, Adresse: <https://www.vishay.com/docs/28714/melfpre.pdf>.
 - [18] (Feb. 2018). „Datasheet Small Signal Zener Diodes“, Vishay, Adresse: <https://www.vishay.com/docs/83340/bzt52g.pdf>.
 - [19] (Juli 2016). „Datasheet Aluminum Capacitors“, Vishay, Adresse: <https://www.vishay.com/docs/25003/ecv.pdf>.
 - [20] (Aug. 2017). „Datasheet N-Channel 60 V (D-S) MOSFET“, Vishay, Adresse: <https://www.vishay.com/docs/71333/2n7002k.pdf>.
 - [21] (Nov. 2012). „Datasheet Optocoupler, Phototransistor Output, Dual Channel, SOIC-8 Package“, Vishay, Adresse: <https://www.vishay.com/docs/81956/vod205t.pdf>.
 - [22] (Dez. 2012). „Datasheet Low Input Current, Phototransistor Output, SOP-4, Mini-Flat Package“, Vishay, Adresse: <https://www.vishay.com/docs/83446/vom617a.pdf>.
 - [23] Benutzer:Cepheiden. (Feb. 2009). „Tombstone Effect DE“, Wikimedia Commons, Adresse: https://commons.wikimedia.org/wiki/File:Tombstone_Effect_DE.svg.
-

- [24] (Aug. 2010). „PopcornBGA.jpg“, Wikimedia Commons, Adresse: <https://commons.wikimedia.org/wiki/File:PopcornBGA.jpg>.
- [25] (Juni 2018). „UART (universal asynchronous receiver/transmitter)“, ITWissen, Adresse: <https://www.itwissen.info/UART-universal-asynchronous-receiver-transmitter.html>.
- [26] (März 2020). „UART Mode“, Tibbo Technology, Adresse: https://docs.tibbo.com/taiko/ser_uart_mode.
- [27] Benutzer:Andreas. (Feb. 2019). „Serial Peripheral Interface“, Mikrocontroller.net, Adresse: https://www.mikrocontroller.net/articles/Serial_Peripheral_Interface.
- [28] (März 2020). „HiFi-Lexikon: Class-D-Verstärker, Schaltverstärker“, fairaudio.de, Adresse: <https://www.fairaudio.de/lexikon/class-d/>.
- [29] Benutzer:Rohitbd und Benutzer:Pemu. (März 2009). „Pwm amp de“, Wikimedia Commons, Adresse: https://commons.wikimedia.org/wiki/File:Pwm_amp_de.svg.
- [30] (Sep. 2018). „Datenblatt F/UTP Patchkabel CAT.5e“, Lapp Gruppe, Adresse: https://www.lappkabel.de/fileadmin/documents/technische_doku/datenblaetter/ETHERLINE/Etherline_Leitungen/DBCE6547DE.pdf.
- [31] (März 2020). „wxMaxima“, wxMaxima Developers, Adresse: <http://wxmaxima-developers.github.io/wxmaxima/>.
- [32] (März 2019). „LMR33630 Datasheet“, Texas Instruments, Adresse: <https://www.ti.com/lit/ds/symlink/lmr33630.pdf>.
- [33] Benutzer:Ravi. (Feb. 2018). „What is a Power Amplifier? Types, Classes, Applications“, Electronics Hub.
- [34] (Apr. 2019). „AVR“, Mikrocontroller.net, Adresse: <https://www.mikrocontroller.net/articles/AVR>.
- [35] Benutzer:SM. (Jan. 2018). „Arduino as ISP and Arduino Bootloaders“, Arduino, Adresse: <https://www.arduino.cc/en/Tutorial/ArduinoISP>.
- [36] Benutzer:Zom. (März 2020), Arch Linux Wiki, Adresse: <https://wiki.archlinux.org/index.php/Systemd>.
- [37] (März 2020). „EAGLE Premium im Vergleich zu EAGLE Standard und zu EAGLE Free“, Autodesk, Adresse: <https://www.autodesk.de/compare/eagle-vs-eagle-premium>.

- [38] (Jan. 2015). „ATmega328P Datasheet“, Atmel, Adresse: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.
- [39] (Nov. 2012). „PAM8403 Datasheet“, Diodes, Adresse: <https://www.diodes.com/assets/Datasheets/PAM8403.pdf>.

Abkürzungen

API	Application Programming Interface
APNs	Apple Push Notification Service
CAD	Computer-aided Design
CIL	Common Interface Language
CLR	Common Language Runtime
COM	Communication (Port)
CPU	Central processing unit
DIP	Dual In-Line Package
.NET	.NET-Framework
DRC	Design Rule Check
EAGLE	Einfach Anzuwendender Grafischer Layout-Editor
ERC	Electrical Rule Check
FIFO	First In First Out
GCM	Google Cloud Messaging
GPL	General Public License
HTTP	Hypertext Transfer Protocol
I ² C	Inter-IC
IC	Integrated Circuit
ICSP	In-Circuit Serial Programmer
IDE	Integrated Development Environment
IDE	Integrated Drive Electronics
IETF	Internet Engineering Task Force
IO	Input/Output
ISP	In-System Programming
ISR	Interrupt Service Routine
LPT	Line Printer Terminal
MELF	Metal Electrode Faces
MISO	Master In Slave Out
MOSI	Master Out Slave In
MVVM	Model-View-ViewModel
PCB	Printed Circuit Board
PD	Powered Device

PoE	Power over Ethernet
PoE+	Power over Ethernet Plus
4PPoE	Four Pair Power over Ethernet
PSE	Power Sourcing Equipment
PWM	Pulse Width Modulation
RISC	Reduced Instruction Set Computer
RPi	Raspberry Pi
RTC	Real-Time Clock
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SCK	Serial Clock
SDK	Software Development Kit
SMD	Surface-mounted Device
SMT	Surface Mounting Technology
SOD	Small Outline Diode
SOIC	Small Outline Integrated Circuit
SOP	Small Outline Package
SOT	Small Outline Transistor
SPI	Serial Peripheral Interface
SS	Slave Select
SSH	Secure Shell
THT	Through-Hole Technology
UART	Universal Asynchronous Receiver Transmitter
UI	User interface
URL	Uniform Resource Locator
USB	Universal Serial Bus
UWP	Universal Windows Platform
V-Chip	Vertical Chip
VPN	Virtual Private Network
WPF	Windows Presentation Foundation
WYSIWYM	What You See Is What You Mean
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

Abbildungsverzeichnis

1	App-Ansichten	ix
2	Bestückte Platine Top	xi
3	Bestückte Platine Bottom	xi
2.1	Auswahl des Xamarin-Rahmenwerks bei der Installation	9
2.2	Android SDK Manager	11
2.3	Notwendige Teile des Android SDK	12
2.4	Build-Vorgang einer .NET-Applikation [4]	13
2.5	Build-Vorgang einer Xamarin.Forms-Solution	14
3.1	Datenfluss MVVM [9]	18
4.1	Lokale Benachrichtigung	25
5.1	Aufbau der Solution	28
5.2	VisualStateManager	33
5.3	App-Oberfläche	33
5.4	Abfrage Anrufannahme	35
5.5	Anfrage für Berechtigung	43
7.1	Unübersichtlicher Aufbau	50
8.1	Chip-Bauform [vgl. 16]	55
8.2	MELF-Bauform [vgl. 17]	55
8.3	SOD-123-Bauform [vgl. 18]	56
8.4	V-Chip-Bauform [vgl. 19]	56
8.5	SOT-23-Bauform [vgl. 20]	57
8.6	SOIC-Bauform [vgl. 21]	57
8.7	SOP-Bauform [vgl. 22]	58
8.8	Grabstein-Effekt [23]	58
8.9	Popcorn-Effekt [24]	59
8.10	Aufbau eines UART-Frames [26]	60
8.11	Schematische Darstellung Class-D [29]	62
9.1	Verdrahtungsschema	63
9.2	Ersatzschaltung PoE Type 1 und 2	64

9.3 Ersatzschaltung PoE Type 3 und 4	64
9.4 Eingabe wxMaxima Type 1 & 2	65
11.1 ICSP Schnittstelle [35]	78
11.2 ArduinoISP Sketch	78
11.3 Arduino IDE Einstellungen	79
11.4 Hochladen des Programmes	79
13.1 Exportierte Platine in Fusion 360	89
B.1 Platine TOP-Layer	125
B.2 Platine BOT-Layer	125

Alle Abbildungen ohne Referenzmarke „[#]“ sind Eigenfotos

Tabellenverzeichnis

2.1 Unterschiede Visual-Studio-Editionen [vgl. 2]	10
3.1 Anfrage-Arten des RTSP-Protokolls [5]	16
8.1 Vergleich PoE-Typen [vgl. 15]	52
9.1 Berechnung Ergebnisse	66
12.1 Verfügbare EAGLE-Versionen ab 2017 [vgl. 37]	87
B.1 Platine Stückliste	126