



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT



Lehrstuhl für Technische Elektronik

Prof. Dr.-Ing. Dr.-Ing. habil. Robert Weigel

Prof. Dr.-Ing. Georg Fischer

Bachelorarbeit

im Studiengang

„Elektrotechnik, Elektronik und Informationstechnik (EEI)“

von

Christof Pfannenmüller

zum Thema

Aufbau und Inbetriebnahme einer mobilen Basisstation für feldstärkebasierte Lokalisierung

Betreuer: Dipl.-Ing. Felix Pflaum

Beginn: 25.04.2016

Abgabe: 26.09.2016

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.

Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 26. September 2016

Christof Pfannenmüller

Kurzfassung

Ziel der vorliegenden Arbeit war der Aufbau einer Basisstation mit sechs unabhängigen Transceivern. Der Einsatz dieses Aufbaus diene der späteren Lokalisierung von Sensoren oder anderen Sendern im Sub-GHz-Frequenzbereich um 868 MHz dienen. Die relative Ortsbestimmung zur Basis sollte energieeffizient sein und gleichzeitig eine hohe Auflösung bieten.

Die genaue Position des Senders wurde dazu auf Basis der unterschiedlichen Feldstärken an den Transceivern eruiert. Ausgenutzt wurde dabei, das typische integrierte Transceiver die Empfangsfeldstärke selbst auswerten und bereitstellen, sodass diese ausgelesen werden kann. Das Erkennen von Übertragungen und das anschließende Auslesen der anfallenden Daten wurde dabei von einem Mikrocontroller übernommen. Zur weiteren Verarbeitung der Daten sollten diese anschließend einem Computer zur Verfügung gestellt werden. Dazu wurde sowohl eine USB- als auch eine Netzwerk-Schnittstelle vorgesehen. Die beim Aufbau der Platine verwendete Hardware basierte zum Großteil auf Bauteilen des Herstellers Infineon. Beim Layout der Platine wurden die sechs Transceiver sternförmig und regelmäßig um den Mikrocontroller und dessen Peripherie angeordnet, um ein gleichmäßiges Empfangsverhalten aus allen Raumrichtungen zu gewähren. Die Funksegmente der Platine wurden so gestaltet, dass diese bei Bedarf abgetrennt und mit einer Kabelverbindung weiter voneinander entfernt werden konnten. Antennen zum Senden wurden über Steckverbinder an die Basisstation angeschlossen werden.

Der anschließende Softwareentwurf für die Basisstation nutzte zu einem Großteil bereits bestehende Bibliotheken und hatte zur Aufgabe ankommende Übertragungen erkennen, die zur Ortung notwendigen gemessenen Werte abfragen und an den Hostcomputer weiterleiten.

Abstract

Current localization measurements by magnitude and phase of electromagnetic waves have been complex and consume plenty of energy. However almost every receiver has knowledge of the electrical field strength, correlating to distance from transmitter, this information is nearly unused. A multi transceiver base station should start communication with a mobile wireless sensor. The relative positioning to the base could be calculated by the received signal strength (RSSI) already provided from transceivers without additional components. Due to the permeability of walls, the possible range and the wavelength, associated to resolution of localization, Sub-GHz frequency range is used. Therefore six identical transceiver-ICs were arranged over all horizontal directions in space. By detachable design of the transceiver-modules a rearranging of the antennas was made feasible to provide different distances for best resolution of localization. The ICs were controlled by a XMC4500 microcontroller connected to the transceivers with SPI and IRQ line for finished transmission. Distribution of received data and signal strength measurements to a host computer is accomplished by the XMC4500 over Ethernet and USB.

Abkürzungsverzeichnis

PCB	Printed Circuit Board	3
EDA	Electronic Design Automation	3
CAD	Computer-aided design	3
SMD	Surface-mounted device	4
SPI	Serial Peripheral Interface	5
DRC	Design-Rule-Check	5
FIFO	First In – First Out	6
NCS	Non-Chip-Select	6
IC	Integrierter Schaltkreis	7
SMA	Sub-Miniature-A	7
LQFP	Low Profile Quad Flat Package	9
BGA	Ball Grid Array	10
JTAG	Joint Test Action Group	10
TVS	Transient Voltage Suppressor	11
LDO	Low Drop-Out	13
SOT	Small Outline Transistor	13
STEP	Standard for the Exchange of Product Model Data	15
NC	Numerical Control	15
IDE	integrated development environment	17
GUI	Graphical User Interface	17
SDK	Software development kit	17
CPU	Central Processing Unit	18
USIC	Universal Serial Interface Channel	19
ETH	Ethernet MAC (Ethernet Medium Access Control)	18
USB	Universal Serial Bus	18
GPIO	General Purpose Input/Output	18

ISR	Interrupt Service Routine	20
ERU	Event Request Unit	20
ERS	Event Request Select	20
ETL	Event Trigger Logic	20
OGU	Output Gating Unit	20
NVIC	Nested Vectored Interrupt Controller	20
GND	Masse (Ground)	22
CMSIS	Cortex Microcontroller Software Interface Standard	21
MISO	Master-In Slave-Out	22
MOSI	Master-Out Slave-In	22
ASCII	American Standard Code for Information Interchange	23
IRQ	Interrupt Request	24
PLL	Phasenregelschleife (phase-locked loop)	25
AGC	automatic gain control	26
RSSI	Received Signal Strength Indication	27

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Zieldefinition	2
1.3	Projektmanagement	2
2	Platinenaufbau	3
2.1	Vorüberlegungen	3
2.2	Layoutprogramm Altium Designer	3
2.3	Verwendete Hardware	6
2.3.1	TDA5340	6
2.3.2	XMC4500	9
2.3.3	USB	10
2.3.4	Ethernet	11
2.3.5	Spannungsversorgung	12
2.4	Generierte Dokumente	14
2.5	Bestückung	16
3	Software	17
3.1	DAVE Entwicklungsumgebung	17
3.2	Verwendete Peripherie des XMC4500	18
3.2.1	GPIO	18
3.2.2	USIC	19
3.2.3	ERU	20
3.2.4	USB	20
3.2.5	Ethernet	21
3.3	Verwendete Bibliotheken	21
3.3.1	XMC Library (XMC Lib)	21
3.3.2	SPI Library	21
3.3.3	TDA5340 Library	23
3.3.4	Virtueller COM Port	23
3.4	Programmablauf	24
3.4.1	Konfiguration der Funkmodule	25
3.4.2	Interruptbasierte Datenerfassung	25
3.4.3	Weiterleitung der erfassten Daten	26

4	Feldtest	31
4.1	Aufbau	31
4.2	Durchführung	31
4.3	Ergebnisse und Auswertung	32
5	Zusammenfassung und Ausblick	33
	Abbildungsverzeichnis	35
	Tabellenverzeichnis	37
	Quellcodeverzeichnis	39
6	Anhang	41
6.1	Schaltpläne	41
6.2	Seriennummern	48
6.3	3D-Daten	49
6.3.1	Platine	49
6.3.2	Gehäuse	49
6.4	Layout Aufsteckboard TDA5340	50
6.5	Quellcode	51
6.5.1	Main.c	51
6.5.2	ISRs.c	59
6.5.3	Init.c	60

Einleitung

1.1 Motivation

Die Lokalisierung von Objekten bildet in der Sicherheits- und Automatisierungstechnik eine immer größere Basisdisziplin. Sie ist daher seit Anbeginn des Informationszeitalters eine zentrale Aufgabe der Elektronik. Im Zeitalter von Industrie 4.0 und allgegenwärtigen autonomen Systemen wird die Ortung von Objekten unersetzlich und gewinnt weiter an Bedeutung.

Bisherige Ansätze zur elektronischen Ortsbestimmung sind meist aufwändig mit einem hohen Energieverbrauch und benötigen viele Teilkomponenten. Dies macht die elektronische Ortsbestimmung teuer, was sich vor allem im Betrieb von Systemen wie GPS oder Galileo zeigt. Daneben haben diese Systeme vor allem den Nachteil, dass die verwendeten Frequenzen Wände kaum durchdringen und somit innerhalb von Gebäuden nicht verwendet werden können. Genau dieses Anwendungsszenario stellt jedoch in der Industrie 4.0 eine typische Fabrik dar. Dies führt dazu, dass für das am häufigsten verwendete Beispiel, einen autonom agierenden Roboter in einer Lagerhalle, andere Techniken zur Ortsbestimmung eingesetzt werden müssen.

Ortungsansätze für ein solches autonomes bewegliches System existieren bereits und werden wie bei der WLAN-basierten Ortung auch eingesetzt. Diese Systeme haben jedoch den Nachteil, dass sie viele Sender mit bekanntem Ort benötigen. Dies erfordert eine aufwändige Kalibrierung des Systems und führt vor allem zu einem großen Energieverbrauch des Gesamtsystems, da alle stationären Einheiten dauerhaft senden müssen, um eine Ortung zu ermöglichen. Ein weiterer Nachteil der Systeme ist, dass nur der zu ortende Client die Informationen über seinen Standort hat. Für ein außenstehendes System, wie den steuernden Hauptrechner einer Industrieanlage, ist nicht festzustellen, wo das Objekt sich befindet. Durch diese Probleme ließe sich das Anwendungsszenario eines einzelnen autarken Funksensors, dessen Messwert vom Hauptrechner abgefragt wird, nur schwer realisieren. Der Sensor müsste über eine große Energiereserve verfügen und ständig aktiv sein, um zu erkennen wenn er dazu aufgefordert wird seine Position zu ermitteln und mitzuteilen. Durch das Verlagern der Ortungsaufgabe auf das stationäre System müsste der Sensor nur beim Vorliegen eines neuen Messwertes eine Funkverbindung aufbauen.

Die Basisstation könnte aus dem empfangenen Signal sowohl den Messwert extrahieren als auch eine Lokalisierung durchführen. Sollte der stationäre Teil aus nur einer Einheit bestehen, wäre auch eine Änderung am Gesamtsystems, wie eine Ortsänderung der stationären Einheit, einfach möglich, da nur die relative Position zu dieser ermittelt wird, weshalb eine Neukalibrierung des Systems entfällt. Der Sensor selbst benötigt im Normalfall keine Information über seinen Aufenthaltsort. Dies erlaubt eine mobile Ortung da die Basis unkompliziert bewegt werden kann.

Es bietet sich hierfür eine Nutzung des Sub-GHz-Frequenzbandes an, welches deutlich weniger belegt ist, in dem meist nur kurze Funkdauern verwendet werden und in welchem eine Übertragung vor allem energieeffizienter möglich ist **SabolkicGHzoderSub**. Ein weiterer Vorteil des verwendeten Frequenzbandes sind die bessere Penetrationseigenschaften durch Wände und Personen im Vergleich zu Ansätzen, die etwa auf WLAN basieren.

1.2 Zieldefinition

Zum Umsetzen einer solchen oben beschriebenen Ortung sollte eine Basisstation aufgebaut werden, welche als stationäre Einheit einen Sender orten könnte. Die Aufgabenstellung dieser Bachelorarbeit bestand im Einarbeiten in den vorgegeben Transceiver TDA5340 und den Mikrocontroller XMC4500. Bei der Gestaltung des Layouts für die Platine mit Altium Designer wurden diese über einen SPI-Bus verbunden sein. Die Platine sollte zusätzlich ein Konzept zur Bereitstellung der Versorgungsspannung enthalten. Um die Weiterverarbeitung der bezogenen Messwerte sicherzustellen, war eine LAN-Schnittstelle sowie eine Möglichkeit für einen USB-Anschluss zu einem PC zu integrieren.

Das ursprünglich geplante Design von PCB-Antennen an den Transceivern auf der Platine wurde wegen des Umfangs der dazu notwendigen Simulationen zu Beginn aus der Aufgabenstellung entfernt. Stattdessen wurden zugekaufte Antennen mit der Basisstation verwendet, welche über Steckverbinder angeschlossen wurden.

Eine funktionierende Firmware für das Initialisieren und Betreiben der Basisstation war ebenfalls mit der dazu notwendigen Entwicklungsumgebung zu erstellen. Dabei musste der Mikrocontroller auf, vom Transceiver ausgelöst, Interrupts für ankommende Übertragungen reagieren und Messwerte erfassen beides wurde anschließend entsprechend weitergeleitet. Durch die Verwendung von Makros beim Softwareentwurf sollte eine spätere Anpassung, wie das Tauschen von Pins leichter möglich sein.

1.3 Projektmanagement

Die vorliegende Arbeit wurde innerhalb von fünf Monaten am Lehrstuhl für Technische Elektronik der Universität Erlangen-Nürnberg angefertigt. Dabei lag der Fokus in den ersten beiden Monaten auf dem Layout der Platine und dem anschließenden Bestücken. In der folgenden Zeit wurde vermehrt auf die Software für den Betrieb der Basisstation eingegangen. Außerdem wurde in den letzten zwei Monaten die Dokumentation mit Latex erstellt. Als Versionskontrolle für das gesamte Projekt wurde Github eingesetzt.

Platinenaufbau

2.1 Vorüberlegungen

Die Zielsetzung im Aufbau der Platine war eine kompakte Basisstation mit sechs Transceivern und einer zentralen Steuereinheit. Um sicherzustellen, dass alle Antennen gleichmäßig in die sechs vorgegebenen Raumrichtungen abstrahlen, sollte bereits die Platine symmetrisch aufgebaut werden. Dazu wurde zuerst das Layout der sechs identischen Transceiver-Einheiten mit dem TDA5340 Baustein und den Antennenanschlüssen erstellt und anschließend gleichmäßig um die weiteren für die Schaltung notwendigen funktionellen Segmente angeordnet.

2.2 Layoutprogramm Altium Designer

Bei dem Entwicklungswerkzeug „Altium Designer“ des Entwicklers Altium Limited handelt es sich um ein System zum Entwurf von gedruckten Schaltungen oder Printed Circuit Boards (PCBs). Ein solches Programm wird auch als Electronic Design Automation (EDA) oder ECAD für electronic Computer-aided design (CAD) bezeichnet, da es den Entwickler bei der Umsetzung der Anforderungen in einen Schaltplan und später eine Platine unterstützen soll. Wie viele andere EDA-Programme ist auch Altium Designer so aufgebaut, dass sich der Entwickler zuerst mit dem allgemeinen symbolisierten Schaltplan befassen kann und erst zu einem späteren Zeitpunkt die tatsächliche Anordnung der Bauteile auf dem PCB-Substrat festgelegt wird. Somit können zuerst im Schematic Editor die Funktionen der Schaltung umgesetzt werden. Dazu werden die verwendeten Bauteile aus zuvor angelegten Bibliotheken verwendet oder es werden bestehende Libraries genutzt, die etwa vom Hersteller der Bauteile zur Verfügung gestellt werden. Altium selbst bietet hierfür auch diverse Möglichkeiten an und stellt Bauteile nach Hersteller und Art geordnet bereit. In den Bibliotheken sind alle im weiteren Verlauf benötigten Informationen über die einzelnen Bauteile enthalten. So liegen dort etwa entsprechenden Abbildungen für das Bauteil im Schaltplan vor. In den so genannten „Footprints“ zu jedem Bauteil, welche ebenfalls in den Bibliotheken enthalten sind, wurde zuvor die, für das physikalische

Gehäuse, notwendigen Abmessungen, Löt pads und Ausmaße für Lötstopplack um das Bauteil festgelegt. Da es Bauteile, wie den verwendeten Mikrocontroller, in verschiedenen Gehäusen geben kann, besteht somit auch die Möglichkeit hier zwischen verschiedenen Footprints zu wählen. Da viele Gehäuse herstellerübergreifend genormt sind, konnten teilweise bestehende Footprints genutzt oder diese mehrfach verwendet werden.

Wie bereits erwähnt wird im EDA-Programm zuerst der symbolische Schaltplan erstellt. Dieser wird anschließend in ein Layout für eine Platine umgewandelt. Die zu den Schaltplansymbolen korrespondierenden Footprints werden dazu auf dem Layout der Platine angeordnet und durch das „Routing“ werden die Leiterbahnen definiert. Altium Designer ist dabei in drei Teilbereiche unterteilt: im „Board Planning Mode“ liegt der Fokus auf dem Anordnen der einzelnen Bauteile und Komponenten auf der Leiterplatte, außerdem wird in diesem Bereich die Form und das Ausmaß der Leiterplatte festgelegt. Im 2D-Modus des PCB-Editor lassen sich anschließend die aus der Definition im Schaltplan ergebenden elektrischen Verbindungen örtlich auf den verschiedenen Kupferebenen (Layern) anordnen. Die Hauptarbeit findet also in diesem Teil des PCB-Editors statt. Der 3D-Modus dient anschließend zur Evaluation des Designs und zur Anpassung an Gehäuse oder andere Komponenten. In den verschiedenen Ebenen oder „Layern“ sind die Kupferebenen und andere Schichten der späteren Platine wie der Bestückungsdruck oder der Lötstopplack gesammelt. Jede Ebene entspricht daher einer zu fertigenden Schicht und existiert für die Vorder- und Rückseite der Platine. Beim Bewegen eines Bauteils wird nicht nur die markierte Abmessung, sondern etwa auch die Anschluss-Pads und der Lötstopplack auf denen entsprechenden Ebenen bewegt. Durch Vias sind elektrische Verbindungen zwischen den Kupferschichten möglich.

Für die Basisstation wurden zwei Kupferebenen und ausschließlich sogenannte Surface-mounted device (SMD)-Bauelemente verwendet. Diese liegen nur auf der Oberfläche der Platine auf und sind durch ihre Lötverbindungen befestigt. Die Vorteile dieser Bauteile sind der geringe Preis und die kleinen Abmessungen.

Wegen der Größe des Projekts wurde zur besseren Übersicht ein so genanntes „Multi-Sheet-Design“ erstellt. Dadurch war es möglich, die verschiedenen funktionellen Blöcke der Basisstation auf getrennte Blätter des Schaltplans zu verteilen. Der Mikrocontroller, die Spannungsversorgung und der Transceiver, sowie die für eine Netzkommunikation notwendigen Bauteile wurden dabei auf unabhängigen Seiten angeordnet und dort die elektrischen Verbindungen erstellt.

Da der Transceiver und die entsprechende Peripherie sechsmal in identischer Anordnung und Beschaltung verwendet wurden und auch auf dem PCB-Substrat mehrfach mit Leiterbahnen verbunden und angeordnet werden mussten, wurde hierfür ein so genanntes „Multi-Channel“-Design gewählt. In Altium Designer können mit diesem Feature identische Schaltungsteile einmal angeordnet, mit Leiterbahnen verbunden und dieses Design auf alle anderen entsprechenden Schaltungsteile angewendet werden. Somit muss das aufwändige Anordnen der Bauteile und die Führung der Leiterbahnen nur bei einem der Kanäle durchgeführt werden. Dazu wurde mit Hilfe eines übergeordneten Sheet-Symbols der Schaltplan des Transceivers in den Schaltplan des Mikrocontrollers eingefügt und diesem somit untergeordnet. Über Ports, welche zum Schaltplansymbol hinzugefügt werden, lassen sich elektrische Verbindungen zwischen Netzen innerhalb der Schaltpläne für Transceiver und Mikrocontroller erstellen. Dabei wird im Transceiver-Schaltplan

ein Port hinzugefügt, der mit dem gewünschten elektrischen Netz verbunden werden kann. Auf dem Schaltplansymbol im Mikrocontroller-Schaltplan wird ein entsprechender gleichnamiger Port erstellt, der mit Netzen am Mikrocontroller verbunden werden kann. Da die drei Verbindungen der Serial Peripheral Interface (SPI)-Kommunikation jeweils aus einem Netz bestehen und etwa alle MISO-Leitungen an demselben Pin des XMC4500 und denselben Anschluss bei allen TDAs angebunden sind, konnte hierfür ein einfacher Port verwendet werden. Alle anderen Anschlüsse, wie etwa die Auswahlleitung für die SPI-Verbindung, welche für jede der sechs verschiedenen Transceiver-Einheiten mit einem anderen Anschluss des Mikrocontrollers verbunden sein mussten, wurden deswegen mit dem Repeat-Kommando erstellt. So wird im untergeordneten Schaltplan, in diesem Fall dem des TDA, der Port beliebig benannt, etwa als „NCS“. Der auf dem Schaltplansymbol erstellte korrespondierende Port wird dagegen in „Repeat(NCS)“ umbenannt. Eine Verbindung mit dem Port auf dem Schaltplansymbol wird dadurch zu einem Bus. Dieser kann aufgetrennt werden und die Verbindungen von jedem Kanal als einzelnen Signal an den Mikrocontroller angeschlossen werden. Globale Netze wie die Versorgungsspannung von 3,3 Volt oder die Masse müssen dabei nicht als Port hinzugefügt werden. Altium Designer stellt deren Verbindungen automatisch her. Wird nun noch der Name des Schaltplansymbols nach dem folgenden Muster angepasst, wird Altium Designer beim Kompilieren des Projektes ein Multi-Channel-Design erstellen, die Kanäle wie angegeben durchnummerieren und den Kanal im PCB-Editor entsprechend mehrfach erstellen. Dazu sollte der Name die Struktur „Repeat(<Name>, <Startnummer> , <Endnummer>)“ aufweisen. Anschließend kann die Anordnung und das Routing des ersten Kanals erfolgen. Da für jeden Kanal ein eigener so genannter „Room“ erstellt wird, lassen sich, nach dem Erstellen der Leiterbahnen des ersten Kanals, die Anordnungen der Bauteile und Leiterbahnen mit dem „Copy Room Format“-Befehl auf alle anderen Kanäle erweitern. Ein Room bezeichnet dabei eine Gruppe an Elementen im PCB-Editor und einen Bereich der Platine, in dem diese angeordnet sind und erlaubt so das Verschieben ganzer Schaltungsteile auf der Platine. Der Schaltplan des Hauptkanals, welcher einen einzelnen der sechs Kanal darstellt, ist in Abbildung 2.1 zu erkennen.

Altium Designer nummeriert automatisch die verwendeten Bauelemente durch, um die genaue Identifikation eines Bauteiles zu erlauben und erstellt automatisch einen Aufdruck neben dem Lötpad mit dem Bauteilnamen in der entsprechenden Ebene. Ein solcher Bestückungsdruck zur leichteren Anordnung der Bauteile auf der fertig entwickelten Platine wurde nur auf der Hauptplatine erstellt. Auf den Teilplatinen für die Transceiver wurde dieser aus Platzgründen weggelassen.

Um sicherzustellen, dass die sich aus der Bauteilanordnung ergebenden Pads und die Leiterbahnen auch fertigbar sind, stellt Altium Designer zwei Design-Rule-Checks (DRCs) bereit. Im ersten Live-DRC werden „Violations“, also Bauteile mit Verstößen gegen die Designregeln, durch ein farbiges Overlay markiert. Im folgenden kompletten Test zeigt Altium alle weiteren Verstöße an. Als Verstoß gelten etwa Leiterbahnen unterschiedlicher Netze, die sich berühren und zu einem Kurzschluss führen würden oder auch Verletzungen der festgelegten Abstandsregeln (Clearance). Alle Regeln für Violations können im „PCB Rules Editor“ eingestellt werden und so an die Möglichkeiten des PCB-Herstellers angepasst werden. Da die Leiterplatte für die Basisstation von Multi Circuit Boards Ltd. hergestellt wurde, konnten die auf der Webseite dieses Herstellers angegebenen

Designregeln für den DRC übernommen werden. Da Altium Designer im DRC sich kurzgeschlossene und falsch verbundene Pads angezeigt werden, muss kein extra „Layout vs. Schematic“-Test durchgeführt werden, der den vorher erstellten Schaltplan mit dem Layout abgleicht. Solche Unterschiede zwischen Schaltplan und dem PCB-Layout würden durch den DRC bereits angezeigt.

2.3 Verwendete Hardware

2.3.1 TDA5340

Der verwendete Transceiver TDA5340 wird von Infineon Technologies AG entwickelt und vertrieben. Er ist Teil der SmartLEWIS Produktfamilie die energiesparende Lösungen für Funkanwendungen im Frequenzspektrum unterhalb von einem Gigahertz bietet. Der Transceiver kommuniziert mit seinem Host über das SPI-Protokoll, der Mikrocontroller ist in diesem Fall sternförmig mit den einzelnen TDA-Bausteinen verbunden, die als Slaves fungieren. Die Daten werden mit drei gemeinsamen Leitungen übertragen, eine vierte Leitung dient dem XMC zur Auswahl des gewünschten Slaves für die Kommunikation. Diese Non-Chip-Select (NCS)-Leitung arbeitet active-low, sodass der jeweilige TDA5340 eine Interaktion akzeptiert, sobald diese vom XMC-Baustein auf Massepotential gezogen wird. Von den drei eigentlichen Datenleitungen fungiert eine als reiner Ausgang des Masters bzw. Dateneingang des TDA (MOSI), eine zweite als Eingang des Masters (MISO) und die dritte als ein vom Mikrocontroller getriebenes Clock-Signal. Bei dem auf MISO und MOSI anliegenden Signal handelt es sich um ein unipolar kodierte non-return-to-zero Signal, welches einer logischen 0 bei Massepotential entspricht. Der TDA unterstützt acht verschiedene Instruktionen, die es erlauben entweder einzelne Register des Bausteins zu lesen bzw. zu schreiben, auf mehrere hintereinander folgende Register oder auf die beiden Puffer des Bausteins zuzugreifen. In den beiden Puffern, die als First In – First Out (FIFO)-Strukturen aufgebaut sind, werden die vom TDA erkannten und demodulierten bzw. die auf Übertragung wartenden Signalkomplexe zwischengespeichert. Diese Zwischenspeicherung soll den Mikrocontroller entlasten, so können entsprechende Datenpakete dem TDA5340 mitgeteilt werden und dieser übernimmt selbstständig eine korrekte Modulation und Übertragung mit den eingestellten Parametern.

Der TDA5340 kann sowohl mit einer Versorgungsspannung von 5 V als auch bei 3,3 V arbeiten. Da aber der XMC nur bei letzterer betrieben werden kann, wurde der TDA-Baustein und die externe Beschaltung ebenfalls auf 3,3 V ausgelegt.

Um zu einem späterem Zeitpunkt eine größere Entfernung zwischen den einzelnen Antennen, und somit auch den jeweiligen Transceivern zu erlauben, wurde eine Sollbruchstelle vorgesehen. Dadurch könnte die gesamte Baugruppe von der Mutterplatine entfernt werden, was unter Umständen notwendig gewesen wäre, um die Auswirkungen verschiedener Antennenabstände an der Basis zu evaluieren und somit eine bessere Auflösung in der Ortung zu erlauben. Dazu wurden Anschlussleisten im Rastermaß 2,54 mm an beiden Seiten der Sollbruchstelle vorgesehen. Die Verbindung der Transceiver-Einheiten mit der Hauptplatine wurde über diese Sollbruchstelle hinweg mit Leiterbahnen gewährleistet. Nach dem Abtrennen der TDA-Teilplatine, an der durch Bohrungen vorgesehenen Bruchstelle, wäre die elektrische Verbindung durch Kabel sichergestellt worden. Da es sich bei

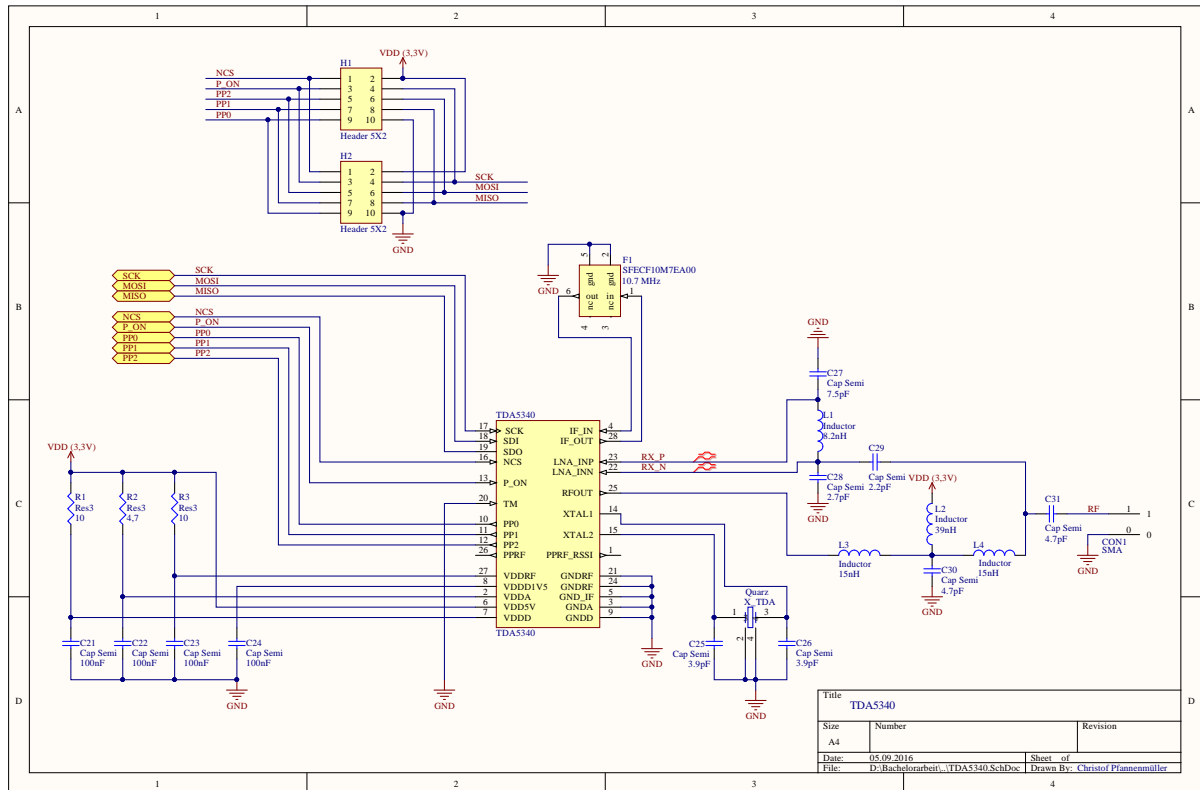


Abb. 2.1: Schaltplan der Transceiverbaugruppe

den zu übertragenden ausschließlich um digitale Signale geringerer Frequenzen im Bereich unter 1 MHz handelt, hätte dies unproblematisch mit ungeschirmten Flachbandkabeln realisiert werden können. Neben der Versorgungsspannung, Masse und den vier für die SPI-Kommunikation notwendigen Signalen wurden noch die drei multifunktionalen Digitalausgänge und der Power-On Reset-Pin (P_ON) dem XMC4500 über die Buchsenleisten zur Verfügung gestellt.

Die drei multifunktionalen Digitalausgänge PP0, PP1 und PP2 des Transceivers können durch entsprechende Kalibrierung von Registers im Integrierten Schaltkreis (IC) mit verschiedenen Signalen belegt werden. So ist es etwa möglich mit diesen einen externen Antennenumschalter zu steuern oder andere Signale wie ein Clocksignal auszugeben. PP2 wird vom TDA standardmäßig als Interrupt Signal verwendet. Die nicht verwendeten PP1 und PP0 wurden bei der späteren Konfiguration in den hochohmigen Zustand geschaltet. Durch einen am P_ON-Pin anliegenden High-Pegel wechseln der Transceiver in den eingeschalteten Zustand. Anderenfalls ist dieser ausgeschaltet und verbraucht typischerweise weniger als 1 μ A.

Die Antenne wurde am oberen Ende jeder TDA5340-Teilplatine vorgesehen. Als Anschluss für die Antenne wurde hier eine Koaxialbuchse in Sub-Miniature-A (SMA)-Ausführung verwendet, welche auf 50 Ω angepasst ist. Durch den Koaxialsteckverbinder konnte sichergestellt werden, dass alle notwendigen Frequenzen auch korrekt und unge-dämpft passieren können. Das Anpassnetzwerk zwischen dem integrierten Transceiver und der verwendeten SMA-Buchse diente der Leistungsanpassung zwischen den Pins des

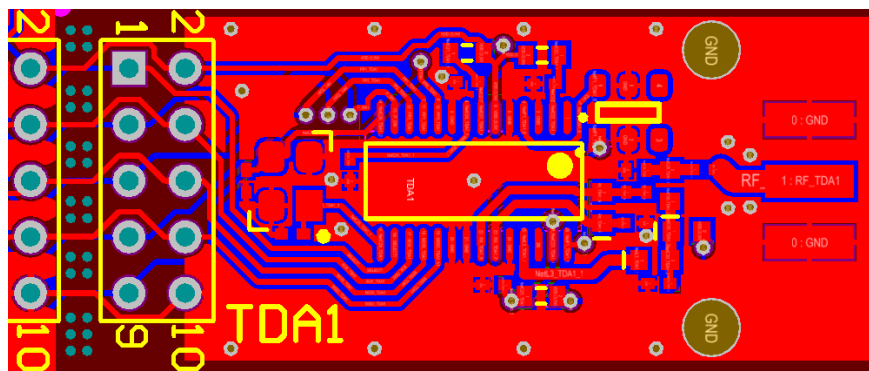


Abb. 2.2: Layout des Transceivers mit der Sollbruchstelle, der Steckerleiste und dem Anpassnetzwerk. Die Lötflächen für die SMA-Buchse sind auf der rechten Seite erkennbar.

TDA5340 und der 50 Ω -Koaxialbuchse. Der Aufbau des Anpassnetzwerkes basiert auf einem von Stefan Erhard erstellten Schaltplan für eine Aufsteckplatine für das Evaluationsboard „XMC 2Go“ von Infineon. Durch die Verwendung von hoch abgestimmten Spulen und Kondensatoren mit Toleranzen von nur $\pm 0,05$ pF bei einem Nennwert von 2,5 pF wurde die korrekte Anpassung sichergestellt.

Zur Verbesserung der Hochfrequenzeigenschaften wurden die Freiräume zwischen den Leiterbahnen mit einer Kupferfläche gefüllt, die mit dem Masseanschluss verbunden war. Durch die Verwendung von Vias, vor allem im Bereich des Anpassnetzwerkes, sollte eine niederohmige Verbindung zwischen den beiden Masseflächen auf der Ober- bzw. Unterseite der Platine erreicht werden. Daneben dienten diese, auf Nullpotential liegenden Vias, jedoch vor allem der Abschirmung der Pfade für die HF-Signale gegen mögliche Einkoppelungen aus der Umgebung, welche ankommende Funksignale stören könnten.

Obwohl der TDA5340 einen eingebauten Zwischenfrequenz-Filter hat, der über eine umschaltbare Bandbreite verfügt, wurde ein externer Keramikfilter verwendet. Der TDA stellt dafür zwei Pins bereit, zwischen denen ein solcher Filter mit einer Frequenz von 10,7 MHz angeschlossen werden kann. Ohne einen hier extern angeschlossenen Filter würde der TDA als einfacher heterodyner Mischer direkt auf die Zwischenfrequenz $f_{IF2} = 274$ kHz heruntermischen. Bei Verwendung eines externen Keramik oder auch eines LC- π -Filters kann das ankommende HF-Signal jedoch in zwei Stufen gefiltert werden, ehe es in das Basisband demoduliert wird, was zu einer höheren Signalqualität führt. Die Umstellung zwischen einfacher und Double Down Conversion erfolgt durch das setzen eines dafür vorgesehenen Bits **TDA-DataSheet TDA-UserManual**

Die elektrischen Verbindungen zwischen den beiden symmetrischen Eingängen des Low Noise Amplifier und dem Anpassnetzwerk wurden mit dem „Differential Pair Routing“-Feature von Altium Designer erstellt. Durch einen im Schaltplan auf die positive und negative elektrische Leitung zwischen dem TDA5340 und dem Anpassnetzwerk angewendeten Parameter wird das Leitungspaar als differentiell markiert. Anschließend kann mit dem interaktiven „Differential Pair Routing“ einer der beiden Leitungen begonnen werden. Altium Designer wird dabei selbstständig versuchen, die zweite Leiterbahn des Paares so anzuordnen, dass die beiden Leiterbahnen symmetrisch und parallel zueinander liegen, sodass Störeinflüsse möglichst gleichmäßig auf die beide Leitungen einwirken.

So sollen Störungen besser toleriert und durch die Symmetrische Signalübertragung insgesamt ausgeglichen werden können **High-Speed-Guide**. Durch die Verwendung des „Differential Pair Routing“ versucht Altium Designer auch die Länge der beiden Verbindungen anzugleichen. Durch die Anpassungen der differentiellen Leiterbahnen wird eine korrekte Übertragung durch gleiche Signallaufzeiten sichergestellt. Da diese Leitungen hochfrequente Signale führen, ist eine genaue Anpassung notwendig.

Um Einkopplungen auf die Pfade für hochfrequente Signale zu vermeiden, wurde das für den Transceiver benötigte Quarz möglichst weit vom Sende- bzw. von den Empfangsanschlüssen des TDA angeordnet. Aus diesem Grund wurde der für den Oszillator benötigte Quarz mit einer Frequenz $f_{Crystal} = 21,948717 \text{ MHz}$ zwischen dem IC und der vorgesehenen Stiftleiste platziert. Die Frequenz des benötigten Quarzes ergibt sich aus dem Zusammenhang

$$f_{Crystal} = f_{IF2} \cdot 80 = \frac{f_{IF1}}{39} \cdot 80 = \frac{10,7 \text{ MHz}}{39} \cdot 80 = 21,948717 \text{ MHz} \quad (2.1)$$

wobei die Zwischenfrequenz der ersten Stufe (f_{IF1}) durch die internen funktionalen Blöcke des TDA und den Keramikfilter vorgegeben ist. Die weiteren Faktoren ergeben sich aus dem Aufbau des Empfängers und werden von Infineon bereitgestellt **TDA-UserManual**

2.3.2 XMC4500



Abb. 2.3: Der XMC4500 Mikrocontroller von Infineon im LQFP-Gehäuse mit 144 Pins
Bauer2012New-Infineon-32

Die Hauptsteuerung der Basisstation übernimmt ein Mikrocontroller der Bauart XMC4500, welcher aus der Mikrocontroller-Familie XMC4000 von Infineon stammt. Diese Baureihe stellt energieeffiziente ICs bereit, welche für industrielle Steuerungen und „Sense & Control“ optimiert sind. Der XMC4500 basiert auf einem Cortex™-M4 Kern des britischen Herstellers ARM. Daneben arbeitet der Mikrocontroller mit der von Infineon selbst entwickelte Entwicklungsumgebung DAVE zusammen. Im speziellen Anwendungsfall kommt die Variante des XMC mit 144 Pins und einem Flash-Speicher von 1024 Kilobit zum Einsatz. Der Chip ist dabei in einem Low Profile Quad Flat Package (LQFP)-Gehäuse verbaut. Durch die Wahl dieses Gehäuses konnte die elektrische

Verbindung mit der Platine relativ leicht durch löten erreicht werden. Im Gegensatz zum ebenfalls erhältlichen Ball Grid Array (BGA)-Gehäuse des XMC sind in diesem alle Kontakte direkt erreichbar und können leicht verlötet werden. Da beim BGA-Gehäuse die Anschlüsse auch in der Mitte unter dem Gehäuse sind, wäre hier ein Layout komplizierter und würde möglicherweise einen genaueren und somit teureren Prozess für die Herstellung der Platine fordern. Ein händisches Nachlöten von Kontakten oder Prüfen der Lötverbindung wäre ebenfalls nicht möglich.

Bei der Auswahl von Pins des XMC zur Verbindung mit den Transceivern wurde vor allem auf die Auswahl für die PP2 Pins geachtet. Um eine spätere Interruptsteuerung möglich zu machen wurden hierfür nur solche Eingänge des Mikrocontroller gewählt, die im Datenblatt mit der Möglichkeit zum Erkennen von Interrupts gekennzeichnet waren. Die Zuordnung der Transceiverausgänge an die Pins des Mikrocontroller sind in Abbildung 2.7 erkennbar und in Tabelle 3.3 aufgeführt. Tabelle 3.1 zeigt die Zuordnung der für die SPI-Kommunikation notwendigen Leitungen an die Pins des Mikrocontrollers.

Zur Verteilung der entstehenden Abwärme wurde auch in diesem Bereich der Platine frei gebliebene Abschnitte zwischen den Leiterbahnen mit geerdeten Kupferflächen gefüllt. Durch teilweise auch mehrfache Durchkontaktierungen wurde sowohl eine saubere Kontaktierung der Flächen durchgeführt, um Flächen schwimmenden Potentials zu vermeiden. Durch die Vias wurde aber auch die thermische Leitfähigkeit zwischen den beiden Kupferlagen erhöht und somit die Abgabe entstehender Wärme von den Bauteilen verbessert. Beim verwendeten LQFP-Gehäuse des XMC4500 liegt die Rückseite des Halbleiters offen und ist nicht im Gehäuse verschlossen. Im Bereich unter der offenliegenden Rückseite des Chips ist deswegen zur Wärmeableitung ein Feld von 6x6 Vias vorgesehen. Dieser Aufbau dient dazu die Temperatur des Chips (junction temperature) auf den maximal erlaubten Wert $T_J = 150^\circ\text{C}$ zu beschränken.

Um die Ausgabe von aktuellen Systemzuständen zu ermöglichen wurden sieben Status-LEDs an freien Ausgänge des XMC4500 angeschlossen. Diese ermöglichten in active-low Ansteuerung eine Anzeige verschiedener im Mikrocontroller ablaufender Prozesse. Für vier der verwendeten Leuchtdioden wurde grün als Farbe gewählt, für die drei weiteren rot. Zur Vereinfachung eines Resets der Hardware wurde ein entsprechender Taster vorgesehen, mit dem der entsprechenden PORST-Pin des Mikrocontroller auf das 0V Potential gezogen wird und somit die Hardware zurückgesetzt wird. Die Programmierung des Mikrocontrollers erfolgt über das Joint Test Action Group (JTAG)-Interface über welches auch das debuggen möglich ist. Der XMC4500 stellt dafür ein JTAG-Modul bereit, welches mit den in IEEE 1149.1 festgelegten Standards übereinstimmt. Verwendet wird hierfür die achtpolige Variante des Debug-Steckers, bei dem der Platine vom JTAG-Adapter Versorgungsspannung und Masse sowie Signale für Reset, Systemtakt und die Steuerleitung übergeben wird **XMC-DataSheet**

2.3.3 USB

Für die Kommunikation des Mikrocontrollers mit einem Computer wird die im XMC bereitgestellte Peripherie genutzt. Zur Verbindung mit einem anderen Gerät wurde deshalb eine kombinierte Micro-USB-Buchse verwendet welche sowohl für Typ A oder Typ B Stecker geeignet ist. Um sowohl den Mikrocontroller als auch einen an die Basisstation

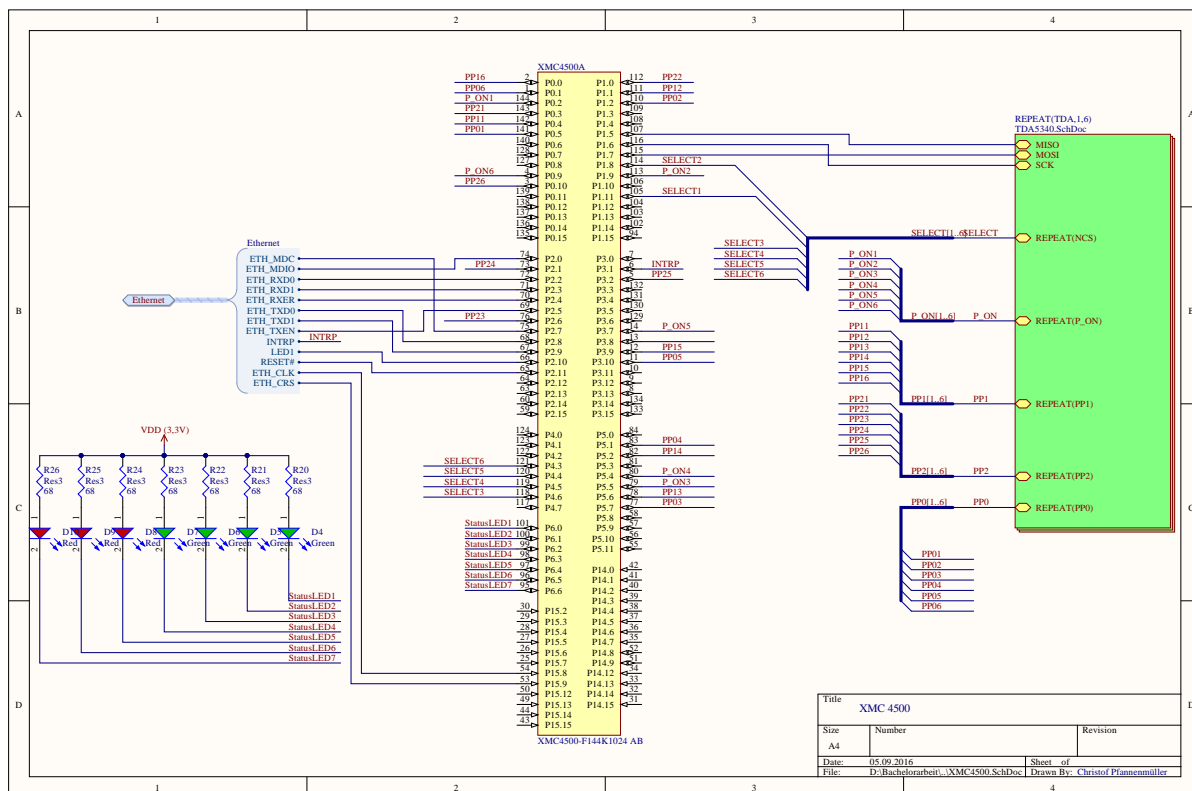


Abb. 2.4: Schaltplan des XMC4500 Mikrocontrollers

angeschlossenen Computer gegen Fehlerströme über die USB-Leitung zu schützen wurden die Datenleitungen mit so genannten Transient Voltage Suppressor (TVS)-Dioden, welche gegen Masse geklemmt sind, geschützt. Sowohl positive als auch negative Spannungsspitzen werden dadurch gegen Masse kurzgeschlossen, was zum Schutz des XMC bzw. des angeschlossenen Computer dient. Um eine Verpolung bei Stromversorgung über die USB-Buchse, und somit eine Zerstörung, zu vermeiden wurde eine Schottky-Diode im Strompfad zum Spannungsregler vorgesehen. Diese soll einen Stromfluss im Verpolungsfall unterbinden. Für den Fall das zeitgleich ein Netzteil sowie eine stromversorgende USB-Verbindung angeschlossen ist, dienen die Schottky-Dioden ebenfalls dem Schutz der Bauteile. Da der Mikrocontroller für die Kommunikation über das USB-Interface die aktuelle Busspannung auf der USB-Leitung benötigt, muss der extra dafür vorgesehene Pin des XMC direkt und ohne schützende Schottky-Diode mit der 5V Leitung der USB-Buchse verbunden werden.

2.3.4 Ethernet

Die Ethernetschnittstelle der Basisstation basiert auf dem Relax Kit von Infineon. Genau wie im Evaluations Board des Herstellers Infineon wurde der Ethernet-Controller KSZ8031RNL von Mircel Inc. verwendet. Dieser stellt alle wichtigen Peripherien selbst zur Verfügung und muss somit nur noch durch ein Quarz und diverse Kapazitäten und Induktivitäten an den Versorgungsleitungen ergänzt werden. Da die im Controller ver-

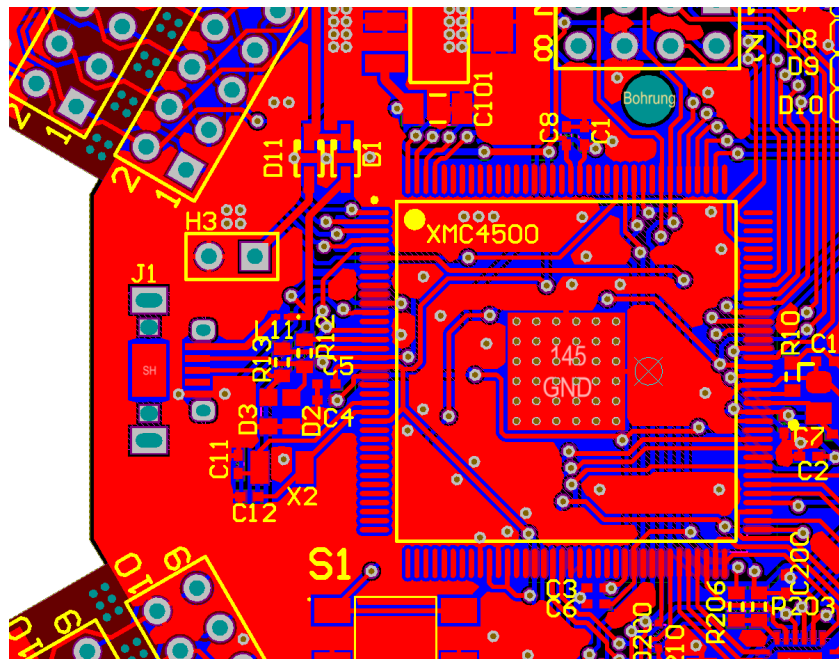


Abb. 2.5: Layout der USB-Buchse (links) auf der Basisstation mit der entsprechenden Schutzbeschaltung. In der Mitte des XMC sind die Thermal Vias zum Abführen der Wärme zu erkennen.

baute Stufe zur Interruptgenerierung nur über einen schwachen Pull-Up Widerstand verfügt, musste ein externer Widerstand von $1\text{ k}\Omega$ verbaut werden. Am Reset-Eingang wurde ebenfalls ein Pull-Up Widerstand verbaut. Dieser wurde um zwei Dioden sowie einen Kondensator zu der im Datenblatt empfohlenen Verschaltung erweitert. So kann sichergestellt werden, dass sowohl beim Anlegen einer Spannung an das Gesamtsystem als auch bei einem Reset des Ethernetbausteins durch den steuernden Mikrocontroller alle Spannungen im sicheren Bereich liegen und die Funktion gewährleistet ist. Die dreizehn zum XMC4500 notwendigen Verbindungen wurden zur besseren Übersicht im Schaltplan in einem Signal-Kabelbaum zusammengefasst. Wegen der Gefahr von Rissen in Lötstellen durch die Platinenbelastung beim Ein- und Ausstecken wurde im Bereich um den Netzwerkstecker die Anordnung von Bauteilen vermieden. Da der KSZ8031RNL nicht lieferbar war und die anfallende Datenmenge nur von geringem Umfang ist, wurden der Controller und die entsprechende Netzwerkbuchse von Würth Electronics zunächst nicht bestückt. Somit wurde eine Verwendung des Ethernet-Controllers auch in der Software des XMC-Mikrocontrollers nicht umgesetzt. Da jedoch ein entsprechendes Softwareprojekt für das Relax Kit von Infineon zur Verfügung gestellt wird, wäre eine Netzwerkkommunikation vermutlich mit wenigen Anpassungen schnell umzusetzen **DAVE-3-Example-**

2.3.5 Spannungsversorgung

Die Bereitstellung der notwendigen Spannung sollte wahlweise über den zur Datenerfassung angeschlossenen Computer oder über ein externes Netzteil erfolgen. Zum Anschluss eines externen Netzteils wurden Lötanschlüsse für eine Steckerleiste im Rastermaß $2,54\text{ mm}$

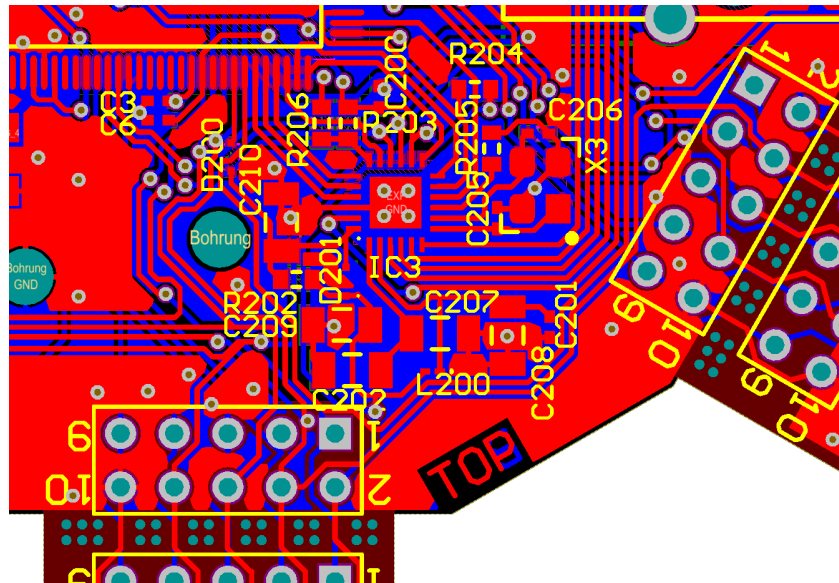


Abb. 2.6: Layout für den Ethernetcontroller von Mircel und die Anbindung an den XMC4500 Mikrocontroller

vorgesehen. Genau wie bei der Stromversorgung über die USB-Buchse wurde auch hier eine Schottky-Diode zum Verpolungsschutz der Schaltung integriert. Ausgelegt ist die Basisstation für ein Gleichspannungsnetzteil mit 5 V Ausgangsspannung, durch den Aufbau mit den beiden verwendeten Schottky-Dioden und die mögliche Eingangsspannung des nachfolgenden Reglers wäre jedoch auch eine angeschlossene 6 V-Versorgungsspannung (bei vernachlässigtem Spannungsabfall an der Diode) unproblematisch. Wegen der bereits erwähnten notwendigen Versorgungsspannung von 3,3 V für den XMC4500 und die Transceiver wurde diese mit einem Low Drop-Out (LDO)-Regler aus externen angeschlossenen Spannungsversorgung generiert. Dieser verwendete Spannungsregler der Bauart MCP1826S von Microchip Technology Inc. sollte die Eingangsspannung auf das gewünschte Niveau herunter regeln. Als Gehäusetyp wurde das dreibeinige Small Outline Transistor (SOT)-Package gewählt, da keine Variante des LDO mit einstellbarer Ausgangsspannung und somit keine Variante des ICs mit mehr Anschlusspins benötigt wurde.

Statt des LDO von Microchip war zunächst ein gleichwertiger Spannungsregler von Infineon, der IFX1117MEV33, vorgesehen. Die beiden LDOs unterscheiden sich in der elektrischen Belegung der Kühlfahne des SOT-223: beim Spannungsregler von Infineon ist diese mit dem 3,3 V Output kontaktiert, beim verwendeten LDO mit Ground. Da es sich bei dem Spannungsregler um ein SMD-Bauteil handelt, ist die Verwendung von Kühlkörpern schwer möglich. Die Abführung der im Spannungsregler erzeugten Verlustwärme erfolgt deshalb üblicherweise über das Anlöten der Kühlfahne an eine genügend große Kupferfläche, die als Wärmesenke dient. So wird die erzeugte Wärme gespreizt und kann gut an die Umgebung abgegeben werden.

Die im Spannungsregler umgesetzte Verlustleistung ergibt sich aus der vernichteten Spannungsdifferenz mal den Strom zu

$$P_V = (U_{Eingang} - U_{Ausgang}) \cdot I_{Basisstation} \quad (2.2)$$

wobei für den Gesamtstrom der Basisstation $I_{Basisstation}$ eine maximale Stromaufnahme des Mikrocontroller von 122 mA sowie bei den Transceivern eine Stromaufnahme von 26 mA angenommen werden kann **TDA-DataSheet XMC-DataSheet** Mit einem Spannungsabfall von 0,37 V an der Diode ergibt sich die Verlustleistung im LDO zu

$$P_V = ((5\text{ V} - 0,37\text{ V}) - 3,3\text{ V}) \cdot (122\text{ mA} + 6 \cdot 26\text{ mA}) = 367\text{ mW} \quad (2.3)$$

Beide Spannungsregler dürfen nur bis zu einer Temperatur des Halbleiters von 125 ° C betrieben werden. Bei der Verwendung des SOT-223-Gehäuses ohne eine Kupferfläche zum Abführen der Wärme würde die Chip-Temperatur T_j sogar bei einer Umgebungstemperatur von 25 ° C schnell kritische Werte erreichen, wie Gleichung 2.4 zeigt. Der zugrundeliegende Wärmewiderstand $R_{th,j-amb} = 164\text{ K/W}$ zwischen dem Halbleiter und der Umgebung, welcher durch das Gehäuse bestimmt wird, ist dem Datenblatt des Infineon Spannungsregler entnommen.

$$T_j = T_{amb} + \Delta T_{j-amb} = T_{amb} + P_V \cdot R_{th,j-amb} = 85,188\text{ °C} \quad (2.4)$$

Bei Verwendung des zuerst eingepplanten IFX1117MEV33 wäre wegen der elektrischen Belegung der Kühlfinne eine Kupferfläche auf 3,3 V Potential zum Kühlen notwendig, welche elektrisch isoliert sein müsste. Wegen des Platzbedarfs durch den XMC4500 und andere Bauteile wäre somit nur eine Kupferfläche mit Abmessungen von etwa 15 mm auf 16 mm möglich, da die elektrischen Verbindungen bestehender Bauteile des Bereiches nicht unterbrochen werden sollten. Auf diese Weise wäre nicht einmal die im Datenblatt empfohlene Kühlfläche von 300 mm² erreichbar.

Durch die Verwendung des entsprechenden Bauteils von Microchip konnte auf eine abgetrennte Kupferinsel verzichtet werden und somit die bereits erwähnte GND-Kupferfläche um den Mikrocontroller als gemeinsame Masse- und Kühlfläche verwendet werden. Wegen der vorderseitigen Bauteilbestückung war die verfügbare Kupferfläche auf der Platinenrückseite größer. Um dies beim Ableiten der Wärme vom Bauteil zu nutzen, wurden vor allem im Bereich um die Kühlfinne des SOT-223-Gehäuses Durchkontaktierungen angebracht. Diese parallelen „Thermal Vias“ konnten als Wärmepfad zur unteren Kupferfläche dienen. Außerdem wurde der JTAG-Stecker des XMC absichtlich im Bereich neben dem Spannungsregler angebracht. Durch die große Oberfläche stellt auch dieser eine gute Wärmesenke dar.

2.4 Generierte Dokumente

Altium Designer kann aus den erstellten PCB-Daten die für die weitere Verarbeitung benötigten Dateien generieren. Im dazu vorgesehene Output-Job-Manager können entsprechende Outputs gewählt werden und einem Output-Container zugeordnet werden. Dies ist vor allem zum Erstellen gewünschter Dateistrukturen bei größeren Projekten notwendig um diese übersichtlich zu halten. Im der vorliegenden Arbeit wurde dies jedoch nur bedingt benötigt. Für die Bestückung der Basisstation wurde zunächst eine „Bill of materials“, also eine Materialliste, exportiert mit deren Hilfe die entsprechenden Bestellnummern des Lieferanten Digi-Key herausgesucht und sortiert werden konnten. Mithilfe der ebenfalls generierten „Assembly Drawings“ war die Ausgabe aller Bauteile

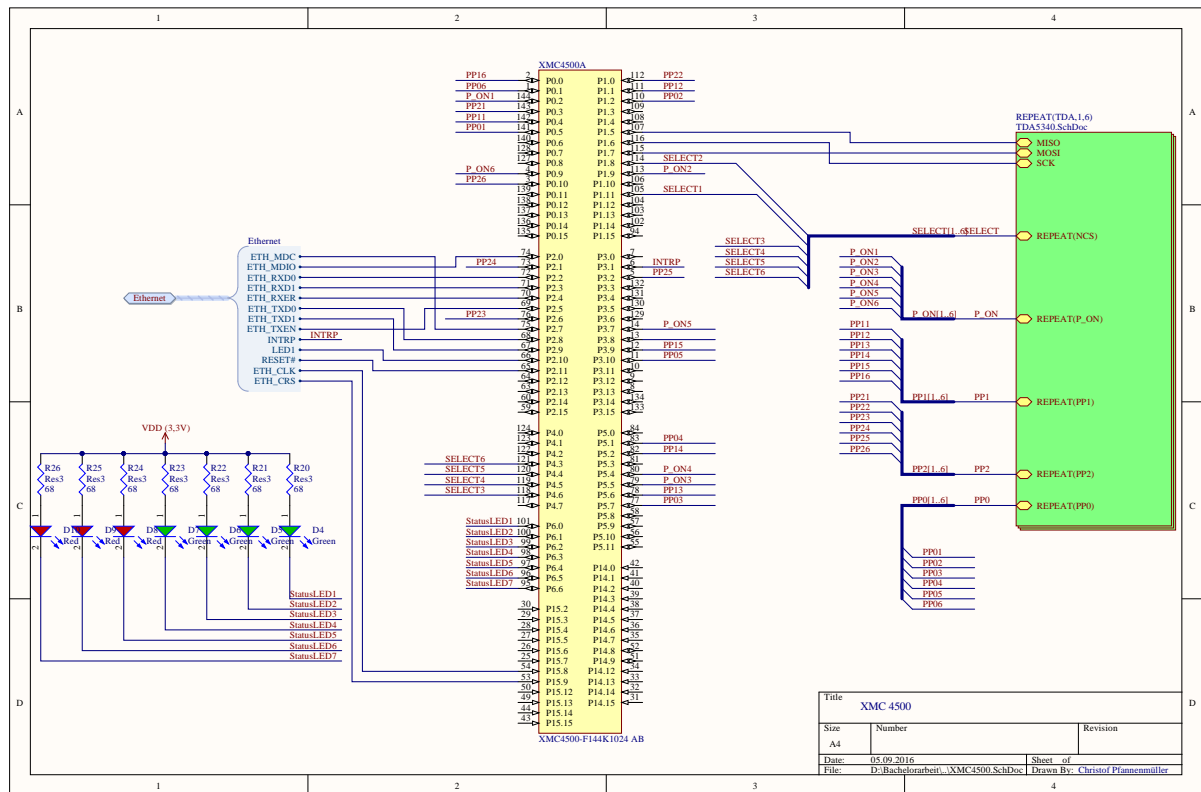


Abb. 2.7: Schaltplan des XMC4500 Mikrocontrollers

und deren Platzierung auf der Platine möglich. Ebenfalls wurden in diesem Menü die strukturierten Schaltpläne ausgegeben.

Zur dreidimensionalen Visualisierung wurde die fertige Platine mit den in den Footprints enthaltenen Bauteilabmessungen und Höhen als 3D-Modell im Standard for the Exchange of Product Model Data (STEP)-Format ausgegeben. Mithilfe des CAD-Programms „SolidWorks“ wurde daraus ein Gehäuse mit Deckel für die Basisstation erstellt. Diese wurde anschließend mit einem 3D-Drucker ausgedruckt und dient dazu, die Platine zu schützen.

Die Fertigung der Platine durch den Hersteller erfolgte durch so genannte Gerber Daten, die ebenfalls aus dem Output-Job-Manager generiert wurden. Dabei wird für jedes Layer des PCB-Editors eine Gerberdatei erstellt, in der die Geometrie der entsprechenden Lage angegeben ist. Jede Lage entspricht in der Herstellung dabei einem Fertigungsschritt. Um Bohrungen in der Platine zu setzen, werden zusätzliche „NC Drill-Files“ also Daten für die Numerical Control (NC) der automatischen Maschinen zum setzen von Bohrungen. Diese enthalten den Bohrdurchmesser, die Art der Bohrung sowie den Ort auf der Platine und müssen zusätzlich aus Altium Designer exportiert werden. Solche Bohrungen sind sowohl für Vias, als auch für Befestigungsbohrungen notwendig.

2.5 Bestückung

Die gefertigte Platine wurde vor der Programmierung mit den notwendigen Bauteilen bestückt. Dabei wurde zuerst der XMC4500 mit einer Bestückungsmaschine auf dem vorgesehenen Footprint der Platine verlötet. Im folgenden wurden alle weiteren Bauteile sowie die sechs Transceiver mit Lötpaste auf der Platine befestigt und anschließend durch Erhitzen der Platine auf der Heizplatte verlötet. Das kontaktieren durch die Heizplatte wurde in mehreren Schritten durchgeführt. Dabei wurden zunächst Bauteile einer guten Toleranz hoher Temperaturen wie Stecker und Widerstände bestückt. Temperaturanfällige Bauteile wie Leuchtdioden und integrierte Schaltkreise wurden nach Möglichkeit zu einem späterem Zeitpunkt befestigt. Wie bereits erwähnt wurde der Ethernet-Controller nicht bestückt.

Software

3.1 DAVE Entwicklungsumgebung

Das Programm DAVE (Digital Application Virtual Engineer) wird von Infineon Technologies AG entwickelt. Es basiert auf der Entwicklungsumgebung oder integrated development environment (IDE) „eclipse“ die von der Eclipse Foundation entwickelt wird. Eine IDE beschreibt dabei allgemein ein Programm zur Softwareentwicklung, welches die einzelnen dazu notwendigen Tools gesammelt zur Verfügung stellt. Dies sind vor allem der Compiler, der Linker, und der Debugger auf die im folgenden noch eingegangen werden soll. DAVE stellt eine Möglichkeit zum Editieren von Quelltexten und Anordnen der einzelnen Programmdateien bereit. Über den enthaltenen GNU C-Compiler wird der erstellte Quellcode in vom XMC lesbare Maschinensprache übersetzt und anschließend durch den Linker zu einem ausführbaren Programm vereint. Durch den enthaltenen Debugger kann das Programm in den Speicher des XMC geladen werden. Dort kann der Programmablauf gestartet und gestoppt werden, außerdem können Werte einzelner Register und Variablen ausgelesen werden **dausmann2011c**

DAVE greift bei der Programmierung von Mikrocontrollern der XMC-Serie auf die sogenannten XMC Libraries zurück, die von Infineon ebenfalls zur Verfügung gestellt werden. Auf diese soll ebenfalls im weiteren Verlauf eingegangen werden. Ein weiteres Feature in der IDE sind die sogenannten DAVE APPs. Mit diesen soll die Programmierung des Mikrocontrollers durch ein Graphical User Interface (GUI) ermöglicht werden. Dazu werden für mögliche, von der Hardware zu verrichtende Teilaufgaben, APPs von Infineon bereitgestellt. Durch das Einfügen der entsprechenden APPs in das Projekt können diese angepasst und miteinander grafisch verschalten werden. So wird der spätere Programmablauf im Mikrocontroller und dessen Aufgaben festgelegt. Nachdem vom Programmierer noch die Pins ebenfalls grafisch den Aufgaben zugeordnet werden, generiert DAVE den Programmcode mit den in den Apps enthaltenen Informationen **DAVEQuickStart** Mithilfe des DAVE Software development kit (SDK) können nicht nur die Parameter der APPs beim Programmieren, sondern auch diese selbst grundlegend angepasst werden und das Entwickeln eigener APPs ist möglich **DAVE-Version-4**

Im Verlauf dieser Arbeit wurden DAVE APPs jedoch nur in einem bereits existierenden

Softwareprojekt für ein Relax Kit genutzt, mit welchem Signale zum Testen der Empfänger an die Basisstation gesendet wurden. In der Basisstation selbst wurden die APPs jedoch nicht benutzt.

3.2 Verwendete Peripherie des XMC4500

Der XMC4500-Baustein enthält diverse funktionelle Blöcke, die mit einer Bus Matrix an die ARM Cortex-M4 Central Processing Unit (CPU) angebunden sind. Dieser Aufbau soll den Prozessor entlasten und im Programmablauf Ressourcen freihalten für andere Operationen. Für die Basisstation waren vor allem der USIC, der Universal Serial Bus (USB) sowie der Ethernet MAC (ETH) die bedeutende Peripherie. Von besonderer Bedeutung sind jedoch die General Purpose Input/Outputs (GPIOs).

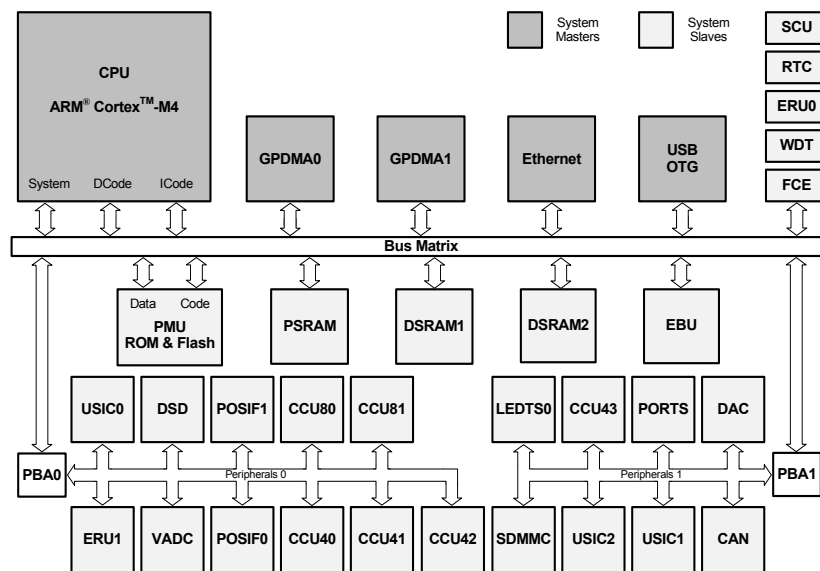


Abb. 3.1: Blockdiagramm zur Peripherie des XMC4500 Mikrocontroller und die Anbindung an den Cortex-M4 Kern **XMC-DataSheet**

3.2.1 GPIO

Die GPIOs werden im so genannten PORTS-Modul der XMC-Architektur gesteuert. In dieser lassen sich die Treiberstufen für die entsprechenden Pins des Mikrocontroller regeln. Dieses Modul ist über die Peripheriebrücke PBA1 ebenfalls an die Bus Matrix und somit den Cortex-M4 Kern angebunden.

Das Modul stellt für jeden Pin die erste Funktionsauswahl bereit und legt so die Datenrichtung fest. Im „Port Input/Output Control Register“ des Moduls wird für jeden Pin festgelegt, ob er als Eingang oder Ausgang verwendet wird. Das momentane elektrische Potential am Eingang wird bei letzterem mit einem Schmitt-Trigger in ein binäres logisches Signal übersetzt. Ist der Pin ein Eingang, so kann das logische Eingangssignal dort zusätzlich invertiert werden. Wird ein Pin des XMC als Ausgang konfiguriert, so kann

gesteuert werden, ob es sich um einen GPIO-Pin handelt, dessen Status von der Software direkt festgelegt wird. Dabei kann ausgewählt werden, ob das logische Ausgangssignal durch einen Treiber in Open-Drain-Konfiguration oder durch Push-Pull erzeugt werden soll.

Zur Nutzung eines Ausgangs durch die im Mikrocontroller verfügbaren Peripherie sind diese direkt mit den entsprechenden Modulen verbunden. Dadurch kann das Modul selbst den elektrischen Zustand am Eingang auslesen und verwerten oder festlegen wenn es sich um einen Ausgang handelt **XMC-Reference** Auch das weitere Verhalten von Pins, wie etwa beim Anschalten, bevor die Versorgungsspannung ein gültiges Level erreicht hat, lassen sich im PORTS-Modul anpassen.

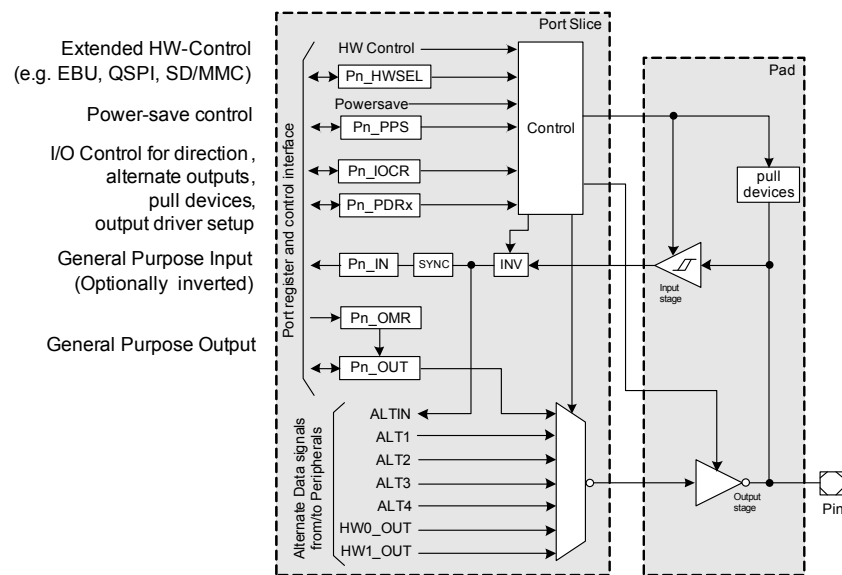


Abb. 3.2: Struktur eines Digitalpins des XMC Mikrocontroller **XMC-Reference**

3.2.2 USIC

Die ICs der XMC-Familie verfügen über ein Modul zu Kommunikation über diverse serielle Protokolle, den Universal Serial Interface Channel (USIC). Dieses ist programmierbar und erlaubt damit eine individuelle Verwendung, kann aber gleichzeitig die notwendigen Arbeiten für den Prozessor übernehmen. Der XMC4500 verfügt über insgesamt sechs USIC-Kanäle und kann somit mehrere Protokolle gleichzeitig verwenden. Die Mikrocontroller unterstützen die Protokolle UART, I²C, IIS, LIN und das für die Basisstation verwendete SPI in einfacher, doppelter und quad-Ausführung. Für diese Arbeit wurde alle Kommunikation mit einem gemeinsamen USIC-Kanal umgesetzt. Da die einzelnen USICs und deren Kanäle verschieden viele Slave-Select-Leitungen besitzen, wurde der Kanal 0 des USIC 0 ausgewählt, nur dieser verfügt über die benötigten sechs Select-Leitungen. Eine Umsetzung mit einem anderen USIC wäre ebenfalls möglich gewesen. Die Wahl des Transceivers hätte dann aber manuell erfolgen müssen und hätten nicht vom Modul geregelt werden können.

Signal	Port/Pin des XMC
MISO	1.5
MOSI	1.7
SCK (Clock)	1.6
Select TDA1	1.11
Select TDA2	1.8
Select TDA3	4.6
Select TDA4	4.5
Select TDA5	4.4
Select TDA6	4.3

Tab. 3.1: Zuordnung der für die SPI-Kommunikation notwendigen Signale an die Pins des Mikrocontrollers. Alle Leitungen sind mit dem Kanal 0 der USIC 0 verbunden.

3.2.3 ERU

Von zentraler Bedeutung für die Funktion der Basisstation war die Behandlung von Interrupts durch den IC. Der XMC4500 besitzt dafür zwei entsprechende Event Request Unit (ERU)-Module, die einen solchen erkennen können und den Prozessor zum Aufrufen einer Interrupt Service Routine (ISR) auffordern können. Jedes Modul verfügt über vier Kanäle, auf denen bei einem Interrupt ein vierstufiger Prozess durchlaufen wird: In der ersten Stufe der ERU, der so genannten Event Request Select (ERS), lassen sich aus zwei Eingängen mit jeweils vier Signalen die gewünschten Eingänge wählen. In der Event Trigger Logic (ETL) generiert der IC aus dem Signalstatus ein Trigger-Event, indem Veränderungen erkannt werden. So kann eine fallende oder steigende Flanke, die einen Interrupt auslösen soll, erkannt werden. In der Cross Connection Matrix können Signale der verschiedenen ETLs zu den vier Output Gating Units (OGUs) weitergeleitet und somit dort untereinander und mit Triggersignalen von anderen Peripherie-Modulen des XMC kombiniert werden. In der OGU wird durch Vergleich der verschiedenen aufgetretenen Trigger und Muster entschieden, ob ein kompletter Interrupt aufgetreten ist und leitet diese Information entsprechend weiter oder ob etwa nur das gewählte Muster erkannt wurde, was für andere Module wichtig ist. Diese Informationen werden entsprechend an die Peripherie weitergeleitet, sind aber für die Funktion der Basisstation nicht weiter von Bedeutung. Bei Vorliegen aller Bedingungen für einen Interrupt wird diese Information an den Nested Vectored Interrupt Controller (NVIC) im Cortex-M4 weitergeleitet. Dieser Teil des Prozessorkerns erkennt die Interruptaufforderung und sorgt dafür, dass der aktuelle Prozessorstatus gespeichert wird. Nach Ablauf der ISR wird der Prozessorstatus wieder hergestellt.

3.2.4 USB

Das USB-Modul des XMC4500 arbeitet nach den Spezifikationen für USB 2.0 und den „On-The-Go“-Spezifikationen der Version 1.3. Der Mikrocontroller könnte durch das USB-Modul entweder als Host oder als USB-Slave arbeiten. In diesem Fall wurde der IC als Slave betrieben. Das USB-Modul verfügt über eine eigene Interruptsteuerung und ist damit eine der oben erwähnten Peripherien des XMC, deren Steuerung auch über

Interrupts gelöst ist. Die gesamte Übertragung wird durch den USB-Kern gesteuert und empfangene oder zu sendende Pakete werden in einem FIFO-Puffer gespeichert. Für die Kommunikation der Basisstation mit dem Host-Computer wird ein virtueller COM-Port durch das USB-Interface emuliert.

3.2.5 Ethernet

Im XMC4500 werden Netzwerkverbindungen durch das Ethernet-Modul behandelt. Dieses unterstützt Datenübertragungen mit Geräten über IPv4 und IPv6 sowie Übertragungsraten von 10/100 MBit/s. Dazu werden zunächst die Daten von der CPU über ein Bus-Interface übertragen. Im „MAC Transaction Layer“ werden die vom Prozessor bzw. über Ethernet empfangenen Datenpakete zwischengespeichert. Der Ethernet-Kern formatiert die zu sendenden Daten und stellt sie einem „Physical Layer“ zu, welches die Daten für den Kanal moduliert.

3.3 Verwendete Bibliotheken

3.3.1 XMC Library (XMC Lib)

Infineon stellt für seine ICs der XMC4000 Serie, zu der auch der XMC4500 gehört, die „XMC Peripheral Library“ bereit. Diese erlaubt einen vereinfachten Zugriff auf alle Module und die entsprechenden Register und soll dadurch den Modulzugang übersichtlich gestalten und den Programmcode vereinfachen und leichter lesbar machen. Die Software baut auf dem Cortex Microcontroller Software Interface Standard (CMSIS) auf, erlaubt die Verwendung verschiedener Compiler und kann mit oder ohne DAVE bzw. mit oder ohne DAVE APPs verwendet werden. Für die Programmierung wurde mit der Software „doxygen“ eine Dokumentation zur Bibliothek als HTML generiert.

3.3.2 SPI Library

Die verwendete Bibliothek zur Steuerung des SPI-Interfaces basiert auf dem SPI-Modul der XMC Library. Die von Infineons XMC Library zur SPI-Kommunikation zur Verfügung gestellten Funktionen steuern das USIC-Modul des Mikrocontrollers an. Die Bibliothek zur SPI-Kommunikation muss somit nur noch die Funktion

```
1 XMC_SPI_CH_Transmit(channel, data, XMC_SPI_CH_MODE_STANDARD);
```

der XMC Library aufrufen, um eine Übertragung über das Serielle Interface durchzuführen. Daneben liegt die Hauptaufgabe der Bibliothek vor allem in der Auswahl des entsprechenden Transceivers über das Slave-Select-Signal. Dazu initialisiert die Bibliothek zuerst den USIC entsprechend den in der Headerdatei vorgegebenen Pins für MISO, MOSI und den Pins zur Auswahl des jeweiligen Slaves.

SPI Übertragung

Die durch die Software umgesetzte serielle Übertragung zwischen den ICs entspricht dem SPI-Protokoll, das ursprünglich von Motorola entwickelt wurde. **BuchSPI** Durch das

Chip Select-Signal wählt der XMC den entsprechenden Transceiver aus. Da der TDA5340 active-Low arbeitet, also bei anliegendem Masse (GND)-Potential als ausgewählt gilt, wird dieser Eingang im Datenblatt auch als Non-Chip-Select (NCS) bezeichnet.

Der USIC des XMC beginnt nun die Clockleitung des Busses zu treiben. Eine Periode dieser zyklischen Rechteckspannung begrenzt dabei die Zeit in der das auf den Master-Out Slave-In (MOSI) und Master-In Slave-Out (MISO)-Leitungen ein Bit übertragen wird. Hierbei existieren verschiedene Konfigurationen: so kann die Clock-Leitung im Ruhezustand einen High- oder einen Low-Pegel aufweisen, was durch die Polarität (CPOL) festgelegt wird. Das Auslesen der Spannungswerte auf den Datenleitungen kann bei fallender und steigender Taktflanke (CKPHA) durchgeführt werden, woraus sich insgesamt vier Kombinationsmöglichkeiten ergeben.

Dabei stellt nach der Definition aus dem Datenblatt des Transceivers ein 0 V-Spannungspegel eine logische 0 dar. In der Kommunikation mit den TDA5340 werden jeweils acht Bit zu einem Datenwort zusammengefasst. Der Transceiver erwartet mit dem ersten Datenwort eine Instruktion und je nach Art dieser noch einen Parameter durch die Übertragung des folgenden Wortes. Mögliche Anweisungen sind dabei das Lesen oder Schreiben eines Registers dessen Adresse als Parameter übertragen wird, das Auslesen oder Beschreiben der FIFO-Speicher oder das Setzen des TDA5340 in den so genannten „Transparent Mode“ auf den nicht weiter eingegangen werden soll. Nach Übertragung von Instruktion und Parameter werden entweder vom Master oder vom Slave, also vom Transceiver, die gewünschten Daten übertragen.

Instruktion	Datenwort	übergebener Parameter
Register Schreiben (schnell)	0x01	Adresse des ersten zu schreibenden Registers
Register Schreiben	0x02	Adresse des zu schreibenden Registers
Register Lesen	0x03	Adresse des zu lesenden Registers
FIFO Lesen	0x04	<i>keinParameter</i>
Register Lesen (schnell)	0x05	Adresse des ersten zu lesenden Registers
FIFO Schreiben	0x06	Länge der Übertragung -1

Tab. 3.2: SPI-Instruktionen des Transceivers und die entsprechenden auf der MOSI-Leitung übertragenen Datenwörter und Parameter. Die beiden Instruktionen zum transparenten Senden sind nicht enthalten.

Beim Schreibzugriff auf den TDA kann durch geeignete Instruktion gewählt werden ob nur auf ein Register geschrieben wird oder ob der Transceiver nachfolgende Datenwörter als Werte in die folgenden Register übernehmen soll. Beim Lesezugriff besteht die selbe Auswahlmöglichkeit. Bild 3.3 zeigt den Start einer solchen SPI-Kommunikation zwischen dem Transceiver TDA1 und dem Mikrocontroller. Die Dauer der Übertragung betrug etwa 0,26 ms. Bei dieser wurde die Instruktion „Read from Chip“, dargestellt durch den Hexadezimalwert 0x03 und die Registeradresse 0xD3 als Parameter vom Mikrocontroller zum Transceiver übertragen. Dieser teilt durch das folgende Datenwort auf der MISO-Leitung, in diesem Fall 0xFF den aktuellen Wert des Registers mit. Der Master beendet daraufhin die Kommunikation durch einen High-Pegel an der Select-Leitung

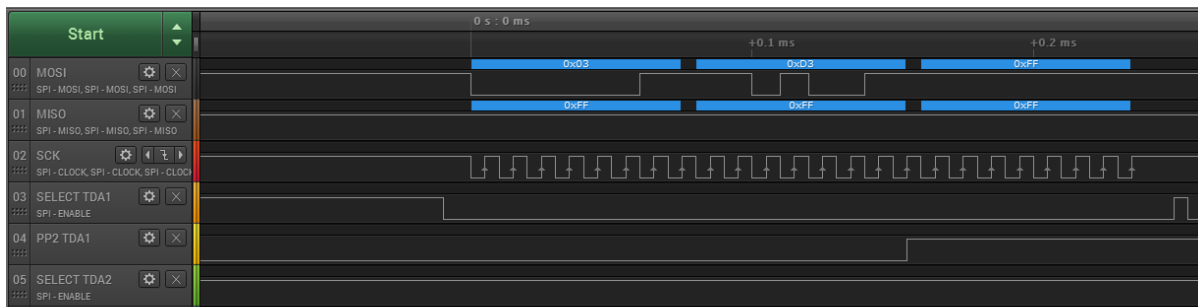


Abb. 3.3: Start einer SPI-Übertragung zwischen dem Mikrocontroller und dem Transceiver

3.3.3 TDA5340 Library

Die Hauptaufgabe in der Kommunikation mit den Transceiver-ICs wurde durch die vorgegebene Bibliothek für den TDA5340 übernommen. Diese stellte Funktionen zum Senden und Empfangen von Daten mit dem Transceiver zur Verfügung. Die Bibliothek liest dazu den Empfangs-FIFO aus oder schreibt in den Puffer für zu sendende Daten. Auch der Lese- und Schreib-Zugriff auf die Steuerregister der Transceiver kann über die Bibliothek geregelt werden. Dazu stellt die Bibliothek auch entsprechende Makros bereit, welche die Namen der Register in die hexadezimale entsprechende Adresse umwandeln um so die Lesbarkeit erheblich zu erhöhen. Daneben wurde über die Bibliothek auch die notwendigen Einstellungen für das Erkennen von Interrupt im XMC4500 vorgenommen. In diesem Bereich waren die notwendigen Anpassungen der Library am tiefgreifendsten, da diese nur für die Interruptbehandlung mit einem Transceiver ausgelegt war. Bei anderen Funktionen der Bibliothek waren nur kleinere Anpassungen notwendig, sodass etwa sichergestellt wurde, für welchen Transceiver die aufgerufene Funktion ausgeführt werden sollte, etwa bei welchem das entsprechende Register ausgelesen wurde. Zur Verbindung mit den TDA5340 basierte die Bibliothek auf der SPI Library. Dieser wurde die Nummer des Transceivers als Chip-Select übergeben um sicherzustellen, dass mit dem richtigen kommuniziert wurde.

3.3.4 Virtueller COM Port

Die Kommunikation der Basistation mit dem Hostcomputer zum Übertragen der gemessene Werte wurde nach dem Vorbild eines Beispielprojektes für DAVE umgesetzt. Die Bereitstellung des virtuellen seriellen Ports erfolgt auf Seiten des XMC über die LUFA (Lightweight USB Framework for AVR)-Bibliothek. Mit dieser beschränkt sich die Ausgabe über den COM-Port auf das Übergeben der zu sendenden Zeichen an eine entsprechende Funktion. Eine Steuerung des XMC durch Empfangen von Daten über den COM-Port wäre mit der Bibliothek ebenfalls möglich, war jedoch nicht notwendig. Die Bibliothek und das Beispielprojekt wurde dahingehend angepasst, dass ganze Zeichenketten statt nur einzelner Zeichen der Funktion zum Senden übergeben werden konnten. Auch wurde das Senden von Integer-Variablen ermöglicht, indem diese zu Zeichen umgewandelt wurden. Dabei wurde der American Standard Code for Information Interchange (ASCII) beachtet, nach dem die Übertragung funktioniert. So benötigt der COM-Port zur Ausgabe

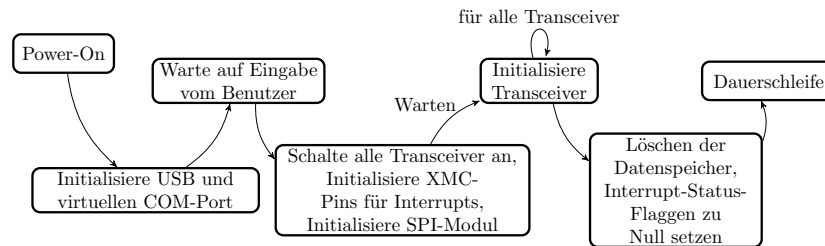


Abb. 3.4: Programmablauf der Basisstation

der Ziffer 1 den dezimalen Wert 49 nach der Vorgabe durch den ASCII-Code.

3.4 Programmablauf

Im Programmablauf des XMC wurde zunächst eine Warteschleife umgesetzt, in welcher der Mikrocontroller auf eine erste Eingabe durch den Benutzer am Hostcomputer wartet. Bereits hier wurde die Übertragung über den seriellen COM-Port initialisiert. Nachdem diese Bestätigung über den COM-Port vom Mikrocontroller empfangen wurde setzt dieser zunächst alle Transceiver in den Schlafmodus, von welchem aus eine Kommunikation möglich ist. Daraufhin beginnt der XMC4500 mit der Konfiguration der für die Interrupts notwendigen Pins und ermöglicht somit das Empfangen von Interrupt Requests (IRQs) durch die TDA5430. Anschließend initialisiert er zunächst das SPI-Modul, um im folgenden die Transceiver darüber konfigurieren zu können. Bevor das dazu notwendige Schreiben in die Register der TDAs jedoch möglich ist, wird eine gewisse Verzögerung benötigt. Diese resultiert daher, dass der Wechsel des TDA vom ausgeschalteten Zustand in den Schlafmodus eine gewisse Zeitspanne benötigt. Im folgenden werden alle Transceiver initialisiert und anschließend in den Receiver-Modus gefahren. Daraufhin setzt der IC noch alle für die Übertragung notwendigen Variablen und alle Felder zum Speichern von Daten zu null. Nun beginnt der Prozessor mit einer Dauerschleife, in der dieser auf das Ankommen von Übertragungen wartet und teilt dies auch dem Benutzer am Hostcomputer über eine Ausgabe mit. In dieser Endlosschleife wechselt sich die interruptbasierte Datenerfassung mit der Weiterleitung der erfassten Daten zyklisch ab, wobei nur bei erfolgreichem Empfang ein Senden an den steuernden Computer erfolgt.

Bei der Programmierung wurde darauf geachtet möglichst alle Konstanten wie Adressen von Pins und Ports durch Makros für den Präprozessor auszulagern. Dadurch konnte einerseits die Lesbarkeit des Quellcodes erhöht werden, andererseits sind so aber Änderungen durch vertauschen von Pins, sowohl in der Hardware als auch nur in Software, leichter möglich. Da es sich beim XMC4500 um einen Mikrocontroller mit einer 32Bit Architektur handelt wurden Integervariablen ausschließlich vom Typ `uint32_t` oder `int32_t` verwendet. So konnte eine genaue Anpassung der Software an die Registergrößen im XMC vorgenommen werden.

3.4.1 Konfiguration der Funkmodule

Bei der Initialisierung erhalten die Transceiver die gewünschten Werte für die Sende- und Empfangs-Frequenzen. Diese werden über die Teilerate für die Phasenregelschleife (PLL) übergeben und ermittelt. Über die entsprechenden Register wird auch das Verhalten bei ankommenden Übertragungen eingestellt. Der TDA5340 stellt dafür mehrere Konfigurationssätze zur Verfügung in denen unterschiedliche Frequenzen eingestellt werden können. In diesem Fall wird zum Empfangen die Konfiguration A und Konfiguration B zum Senden eingestellt. Die entsprechenden Werte wurden zuvor aus der Zwischenfrequenz $f_{IF2} = 274 \text{ kHz}$ und der gewünschten Übertragungsfrequenz errechnet.

Im „Interrupt Mask“-Register wurde eingestellt bei welchen Ereignissen der Transceiver einen solchen auslösen wird. Zur Ausgabe dieser wurde der standartmäßige PP2-Pin des Transceivers verwendet. PP1 und PP0 wurde in einen hochohmigen Zustand versetzt, da diese zum aktuellen Zeitpunkt nicht genutzt wurden.

3.4.2 Interruptbasierte Datenerfassung

Die Erfassung der Daten erfolgt im Programmablauf innerhalb der dauerhaften Ablaufschleife. In der Interrupt Service Routine (ISR), in welche der Prozessor beim Auftreten eines Interrupts springt, wird lediglich einer globalen Variable der Wert 1 zugeordnet. Für jeden der möglichen Transceiver existiert eine solche Flagge, die einen aufgetretenen Interrupt anzeigt. Nach dem Setzen in der ISR kehrt der Prozessor zum Ablauf in die Dauerschleife zurück. Innerhalb dieser wird nun zyklisch abgefragt, ob diese Flagge gesetzt wurde. Beim Auftreten einer solchen Flagge, also nach dem aufgetretenen Interrupt, liest der XMC4500 die Interrupt Status Register des entsprechenden Transceivers aus. Da davon auszugehen ist, dass alle sechs Transceiver-ICs zeitgleich eine Übertragung erhalten, wurde diese mehrstufige Abfrage gewählt. So wird zuerst nur in der ISR die Flagge gesetzt, um die dadurch verstreichende Zeit möglichst kurz zu halten und zu ermöglichen, dass eine solche Flagge auch jederzeit im Programmablauf gesetzt werden kann. In der zweiten Stufe liest der Mikrocontroller nun die drei Interrupt Status Register aus. Dies ist notwendig, da es sich dabei um so genannte „Read-Clear“-Register handelt, welche nach dem Auslesen über SPI automatisch zurückgesetzt werden. In den Interrupt-Registern sind die Ereignisse kodiert, die einen Interrupt ausgelöst haben. Bei der Konfiguration der Funkmodule wurde eingestellt bei welchen Ereignissen der Transceiver einen Interrupt auslöst. Typische Ereignisse sind ein fast gefüllter Empfangs-FIFO was eine angekommene Übertragung anzeigt oder auch ein leerer FIFO was anzeigt, dass dieser ausgelesen wurde. Die dritte Stufe der Datenerfassung ist nun die Abfrage der Daten vom Transceiver. Da diese SPI-Datenübertragung deutlich mehr Zeit in Anspruch nimmt, muss diese getrennt vom Erkennen der Interrupts und dem Auslesen der Interrupt-Register erfolgen. Da die gewünschten Werte in den Registern gespeichert sind, ist das Auslesen zeitkritisch, denn eine erneute Übertragung würde diese Messwerte überschreiben. Jedoch ist das Auslesen der Werte bei weitem nicht so zeitkritisch wie die ankommenden Interrupts, da diese im Verlauf einer Übertragung auch mehrfach ankommen können. Das Abfragen der empfangenen Daten aus dem FIFO-Speicher hat eine noch geringere Priorität, da dieser die Datenpakete bis zum Auslesen behält. Eine Abfrage der Messwerte und ein Auslesen des FIFO wird bei abgeschlossener Übertragung durchgeführt, diesen Zeitpunkt

erkennt der Mikrocontroller durch das entsprechend gesetzte Bit des Interrupt-Register. Der XMC4500 speichert alle abgefragten Werte wie Feldstärke, die Einstellungen der automatic gain control (AGC) und die angekommenen Daten in den vorbereiteten dedizierten Speichern. Abschließend werden die Transceiver wieder in den Empfangsmodus gesetzt.

Beim Setzen der Flaggen durch den Interrupt fiel auf, das die Interrupteingänge für Transceiver 3 und Transceiver 6 sich überlagern. So ist der PP2 Anschluss des dritten TDA5340 mit der ETL3 der ERU0 verbunden, ebenso wie der entsprechende Pin des TDA6. Grund dafür ist, die entsprechenden Pins dem Mikrocontroller auf dem Kanal B3 bzw. A0 der genannten ERU-ETL-Kombination führen. Bei den ersten Tests von Interrupts auf diesen Kanälen stellte sich heraus, das der XMC, entgegen der Erwartungen, somit nicht in der Lage ist ankommende Interrupts der beiden Transceiver zu unterscheiden. Durch entsprechende Einstellungen in der ERU0 war ein Sprung des Prozessors in die entsprechende Interrupt Service Routine nur für jeweils einen der beiden Transceiver möglich. Um trotzdem eine Auslesen der beiden Transceiver zu bewirken, wurden beide Flaggen in der Service Routine gesetzt. Der Versuch den PP2 Pin des TDA6 mittels

Transceiver	Port/Pin am XMC	ERU	ETL	Kanal
TDA1	0.3	1	3	B0
TDA2	1.0	0	3	B0
TDA3	2.6	0	1	B3
TDA4	2.1	1	0	B0
TDA5	3.2	0	0	A1
TDA5	0.10	0	1	A0

Tab. 3.3: PP2-Pins der Transceiver mit den entsprechenden Anschlüssen am XMC4500 und die Verbindung zur ERU

eines kurzen Drahtes auf den Pin 142 des XMC4500 zu legen schlug ebenfalls fehl. Dazu wurde der PP1 Pin des TDA1, welcher mit diesem Pin des Mikrocontroller verbunden ist im Transceiver 1 in einen hochohmigen Zustand geschaltet. Dieser Pin des XMC ist intern mit der ETL2 der ERU0 auf dem Kanal B3 verbunden. Da diese Kombination noch nicht verwendet wurde hätte hier eine Interrupterkennung funktionieren müssen. Warum durch diese Veränderung kein Interrupt ausgelöst werden konnte, ist nicht klar. Da in der Betrachtung mit dem Logic-Analyser keine Flanke auf dem Netz festgestellt wurde, ist zu vermuten, dass das Problem aus der Verbindung mit dem Draht resultiert.

3.4.3 Weiterleitung der erfassten Daten

Dass eine Übertragung über den COM-Port bereits weit vor der Dauerschleife vom Mikrocontroller gestartet wurde, diente dazu, dem Benutzer die Bereitschaft zum Empfangen mitzuteilen sobald sämtliche Initialisierungen abgeschlossen waren. Somit musste der COM-Port auch nun nicht mehr selbst initialisiert werden. Im zweiten Teil der Ablaufschleife des Programmablaufs wurde nun die Weitergabe der empfangenen Daten und der gemessenen Werte behandelt. Dazu wurde nach dem Abholen der empfangenen Daten von den Transceivern eine Statusflagge in Form einer Integer-Variable gesetzt. Nur

beim Auftreten dieser Flagge wurde der Programmteil zum Senden über den COM-Port ausgeführt. Dort wurde nun jeweils abwechseln ein Wert des Speichers und ein String mit einer Beschreibung oder dem Namen des Wertes über COM ausgegeben. Ein Ausschnitt des Quellcodes ist im Quellcode 3.1 zu erkennen. Die Zeichenfolge `\r\n` stellt dabei den Übergang in eine neue Zeile in der Ausgabekonzole dar. Wie in C üblich werden Strings in doppelten Anführungszeichen im Quelltext eingefügt. Eine entsprechende Ausgabe ist in Bild ?? zu erkennen.

```

1  COM_send_string("##### Übertragung erkannt #####\r\n");
2  COM_send_string("Übertragung Nummer ");
3  COM_send_int_as_string(transfervnummer);
4  COM_send_string("\r\n\r\n");
5  COM_send_string("TDA1:");
6  COM_send_string("\r\nPMF:");
7  COM_send_int_as_string(rssiTDA1.pmf);
8  COM_send_string("\r\nPRX:");
9  COM_send_int_as_string(rssiTDA1.prx);
10 COM_send_string("\r\nRX:");
11 COM_send_int_as_string(rssiTDA1.rx);
12 COM_send_string("\r\nPPL:");
13 COM_send_int_as_string(rssiTDA1.ppl);
14 COM_send_string("\r\nAGC:");

```

Quellcode 3.1: Ausschnitt aus dem Senden der Daten über den COM-Port

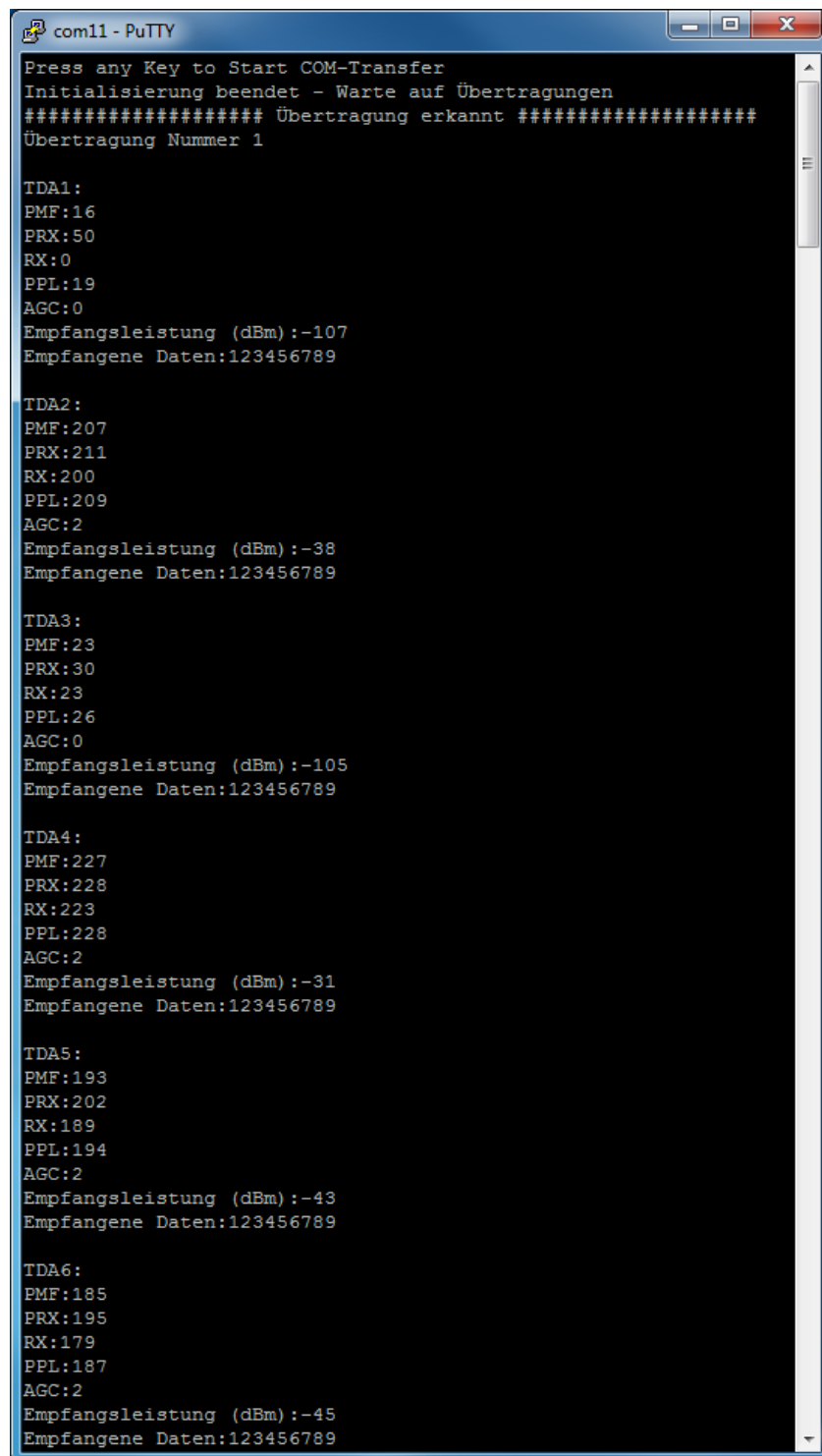
Am Hostcomputer wurden die Daten mit der Software „PuTTY“ entgegengenommen und dem Benutzer in einer Konsole angezeigt. Dazu wurde der von Infineon zur Verfügung gestellte Treiber verwendet, um dem XMC4500 als COM-Port zu erkennen. Die Einstellungen der Seriellen Übertragung wurden vom Beispielprojekt übernommen, sodass am Computer mit einer Baudrate von 115200 Bd und acht Daten Bits pro Zeichen empfangen wurde und was im Programm eingestellt werden musste. Nach der bereits erwähnten anfänglichen Bestätigung der Kommunikation durch den Nutzer, wurden bei jeder vom Mikrocontroller erkannten Übertragung die gemessenen Feldstärkewerte ausgegeben. Neben den PMF, PRX, RX, PPL und AGC-Werten wurde noch die Empfangsleistung ausgegeben. Diese wurde zur Laufzeit aus dem PPL und dem AGC-Wert errechnet und in dBm angezeigt. Dazu diente eine vorgegebene Funktion, welche aus dem PPL-Wert und dem Wert der AGC mittels kalibrierter Parameter eine Feldstärke berechnete. Die Werte wurden in einem Vorprojekt durch Messungen kalibriert und sind hier nicht weiter von Bedeutung. Die Abkürzungen der übertragenen Werte sind in Tabelle 3.4 aufgeführt. Die

PMF	Peak Memory Filter
PRX	Peak Detector
RX	Received Signal Strength Indication (RSSI)
PPL	RSSI Payload Peak Detector
AGC	Automatic Gain Control

Tab. 3.4: Verwendete Abkürzungen der übertragenen Messwerte

empfangenen Daten wurden ebenfalls ausgegeben. Alle Werte und Daten wurden nach Transceiver getrennt ausgegeben, um eine Vergleichsmöglichkeit zu geben und um die Ausgabe möglichst übersichtlich zu gestalten. Auch wurden die empfangenen Übertragungen durchnummeriert und die entsprechende Übertragungsnummer mit ausgegeben. So ließen sich einerseits die Sendepositionen den gemessenen Werte zuordnen. Andererseits

waren so aber auch verlorengegangene Übertragungen sichtbar. Eine typische Ausgabe der Konsole ist im Bild 3.5 zu erkennen. Zur Darstellung einer eingegangenen Übertragung an der Basisstation sollte die LED Nummer 7 nach jedem Empfangen kurz rot aufblinken.



```
com11 - PuTTY
Press any Key to Start COM-Transfer
Initialisierung beendet - Warte auf Übertragungen
##### Übertragung erkannt #####
Übertragung Nummer 1

TDA1:
PMF:16
PRX:50
RX:0
PPL:19
AGC:0
Empfangsleistung (dBm):-107
Empfangene Daten:123456789

TDA2:
PMF:207
PRX:211
RX:200
PPL:209
AGC:2
Empfangsleistung (dBm):-38
Empfangene Daten:123456789

TDA3:
PMF:23
PRX:30
RX:23
PPL:26
AGC:0
Empfangsleistung (dBm):-105
Empfangene Daten:123456789

TDA4:
PMF:227
PRX:228
RX:223
PPL:228
AGC:2
Empfangsleistung (dBm):-31
Empfangene Daten:123456789

TDA5:
PMF:193
PRX:202
RX:189
PPL:194
AGC:2
Empfangsleistung (dBm):-43
Empfangene Daten:123456789

TDA6:
PMF:185
PRX:195
RX:179
PPL:187
AGC:2
Empfangsleistung (dBm):-45
Empfangene Daten:123456789
```

Abb. 3.5: Ausgabe des Mikrocontroller über den COM-Port in der Konsole bei einer beispielhaften Übertragung

Feldtest

4.1 Aufbau

Zur Evaluation der Basisstation wurde ein XMC4500 Relax Kit von Infineon mit einem aufgesteckten Evaluations-Board für den TDA5340 betrieben. Mit Hilfe einer Powerbank konnte dieses mobil über den USB-Anschluss des Relax Kit betrieben werden. Dieses wurde auf eine Sendefrequenz von 868,0 MHz und eine Empfangsfrequenz von 867,999 MHz programmiert was durch die Werte für die PLL im TDA5340 eingestellt wurde. Die Basisstation wurde mit sechs Antennen bestückt, die einen Verstärkungsfaktor von 3,6 dBi und eine Mittenfrequenz von 868 MHz aufwiesen. Die Basis wurde über USB an den Computer zur Auswertung angeschlossen. Das Auslesen der durch virtuellen COM-Port übertragenen Daten erfolgte mit PuTTY. Die Messungen fanden innerhalb des Gebäudes statt.

4.2 Durchführung

Es wurden im selben Raum von diversen Positionen durch einen Tastendruck am Relax Kit eine Übertragung ausgelöst. Dabei wurde die zuvor einprogrammierte Zeichenkette 1,2,3,4,5,6,7,8,9 ausgesendet. Der Abstand zur Basisstation betrug im ersten Test 3,30 m und wurde nach jeder Übertragung um 30 cm verringert. In einem zweiten Test wurde ebenfalls mit einem Abstand von 3,30 m gestartet. Nach jeder Übertragung wurde der Sender 30 cm von der Startposition aus, entlang einer Linie, rechtwinklig zur Sichtverbindung Startpunkt-Basis, vom Relax Kit entfernt. Die gemessenen Werte wurden zur weiteren Auswertung abgespeichert. In beiden Test war die Basisstation so ausgerichtet, das TDA1 in Richtung der gemeinsamen Startposition der Tests zeigte. Logic Analyser und Debugger waren während der Tests nicht an der Basisstation angeschlossen. So sollten Abschattungseffekte durch diese verhindert werden. Die Basisstation wurde auf einem 70 cm hohen Tisch aufgestellt. Der Sender wurde auf gleicher Höhe freischwebend bewegt. Es wurden in beiden Tests zehn Messpunkte gesendet. Sowohl die Antennen an der Basisstation als auch jene am Relax Kit waren senkrecht nach oben zeigend ausgerichtet.

4.3 Ergebnisse und Auswertung

Auffallend ist, dass zwar in jedem Test zehn Mal durch das Drücken des Tasters eine Übertragung ausgelöst wurde, jedoch öfter eine Übertragung an der Basisstation registriert wurde. Im ersten Feldtest wurden fünfzehn, im zweiten sogar sechzehn gültige Übertragungen von der Basisstation an den Hostcomputer weitergegeben.

Im ersten Test konnten bei der ersten Übertragung an den Transceivern eins bis vier keine Daten empfangen werden. Erst in der darauffolgenden zweiten erkannten Übertragung wurde hier die gesendete Zahlenfolge empfangen. Bei der zweiten Übertragung, welche über die Konsole ausgegeben wurde, stimmten die Werte von TDA5 und TDA6 in allen ausgelesenen Registern mit den Messwerten der ersten Übertragung überein. Daraus ist zu folgern, dass es sich bei diesen Werten noch um die Messungen aus dem ersten Transfer handelt. Somit wäre zu vermuten, dass durch eine leichte Verzögerung zwischen den TDA5340 der steuernde Mikrocontroller eine gemeinsame Übermittlung als zwei getrennte Übertragungen interpretiert hat.

Transceiver 1 konnte in dem Test erst ab der vierten Übertragung gültige Daten empfangen. Außerdem waren die gemessenen Empfangsleistungen stets geringer als -100 dBm, lediglich bei der letzten Übertragung, welche bei einem Abstand von 30 cm stattfand, konnte hier ein Wert von -97 dBm gemessen werden. Da diese sehr schwachen Empfangsleistungen in vorherigen Tests nicht auftraten, ist zu vermuten, dass etwa eine nicht richtig verbundene Antenne Grund des schwachen Empfangswertes war.

TDA3 konnte in keiner einzigen Übertragung passable Messwerte liefern. Die errechnete und ausgegebene Empfangsfeldstärke von -114 dBm entsprach dem minimal möglichen Ausgabewert der dafür verwendeten Funktion. Es ist also davon auszugehen, dass in diesem Transceiver nie eine Funkverbindung erkannt wurde. Gründe dafür wären ein Fehler im Anpassnetzwerk zwischen der Antenne und dem Transceiver oder ein Defekt des selbigen. Letzteres ist eher unwahrscheinlich, da eine Verbindung über SPI mit dem IC möglich war. Lediglich ein Teildefekt in der RF-Sektion des Chips wäre also denkbar. Daneben waren vereinzelt auch noch Übertragungen zu erkennen, in denen Registerwerte mit den Messungen aus den folgenden oder vorherigen Übertragungen übereinstimmten. Auch hier ist zu vermuten, dass einzelne Transceiver eine Übertragung nicht erkennen konnten.

Zu beachten ist, dass die vermeintlich doppelt ankommenden Übertragungen auch vom Sender ausgehen konnten. Es ist nicht komplett sicher festzustellen ob die mehrfache Ausgabe einer Übertragung durch die Basisstation bedingt ist oder ob vom Relax Kit mehr als die gezählten Übertragungen versendet wurden. Das Versenden der Nachricht wurde an diesem durch einen Tastendruck ausgelöst. Durch ein Prellen des Tasters konnten auch mehrfache, nur minimal verzögerte Übertragungen ausgelöst worden sein. Für die Durchführung von nachfolgenden Tests wäre demnach ein Sicherstellen einer nur einfachen Übertragung notwendig. Zusätzlich wäre die Ausgabe eines Zähler für die Übertragungen am Relax Kit und ein neuer zu sendender Datensatz für jede Übermittlung hilfreich.

Ein Abtrennen der Transceiver-Baugruppen und somit ein Vergrößern des Abstandes der Antennen war zur Verbesserung der Empfangsleistung nicht notwendig.

Zusammenfassung und Ausblick

Zusammenfassend kann man sagen, dass die Basisstation den gewünschten Zweck gut erfüllt. Dabei besteht trotzdem noch die Möglichkeit zur weiteren Anpassungen an den aktuellen Nutzen. So können alle Transceiver ordnungsgemäß angesteuert werden und die Erkennung der Interrupts durch den Mikrocontroller funktioniert. Trotz der zum jetzigen Zeitpunkt nicht umgesetzten Netzwerkverbindung der Basisstation konnte eine funktionierende und einfache Ausgabe der Werte umgesetzt werden, welche trotzdem eine frei anpassbare und übersichtliche Ausgabe erlaubt. Problematisch war hierbei die Verwendung eines abgekündigten und nicht mehr hergestellten ICs. Mittlerweile existieren jedoch Nachfolgemodelle des Ethernet-Controllers, sodass eine Migration möglich sein sollte, entsprechende Datenblätter mit Hinweisen stellt der Hersteller bereit.

Ausblickend ließen sich an der Basisstation noch weitere Verbesserungen durchführen. So wäre noch das Eruiere des Grundes für den schlechten Empfang am Transceiver 3 notwendig. Dafür könnte durch das Anschließen eines Signalgenerators an die Antennenbuchse der entsprechenden Transceiverbaugruppe ein möglicher Fehler im Anpassnetzwerk aufgezeigt werden. Sollte auch dies nicht zu einer Verbesserung führen, wäre ein Austausch des ICs notwendig.

Für das bessere Erkennen von doppelten Übertragungen wäre das Einfügen eines Zeitstempel in die Ausgabe hilfreich. Dadurch könnten Übertragungen die kurz hintereinander eintreffen, markiert und entsprechend zu einer korrekten zusammengefügt werden. Zu diesem Zweck würde es sich anbieten, die Realtime-Clock des XMC4500 zu verwenden. Dazu würde das auf der Platine vorsorglich verbaute Uhrenquarz verwendet werden. Zwar wäre auch durch das Abwarten auf weitere verzögerte Übertragungen von anderen Transceivern das Problem der auf mehrere Ausgaben verteilten Übertragungen vermeidbar. Dies würde jedoch zu einer Todzeit führen, in der keine andere Übertragung möglich ist, was zu vermeiden ist.

Auch wäre für eine Veränderung an der Platine die Auswahl eines anderen Eingangspins am XMC für das vom PP2 Pin des Transceiver 6 kommenden Interruptsignals sinnvoll. So wären die TDA3 und TDA6 nicht an den selben Interruptkanal des Mikrocontroller angeschlossen. Die dazu notwendigen Änderungen an der Software würden sich auf die Änderungen der entsprechenden Makros in der Headerdatei beschränken. Alternativ ließe

sich möglicherweise das Problem der konkurrierenden Interrupts über Anpassungen in der Software lösen. So wäre es möglich, das Interruptsignal einzelner TDA5340 nicht über den PP2 Pin auszugeben, sondern auch über die ebenfalls mit dem XMC verbundenen PP0 und PP1 Pins. Somit wäre ein Verteilen auf einzelne Kanäle der ERU wahrscheinlich möglich.

Abbildungsverzeichnis

2.1	Schaltplan der Transceiverbaugruppe	7
2.2	Layout des Transceivers mit der Sollbruchstelle, der Steckerleiste und dem Anpassnetzwerk. Die Lötflächen für die SMA-Buchse sind auf der rechten Seite erkennbar.	8
2.3	Der XMC4500 Mikrocontroller von Infineon im LQFP-Gehäuse mit 144 Pins Bauer2012New-Infineon-32	9
2.4	Schaltplan des XMC4500 Mikrocontrollers	11
2.5	Layout der USB-Buchse (links) auf der Basisstation mit der entsprechenden Schutzbeschaltung. In der Mitte des XMC sind die Thermal Vias zum Abführen der Wärme zu erkennen.	12
2.6	Layout für den Ethernetcontroller von Mircel und die Anbindung an den XMC4500 Mikrocontroller	13
2.7	Schaltplan des XMC4500 Mikrocontrollers	15
3.1	Blockdiagramm zur Peripherie des XMC4500 Mikrocontroller und die Anbindung an den Cortex-M4 Kern XMC-DataSheet	18
3.2	Struktur eines Digitalpins des XMC Mikrocontroller XMC-Reference	19
3.3	Start einer SPI-Übertragung zwischen dem Mikrocontroller und dem Transceiver	23
3.4	Programmablauf der Basisstation	24
3.5	Ausgabe des Mikrocontroller über den COM-Port in der Konsole bei einer beispielhaften Übertragung	29
6.1	3D-Modell der Basisstation in Altium Designer	49
6.2	Erstellte Box zum Schutz der Basisstation in der Software SolidWorks	50
6.3	Layout des Aufsteckboards mit dem TDA5340	51

Tabellenverzeichnis

3.1	Zuordnung SPI-Signale an die Pins des Mikrocontrollers.	20
3.2	SPI-Instruktionen des Transceivers und die entsprechenden auf der MOSI-Leitung übertragenen Datenwörter und Parameter. Die beiden Instruktionen zum transparenten Senden sind nicht enthalten.	22
3.3	PP2-Pins der Transceiver mit den entsprechenden Anschlüssen am XMC4500 und die Verbindung zur ERU	26
3.4	Verwendete Abkürzungen der übertragenen Messwerte	27
6.1	Seriennummern der im Projekt verwendeten TDA5340	49

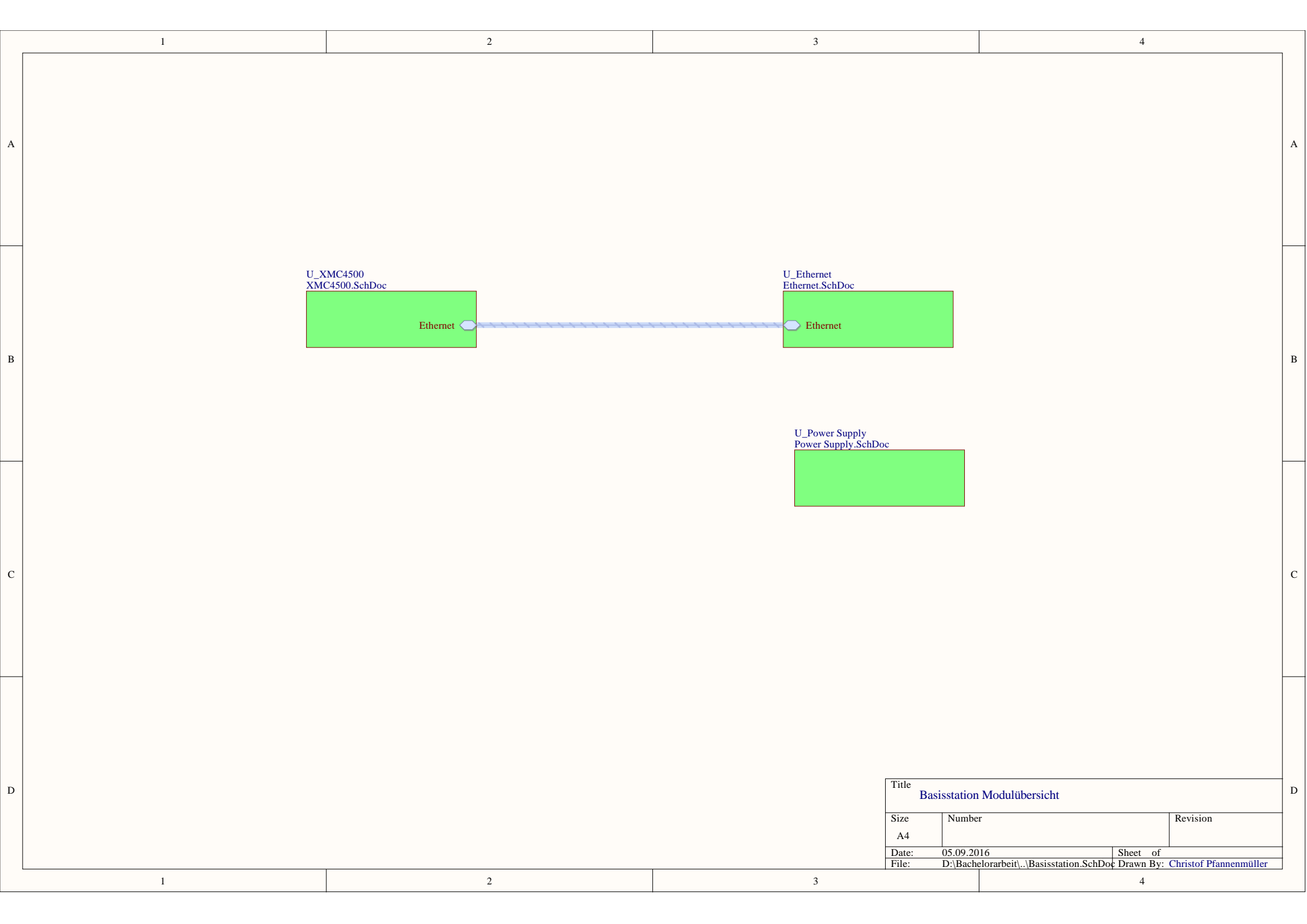
Quellcodeverzeichnis

3.1	Ausschnitt aus dem Senden der Daten über den COM-Port	27
6.1	Hauptdatei des Softwareentwurfs	51
6.2	Interrupt Service Routinen des Softwareentwurfs	59
6.3	Initialisierung des Softwareentwurfs	60

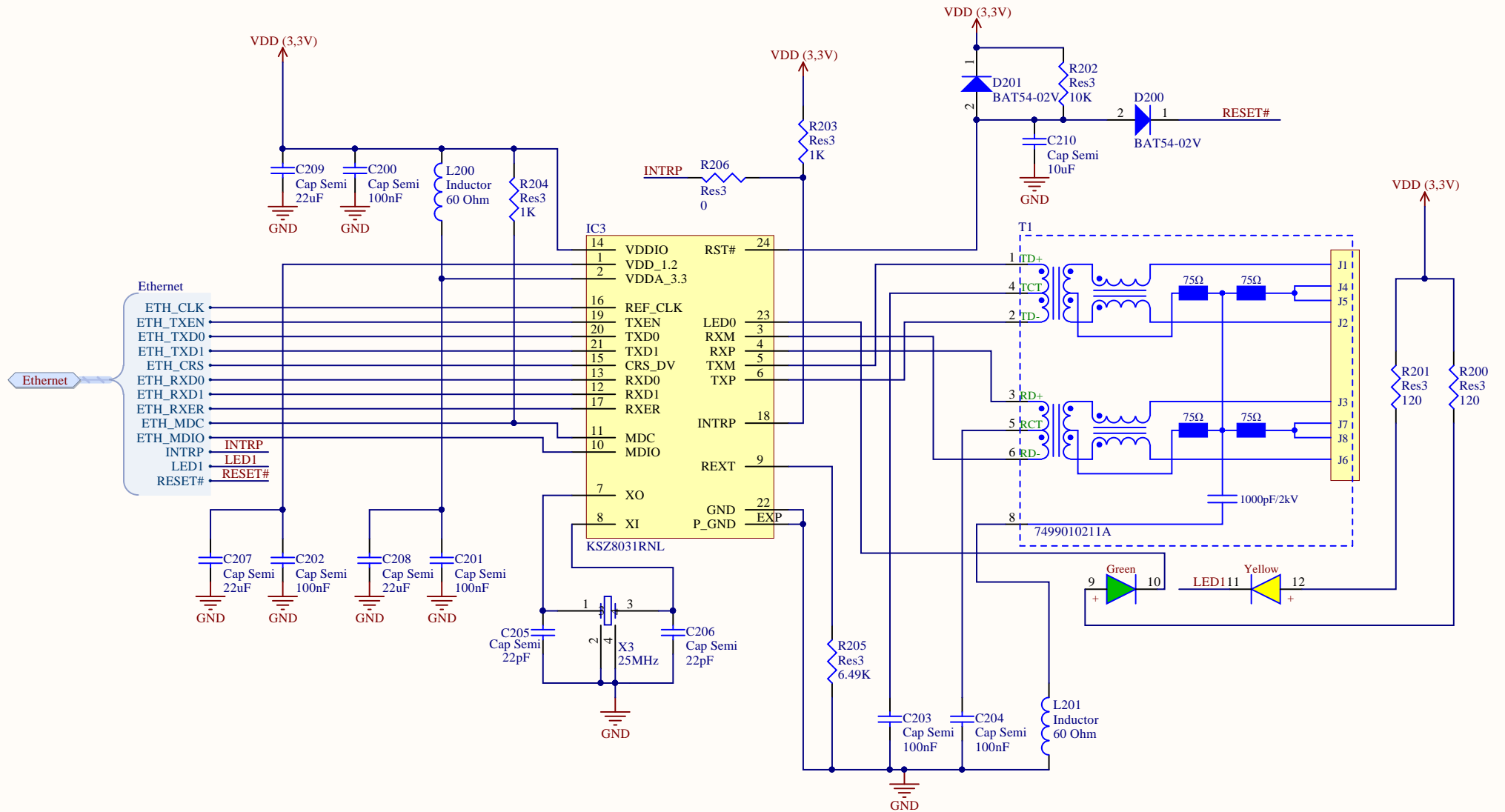
Anhang

6.1 Schaltpläne

Auf den folgenden Seiten sind die in Altium Designer erstellten Schaltpläne für alle Baugruppen der Basisstation dargestellt. Außerdem ist das PCB-Layout der Basisstation abgebildet.



Title			Basisstation Modulübersicht	
Size	Number		Revision	
A4				
Date:	05.09.2016		Sheet	of
File:	D:\Bachelorarbeit\...\Basisstation.SchDoc		Drawn By: Christof Pfannenmüller	



Title		
Ethernet		
Size	Number	Revision
A4		
Date:	05.09.2016	Sheet of
File:	D:\Bachelorarbeit\...\Ethernet.SchDoc	Drawn By: Christof Pfannenmüller

A

B

C

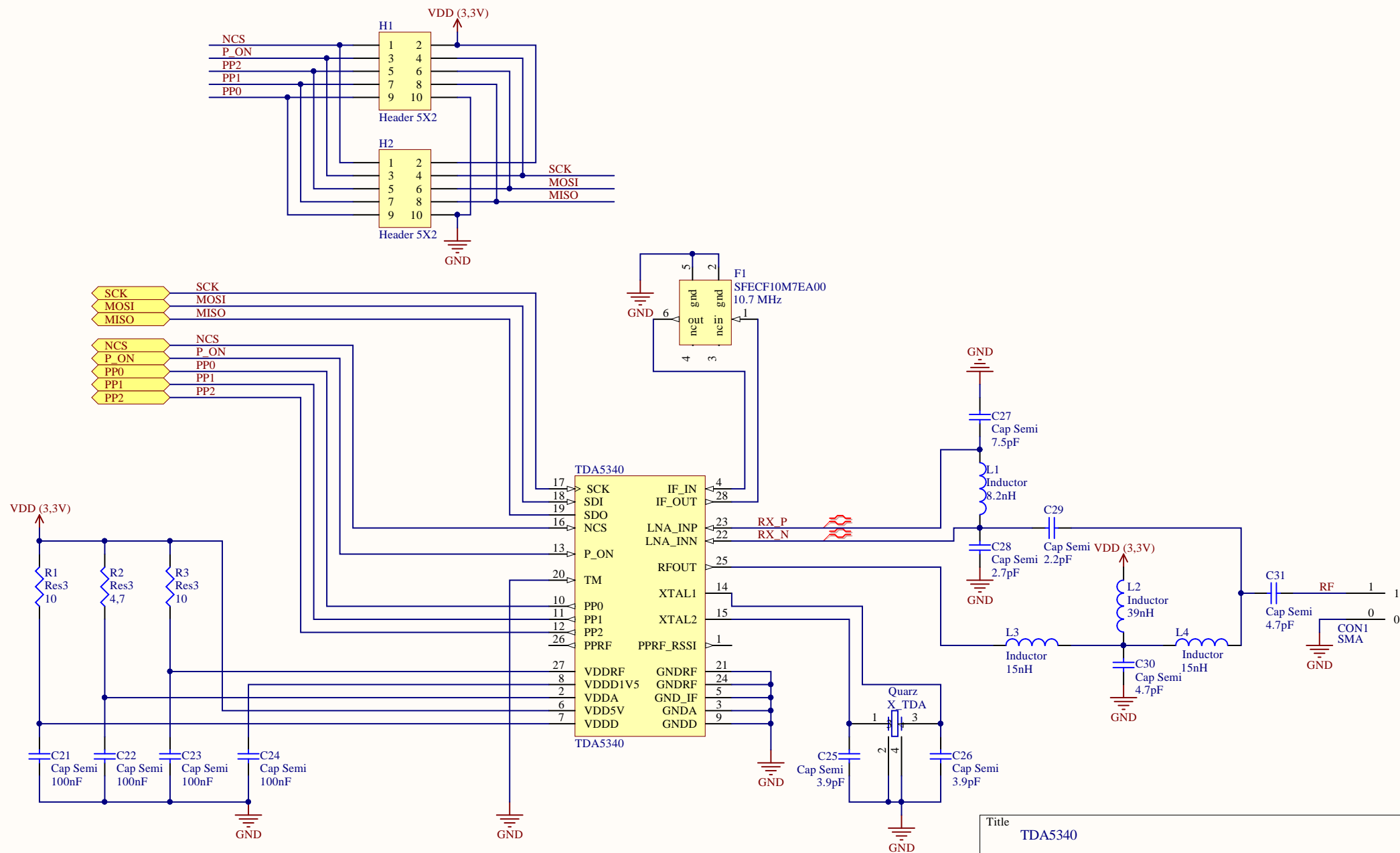
D

A

B

C

D



Title		
TDA5340		
Size	Number	Revision
A4		
Date:	05.09.2016	Sheet of
File:	D:\Bachelorarbeit\...\TDA5340.SchDoc	Drawn By: Christof Pfannenmüller

A

B

C

D

1

2

3

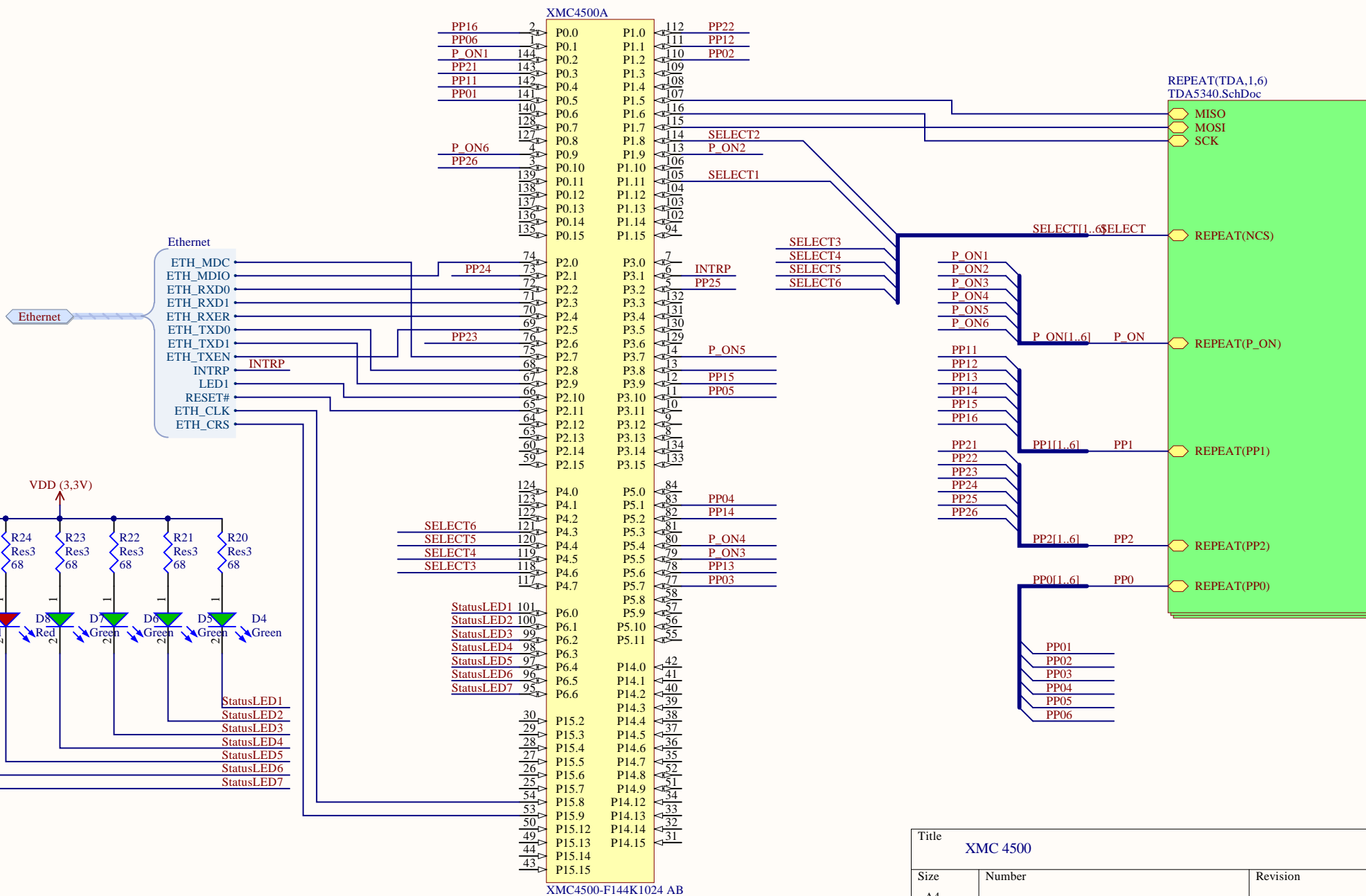
4

A

B

C

D



Title		
XMC 4500		
Size	Number	Revision
A4		
Date:	05.09.2016	Sheet of
File:	D:\Bachelorarbeit\...\XMC4500.SchDoc	Drawn By: Christof Pfannenmüller

1

2

3

4

1

2

3

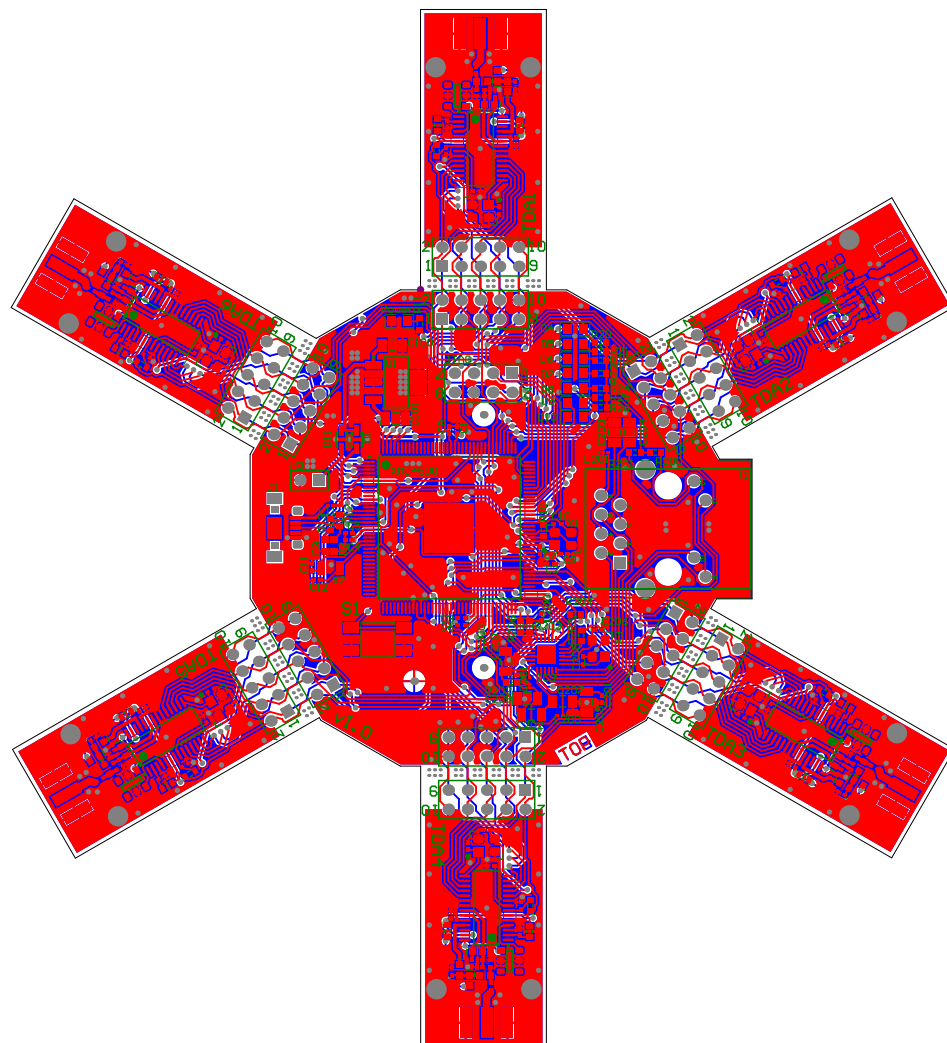
4

A

B

C

D



Altium Limited
12a Rodborough Rd
Frenchs Forest
NSW 2086

ENGINEER: Christof Pfannenmueller		TITLE: Basistation PCB Layout	
PCB DESIGNER:			
DATE: 05.09.2016	PART NO.:		REV:
FILE NAME: PCB_Testplatine.PcbDoc	DWG NO.:		SCALE:

1

2

3

4

6.2 Seriennummern

Alle TDA5340 verfügen über eine eingebaute Seriennummer, welche ausgelesen werden kann. Die Seriennummern der verwendeten TDA5340 sind in der Tabelle aufgeführt.

TDA	Seriennummer
TDA1	33020236
TDA2	11727080
TDA3	11545236
TDA4	11728870
TDA5	11550773
TDA6	33026263

Tab. 6.1: Seriennummern der im Projekt verwendeten TDA5340

6.3 3D-Daten

6.3.1 Platine

Die Platine wurde zur Validierung des Designs als 3D-Modell dargestellt.

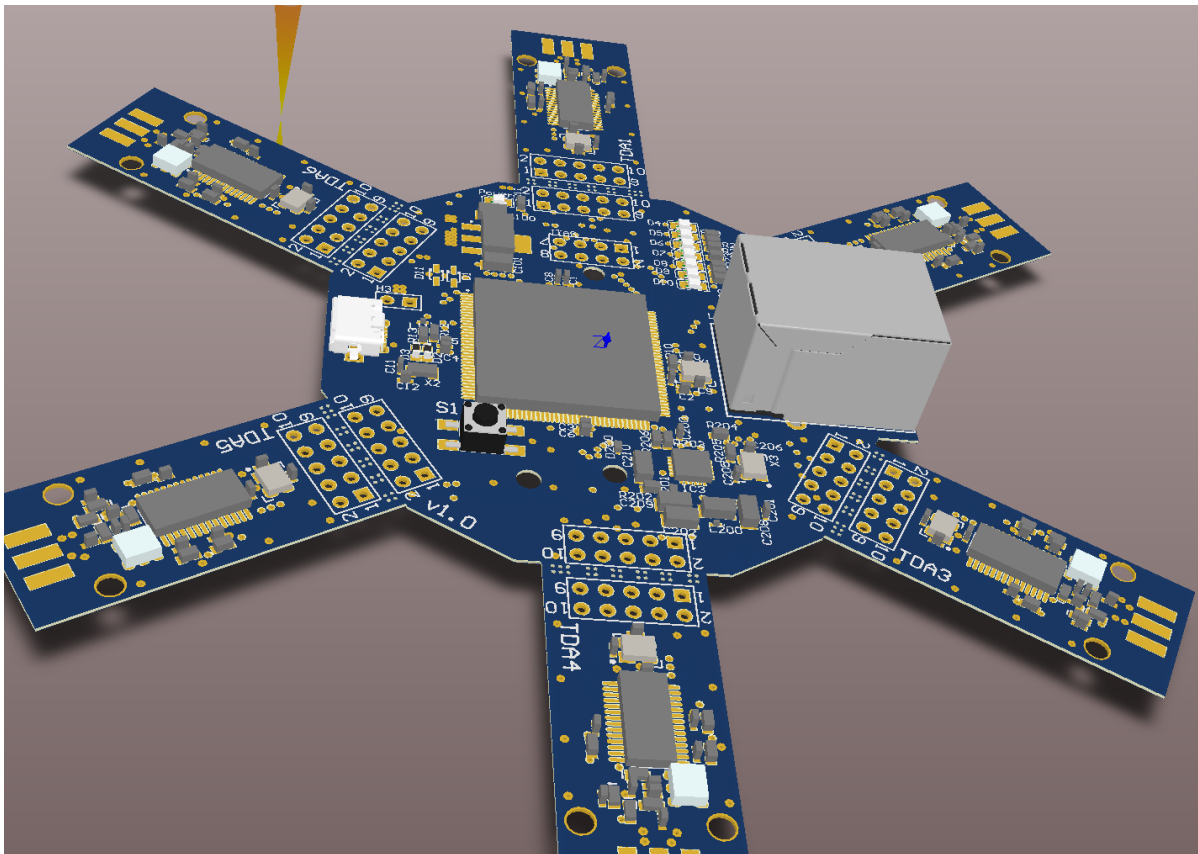


Abb. 6.1: 3D-Modell der Basisstation in Altium Designer

6.3.2 Gehäuse

Mit der Software SolidWorks wurde ein Gehäuse für die Platine erstellt.

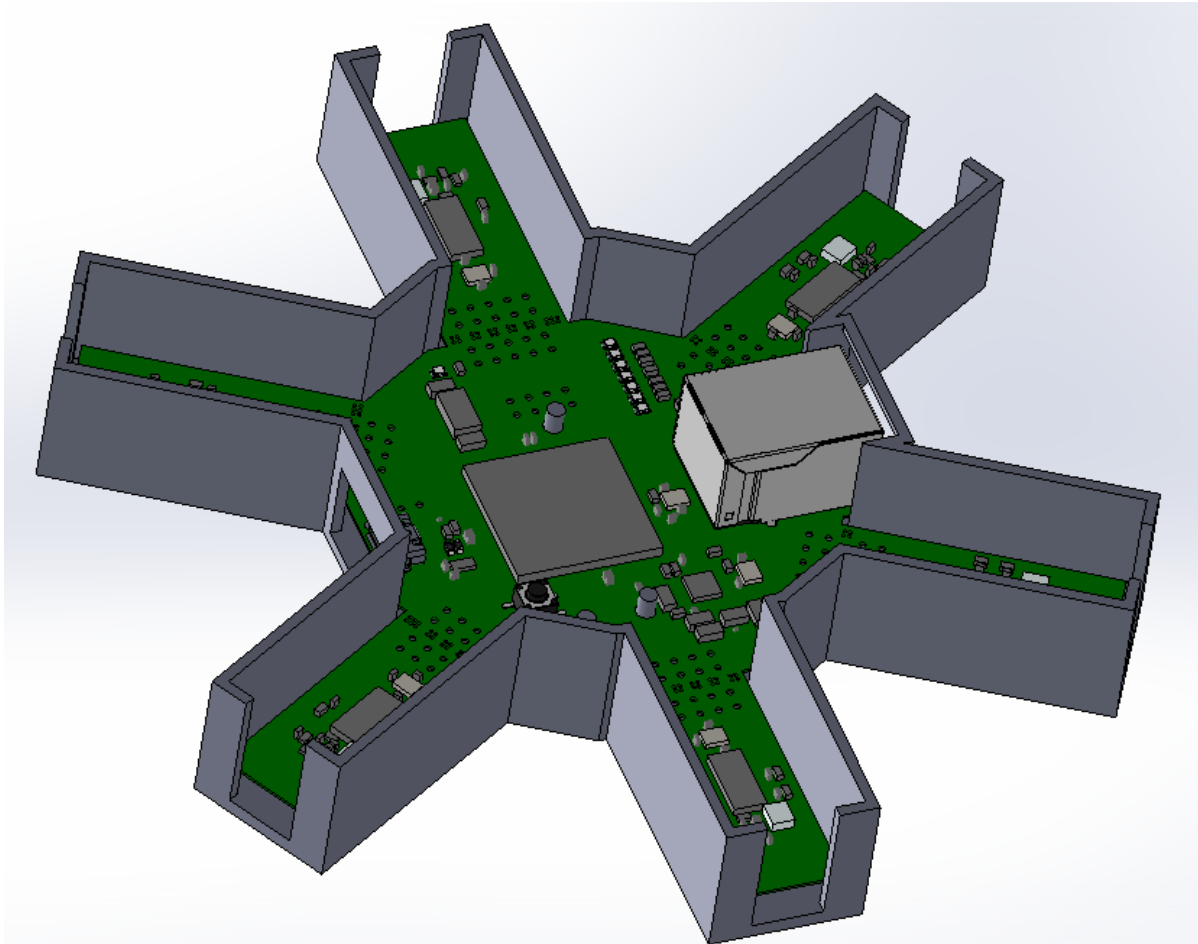


Abb. 6.2: Erstellte Box zum Schutz der Basisstation in der Software SolidWorks

6.4 Layout Aufsteckboard TDA5340

Das Layout der Transceiver-Unterbaugruppen orientiert sich an dem Layout eines Aufsteckboards für den „XMC 2Go“ damit ein TDA5340 auf mit diesem verwendet werden kann.

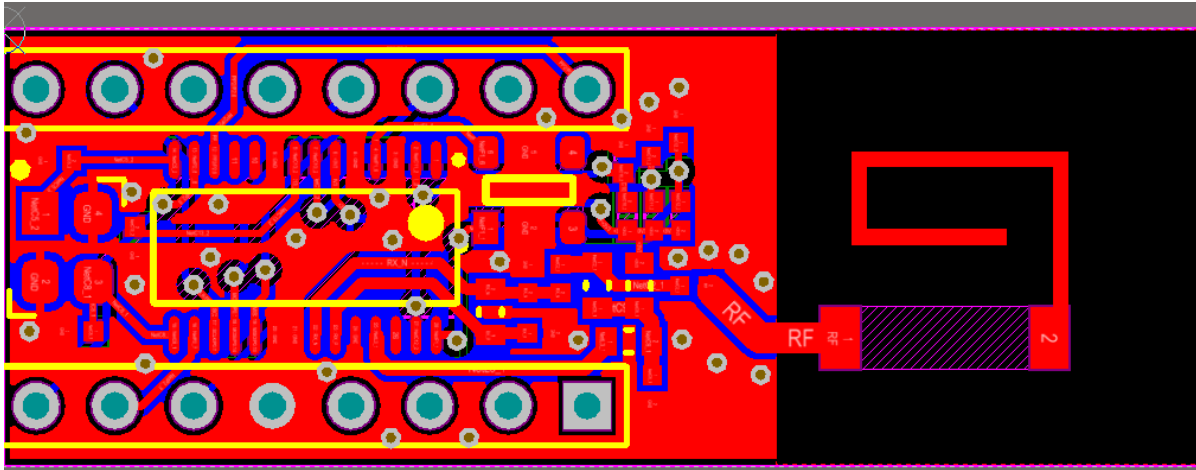


Abb. 6.3: Layout des Aufsteckboards mit dem TDA5340

6.5 Quellcode

6.5.1 Main.c

Quellcode 6.1: Hauptdatei des Softwareentwurfs

```

1  /*
2   * Main.c
3   *
4   * Created on: Jun 13, 2016
5   * Author: Christof Pfannenmüller
6   */
7  #include "Header_general.h" //including all Header files

9  // Global variables
10 uint8_t query_interruptTDA1_flag = 0;
11 uint8_t query_interruptTDA2_flag = 0;
12 uint8_t query_interruptTDA3_flag = 0;
13 uint8_t query_interruptTDA4_flag = 0;
14 uint8_t query_interruptTDA5_flag = 0;
15 uint8_t query_interruptTDA6_flag = 0;

17 int16_t dig_to_dbm(uint8_t dig, uint8_t agc) {
18     int32_t dbm_val = (712L * dig - 231628L + 3289L * agc) / 2048UL;
19     return (int16_t) dbm_val;
20 }

22 int main(void) {

24     init();
25     USB_Init(); //for virt. COM Port
26     COM_wait_for_transfer();

28     set_TDA_status(TDA_ALL, 1);
29     delay(40000);
30     tda5340_gpio_init(TDA_ALL);
31     spi_init(spi_master_ch);

33     delay(500);
34     delay(500);

36     led_on(LED_ALL);
37     delay(4000000);

```

```

38     led_off(LED_ALL);
39     led_on(LED1);
40     delay(4000000);
41     led_off(LED_ALL);
42     led_on(LED1);

44     // set_TDA_status(TDA1, 1);
45     // delay(4000);
46     // set_TDA_status(TDA2, 1);
47     // delay(4000);
48     // set_TDA_status(TDA3, 1);
49     // delay(4000000);
50     // set_TDA_status(TDA4, 1);
51     // delay(4000);
52     // set_TDA_status(TDA5, 1);
53     // delay(4000000);
54     // set_TDA_status(TDA6, 1);
55     // delay(4000);

57     delay(40000);
58     tda5340_init(TDA1); //Verzoegerung nach set Status muss gros genug sein bis SPI Kom
        moeglich ist, delay(45000); müsste das richtige sein
59     tda5340_set_mode_and_config(TDA1, RX_MODE, 0);

61     delay(40000);

63     //für gesamte Platine:
64     tda5340_init(TDA2);
65     tda5340_set_mode_and_config(TDA2, RX_MODE, 0);
66     tda5340_init(TDA3);
67     tda5340_set_mode_and_config(TDA3, RX_MODE, 0);
68     tda5340_init(TDA4);
69     tda5340_set_mode_and_config(TDA4, RX_MODE, 0);
70     tda5340_init(TDA5);
71     tda5340_set_mode_and_config(TDA5, RX_MODE, 0);
72     tda5340_init(TDA6);
73     tda5340_set_mode_and_config(TDA6, RX_MODE, 0);

75     // Ablaufschleife START
        ++++++

76     COM_send_string("Initialisierung beendet - ");
77     uint8_t data_recieved = 0;
78     uint32_t istateTDA1 = 0, istateTDA2 = 0, istateTDA3 = 0, istateTDA4 = 0, istateTDA5
        = 0, istateTDA6 = 0;
79     uint8_t lengthTDA1 = 0, lengthTDA2 = 0, lengthTDA3 = 0, lengthTDA4 = 0, lengthTDA5
        = 0, lengthTDA6 = 0;
80     uint32_t transfernumber = 0;
81     uint32_t led_ctr = 0;
82     char rx_data_TDA1[36] = { 1, 0, 3, 0, 5, 6, 7, 8, 9 };//TODO: Test ob er auch
        wirklich gesetzt wird; aktuell sind vor und nach dem empfangen inhalt der
        variable gleich
83     char rx_data_TDA2[36] = { 1, 0, 3, 0, 5, 6, 7, 8, 9 };
84     char rx_data_TDA3[36] = { 1, 0, 3, 0, 5, 6, 7, 8, 9 };
85     char rx_data_TDA4[36] = { 1, 0, 3, 0, 5, 6, 7, 8, 9 };
86     char rx_data_TDA5[36] = { 1, 0, 3, 0, 5, 6, 7, 8, 9 };
87     char rx_data_TDA6[36] = { 1, 0, 3, 0, 5, 6, 7, 8, 9 };
88     struct rssi {
89         uint8_t pmf;
90         uint8_t prx;
91         uint8_t rx;
92         uint8_t ppl;
93         uint8_t agc;
94     };

96     struct rssi rssiTDA1 = { .pmf = 0, .prx = 0, .rx = 0, .ppl = 0, .agc = 0 };
97     struct rssi rssiTDA2 = { .pmf = 0, .prx = 0, .rx = 0, .ppl = 0, .agc = 0 };
98     struct rssi rssiTDA3 = { .pmf = 0, .prx = 0, .rx = 0, .ppl = 0, .agc = 0 };
99     struct rssi rssiTDA4 = { .pmf = 0, .prx = 0, .rx = 0, .ppl = 0, .agc = 0 };

```



```

100     struct rssi rssiTDA5 = { .pmf = 0, .prx = 0, .rx = 0, .ppl = 0, .agc = 0 };
101     struct rssi rssiTDA6 = { .pmf = 0, .prx = 0, .rx = 0, .ppl = 0, .agc = 0 };

103     query_interruptTDA1_flag = 0;
104     query_interruptTDA2_flag = 0;
105     query_interruptTDA3_flag = 0;
106     query_interruptTDA4_flag = 0;
107     query_interruptTDA5_flag = 0;
108     query_interruptTDA6_flag = 0;

110 //-----
111     COM_send_string("Warte auf Übertragungen\r\n");
112     while (1) {
113         if (query_interruptTDA1_flag) {
114             query_interruptTDA1_flag = 0;
115             istateTDA1 = tda5340_interrupt_readout(TDA1);
116         }
117         if (query_interruptTDA2_flag) {
118             query_interruptTDA2_flag = 0;
119             istateTDA2 = tda5340_interrupt_readout(TDA2);
120         }
121         if (query_interruptTDA3_flag) {
122             query_interruptTDA3_flag = 0;
123             istateTDA3 = tda5340_interrupt_readout(TDA3);
124         }
125         if (query_interruptTDA4_flag) {
126             query_interruptTDA4_flag = 0;
127             istateTDA4 = tda5340_interrupt_readout(TDA4);
128         }
129         if (query_interruptTDA5_flag) {
130             query_interruptTDA5_flag = 0;
131             istateTDA5 = tda5340_interrupt_readout(TDA5);
132         }
133         if (query_interruptTDA6_flag) {
134             query_interruptTDA6_flag = 0;
135             led_on(LED3);
136             istateTDA6 = tda5340_interrupt_readout(TDA6);
137         }
138 //-----

140         if (istateTDA1 & (1 << 1)) {
141             rssiTDA1.pmf = tda5340_transfer(TDA1, READ_FROM_CHIP, RSSIPMF, 0xFF);
142             rssiTDA1.rx = tda5340_transfer(TDA1, READ_FROM_CHIP, RSSIRX, 0xFF);
143             rssiTDA1.agc = (tda5340_transfer(TDA1, READ_FROM_CHIP, AGCADRR, 0xFF) & 0x06)
144                 >> 1;
145             istateTDA1 &= ~(1 << 1);
146         }
147         if (istateTDA2 & (1 << 1)) {
148             rssiTDA2.pmf = tda5340_transfer(TDA2, READ_FROM_CHIP, RSSIPMF, 0xFF);
149             rssiTDA2.rx = tda5340_transfer(TDA2, READ_FROM_CHIP, RSSIRX, 0xFF);
150             rssiTDA2.agc = (tda5340_transfer(TDA2, READ_FROM_CHIP, AGCADRR, 0xFF) & 0x06)
151                 >> 1;
152             istateTDA2 &= ~(1 << 1);
153         }
154         if (istateTDA3 & (1 << 1)) {
155             rssiTDA3.pmf = tda5340_transfer(TDA3, READ_FROM_CHIP, RSSIPMF, 0xFF);
156             rssiTDA3.rx = tda5340_transfer(TDA3, READ_FROM_CHIP, RSSIRX, 0xFF);
157             rssiTDA3.agc = (tda5340_transfer(TDA3, READ_FROM_CHIP, AGCADRR, 0xFF) & 0x06)
158                 >> 1;
159             istateTDA3 &= ~(1 << 1);
160         }
161         if (istateTDA4 & (1 << 1)) {
162             rssiTDA4.pmf = tda5340_transfer(TDA4, READ_FROM_CHIP, RSSIPMF, 0xFF);
163             rssiTDA4.rx = tda5340_transfer(TDA4, READ_FROM_CHIP, RSSIRX, 0xFF);
164             rssiTDA4.agc = (tda5340_transfer(TDA4, READ_FROM_CHIP, AGCADRR, 0xFF) & 0x06)
165                 >> 1;
166             istateTDA4 &= ~(1 << 1);
167         }
168         if (istateTDA5 & (1 << 1)) {

```

```

165     rssiTDA5.pmf = tda5340_transfer(TDA5, READ_FROM_CHIP, RSSIPMF, 0xFF);
166     rssiTDA5.rx = tda5340_transfer(TDA5, READ_FROM_CHIP, RSSIRX, 0xFF);
167     rssiTDA5.agc = (tda5340_transfer(TDA5, READ_FROM_CHIP, AGCADRR, 0xFF) & 0x06)
        >> 1;
168     istateTDA5 &= ~(1 << 1);
169 }
170 if (istateTDA6 & (1 << 1)) {
171     rssiTDA6.pmf = tda5340_transfer(TDA6, READ_FROM_CHIP, RSSIPMF, 0xFF);
172     rssiTDA6.rx = tda5340_transfer(TDA6, READ_FROM_CHIP, RSSIRX, 0xFF);
173     rssiTDA6.agc = (tda5340_transfer(TDA6, READ_FROM_CHIP, AGCADRR, 0xFF) & 0x06)
        >> 1;
174     istateTDA6 &= ~(1 << 1);
175 }
176 //-----
177 if (istateTDA1 & (1 << 3)) {

179     rssiTDA1.prx = tda5340_transfer(TDA1, READ_FROM_CHIP, RSSIPRX, 0xFF);
180     rssiTDA1.ppl = tda5340_transfer(TDA1, READ_FROM_CHIP, RSSIPPL, 0xFF);
181     tda5340_set_mode_and_config(TDA1, SLEEP_MODE, 0);
182     if (!tda5340_receive(TDA1, rx_data_TDA1, &lengthTDA1)) {
183         if (lengthTDA1 > 32)
184             lengthTDA1 = 32;
185     }
186     tda5340_set_mode_and_config(TDA1, RX_MODE, 0);
187     data_recieved = 1;
188     istateTDA1 &= ~(1 << 3);
189 }
190 if (istateTDA2 & (1 << 3)) {

192     rssiTDA2.prx = tda5340_transfer(TDA2, READ_FROM_CHIP, RSSIPRX, 0xFF);
193     rssiTDA2.ppl = tda5340_transfer(TDA2, READ_FROM_CHIP, RSSIPPL, 0xFF);
194     tda5340_set_mode_and_config(TDA2, SLEEP_MODE, 0);
195     if (!tda5340_receive(TDA2, rx_data_TDA2, &lengthTDA2)) {
196         if (lengthTDA2 > 32)
197             lengthTDA2 = 32;
198     }
199     tda5340_set_mode_and_config(TDA2, RX_MODE, 0);
200     data_recieved = 1;
201     istateTDA2 &= ~(1 << 3);
202 }
203 if (istateTDA3 & (1 << 3)) {

205     rssiTDA3.prx = tda5340_transfer(TDA3, READ_FROM_CHIP, RSSIPRX, 0xFF);
206     rssiTDA3.ppl = tda5340_transfer(TDA3, READ_FROM_CHIP, RSSIPPL, 0xFF);
207     tda5340_set_mode_and_config(TDA3, SLEEP_MODE, 0);
208     if (!tda5340_receive(TDA3, rx_data_TDA3, &lengthTDA3)) {
209         if (lengthTDA3 > 32)
210             lengthTDA3 = 32;
211     }
212     tda5340_set_mode_and_config(TDA3, RX_MODE, 0);
213     data_recieved = 1;
214     istateTDA3 &= ~(1 << 3);
215 }
216 if (istateTDA4 & (1 << 3)) {

218     rssiTDA4.prx = tda5340_transfer(TDA4, READ_FROM_CHIP, RSSIPRX, 0xFF);
219     rssiTDA4.ppl = tda5340_transfer(TDA4, READ_FROM_CHIP, RSSIPPL, 0xFF);
220     tda5340_set_mode_and_config(TDA4, SLEEP_MODE, 0);
221     if (!tda5340_receive(TDA4, rx_data_TDA4, &lengthTDA4)) {
222         if (lengthTDA4 > 32)
223             lengthTDA4 = 32;
224     }
225     tda5340_set_mode_and_config(TDA4, RX_MODE, 0);
226     data_recieved = 1;
227     istateTDA4 &= ~(1 << 3);
228 }
229 if (istateTDA5 & (1 << 3)) {

231     rssiTDA5.prx = tda5340_transfer(TDA5, READ_FROM_CHIP, RSSIPRX, 0xFF);

```

```

232     rssiTDA5.ppl = tda5340_transfer(TDA5, READ_FROM_CHIP, RSSIPPL, 0xFF);
233     tda5340_set_mode_and_config(TDA5, SLEEP_MODE, 0);
234     if (!tda5340_receive(TDA5, rx_data_TDA5, &lengthTDA5)) {
235         if (lengthTDA5 > 32)
236             lengthTDA5 = 32;
237     }
238     tda5340_set_mode_and_config(TDA5, RX_MODE, 0);
239     data_recieved = 1;
240     istateTDA5 &= ~(1 << 3);
241 }
242 if (istateTDA6 & (1 << 3)) {

244     rssiTDA6.prx = tda5340_transfer(TDA6, READ_FROM_CHIP, RSSIPRX, 0xFF);
245     rssiTDA6.ppl = tda5340_transfer(TDA6, READ_FROM_CHIP, RSSIPPL, 0xFF);
246     tda5340_set_mode_and_config(TDA6, SLEEP_MODE, 0);
247     if (!tda5340_receive(TDA6, rx_data_TDA6, &lengthTDA6)) {
248         if (lengthTDA6 > 32)
249             lengthTDA6 = 32;
250     }
251     tda5340_set_mode_and_config(TDA6, RX_MODE, 0);
252     data_recieved = 1;
253     istateTDA6 &= ~(1 << 3);
254 }
255 //
-----

256 //send to COM
257 if (data_recieved) {
258     transfernumber++;
259     COM_send_string("##### Übertragung erkannt #####\r\n");
260     COM_send_string("Übertragung Nummer ");
261     COM_send_int_as_string(transfernumber);
262     COM_send_string("\r\n\r\n");
263     COM_send_string("TDA1:");
264     COM_send_string("\r\nPMF:");
265     COM_send_int_as_string(rssiTDA1.pmf);
266     COM_send_string("\r\nPRX:");
267     COM_send_int_as_string(rssiTDA1.prx);
268     COM_send_string("\r\nRX:");
269     COM_send_int_as_string(rssiTDA1.rx);
270     COM_send_string("\r\nPPL:");
271     COM_send_int_as_string(rssiTDA1.ppl);
272     COM_send_string("\r\nAGC:");
273     COM_send_int_as_string(rssiTDA1.agc);
274     COM_send_string("\r\nEmpfangsleistung (dBm):");
275     if (dig_to_dbm(rssiTDA1.ppl, rssiTDA1.agc) < 0) {
276         COM_send_string("-");
277     }
278     COM_send_int_as_string(abs(dig_to_dbm(rssiTDA1.ppl, rssiTDA1.agc)));
279     COM_send_string("\r\n");
280     COM_send_string("Empfangene Daten:");
281     for (int i = 0; i <= lengthTDA1; ++i) {
282         COM_send_int_as_string(rx_data_TDA1[i]);

284     }
285     COM_send_string("\r\n\r\n");

287     COM_send_string("TDA2:");
288     COM_send_string("\r\nPMF:");
289     COM_send_int_as_string(rssiTDA2.pmf);
290     COM_send_string("\r\nPRX:");
291     COM_send_int_as_string(rssiTDA2.prx);
292     COM_send_string("\r\nRX:");
293     COM_send_int_as_string(rssiTDA2.rx);
294     COM_send_string("\r\nPPL:");
295     COM_send_int_as_string(rssiTDA2.ppl);
296     COM_send_string("\r\nAGC:");
297     COM_send_int_as_string(rssiTDA2.agc);

```

```

298     COM_send_string("\r\nEmpfangsleistung (dBm):");
299     if (dig_to_dbm(rssiTDA2.ppl, rssiTDA2.agc) < 0) {
300         COM_send_string("-");
301     }
302     COM_send_int_as_string(abs(dig_to_dbm(rssiTDA2.ppl, rssiTDA2.agc)));
303     COM_send_string("\r\n");
304     COM_send_string("Empfangene Daten:");
305     for (int i = 0; i <= lengthTDA2; ++i) {
306         COM_send_int_as_string(rx_data_TDA2[i]);
307     }
308
309     COM_send_string("\r\n\r\n");
310
311     COM_send_string("TDA3:");
312     COM_send_string("\r\nPMF:");
313     COM_send_int_as_string(rssiTDA3.pmf);
314     COM_send_string("\r\nPRX:");
315     COM_send_int_as_string(rssiTDA3.prx);
316     COM_send_string("\r\nRX:");
317     COM_send_int_as_string(rssiTDA3.rx);
318     COM_send_string("\r\nPPL:");
319     COM_send_int_as_string(rssiTDA3.ppl);
320     COM_send_string("\r\nAGC:");
321     COM_send_int_as_string(rssiTDA3.agc);
322     COM_send_string("\r\nEmpfangsleistung (dBm):");
323     if (dig_to_dbm(rssiTDA3.ppl, rssiTDA3.agc) < 0) {
324         COM_send_string("-");
325     }
326     COM_send_int_as_string(abs(dig_to_dbm(rssiTDA3.ppl, rssiTDA3.agc)));
327     COM_send_string("\r\n");
328     COM_send_string("Empfangene Daten:");
329     for (int i = 0; i <= lengthTDA3; ++i) {
330         COM_send_int_as_string(rx_data_TDA3[i]);
331     }
332
333     COM_send_string("\r\n\r\n");
334
335     COM_send_string("TDA4:");
336     COM_send_string("\r\nPMF:");
337     COM_send_int_as_string(rssiTDA4.pmf);
338     COM_send_string("\r\nPRX:");
339     COM_send_int_as_string(rssiTDA4.prx);
340     COM_send_string("\r\nRX:");
341     COM_send_int_as_string(rssiTDA4.rx);
342     COM_send_string("\r\nPPL:");
343     COM_send_int_as_string(rssiTDA4.ppl);
344     COM_send_string("\r\nAGC:");
345     COM_send_int_as_string(rssiTDA4.agc);
346     COM_send_string("\r\nEmpfangsleistung (dBm):");
347     if (dig_to_dbm(rssiTDA4.ppl, rssiTDA4.agc) < 0) {
348         COM_send_string("-");
349     }
350     COM_send_int_as_string(abs(dig_to_dbm(rssiTDA4.ppl, rssiTDA4.agc)));
351     COM_send_string("\r\n");
352     COM_send_string("Empfangene Daten:");
353     for (int i = 0; i <= lengthTDA4; ++i) {
354         COM_send_int_as_string(rx_data_TDA4[i]);
355     }
356
357     COM_send_string("\r\n\r\n");
358
359     COM_send_string("TDA5:");
360     COM_send_string("\r\nPMF:");
361     COM_send_int_as_string(rssiTDA5.pmf);
362     COM_send_string("\r\nPRX:");
363     COM_send_int_as_string(rssiTDA5.prx);
364     COM_send_string("\r\nRX:");
365     COM_send_int_as_string(rssiTDA5.rx);
366     COM_send_string("\r\nPPL:");

```

```

367     COM_send_int_as_string(rssiTDA5.ppl);
368     COM_send_string("\r\nAGC:");
369     COM_send_int_as_string(rssiTDA5.agc);
370     COM_send_string("\r\nEmpfangsleistung (dBm):");
371     if (dig_to_dbm(rssiTDA5.ppl, rssiTDA5.agc) < 0) {
372         COM_send_string("-");
373     }
374     COM_send_int_as_string(abs(dig_to_dbm(rssiTDA5.ppl, rssiTDA5.agc)));
375     COM_send_string("\r\n");
376     COM_send_string("Empfangene Daten:");
377     for (int i = 0; i <= lengthTDA5; ++i) {
378         COM_send_int_as_string(rx_data_TDA5[i]);
379
380     }
381     COM_send_string("\r\n\r\n");
382
383     COM_send_string("TDA6:");
384     COM_send_string("\r\nPMF:");
385     COM_send_int_as_string(rssiTDA6.pmf);
386     COM_send_string("\r\nPRX:");
387     COM_send_int_as_string(rssiTDA6.prx);
388     COM_send_string("\r\nRX:");
389     COM_send_int_as_string(rssiTDA6.rx);
390     COM_send_string("\r\nPPL:");
391     COM_send_int_as_string(rssiTDA6.ppl);
392     COM_send_string("\r\nAGC:");
393     COM_send_int_as_string(rssiTDA6.agc);
394     COM_send_string("\r\nEmpfangsleistung (dBm):");
395     if (dig_to_dbm(rssiTDA6.ppl, rssiTDA6.agc) < 0) {
396         COM_send_string("-");
397     }
398     COM_send_int_as_string(abs(dig_to_dbm(rssiTDA6.ppl, rssiTDA6.agc)));
399     COM_send_string("\r\n");
400     COM_send_string("Empfangene Daten:");
401     for (int i = 0; i <= lengthTDA6; ++i) {
402         COM_send_int_as_string(rx_data_TDA6[i]);
403
404     }
405     COM_send_string("\r\n\r\n");
406
407     led_ctr = 400000;
408     led_on(LED7);
409     data_recieved = 0;
410 }
411
412 if (led_ctr) {
413     led_ctr--;
414
415     if (!led_ctr)
416         led_off(LED7);
417 }
418 }
419 // Ablaufschleife ENDE
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

432 //      uint8_t agc;
433 // } rssi = { .pmf = 0, .prx = 0, .rx = 0, .ppl = 0, .agc = 0 };
434 //
435 // uint32_t istate = 0, led1_ctr = 0, led2_ctr = 0;
436 //
437 // COM_send_string("Beginn while loop----- \r\n");
438 //
439 // // Main loop
440 // led_off(LED_ALL);
441 // query_interruptTDA6_flag = 0; //damit keine Auswirkungen von Interruots beim
// Einschalten
442 //
443 // while (1) {      //COM_send_string(".");
444 //
445 //     // NINT Interrupt handling
446 //
447 //     if (query_interruptTDA6_flag) {
448 //         query_interruptTDA6_flag = 0;
449 //         istate = tda5340_interrupt_readout(TDA6);
450 //         COM_send_string("Interrupt ist aufgetreten\r\n");
451 //         COM_send_int_as_string(istate);
452 //         led_on(LED5);
453 //     }
454 //
455 //     //
// -----
456 //
457 //     if(XMC_GPIO_GetInput(BUTTON1)) {
458 //         tda5340_transmit(tx_data, 8);
459 //     }
460 //
461 //     //
// -----
462 //
463 //     // Switch to Rx-Mode if Tx is finished
464 //     if (istate & (1 << 18)) {
465 //         tda5340_set_mode_and_config(TDA6, RX_MODE, 0); //TODO: was is
466 //         istate &= ~(1 << 18);
467 //         COM_send_string("Switch to Rx-Mode\r\n");
468 //     }
469 //
470 //     //
// -----
471 //
472 //     // Frame sync - Config A
473 //     if (istate & (1 << 1)) {
474 //         rssi.pmf = tda5340_transfer(TDA6, READ_FROM_CHIP, RSSIPMF, 0xFF);
475 //         rssi.rx = tda5340_transfer(TDA6, READ_FROM_CHIP, RSSIRX, 0xFF);
476 //         rssi.agc = (tda5340_transfer(TDA6, READ_FROM_CHIP, AGCADRR, 0xFF) & 0x06) >>
1;
477 //         istate &= ~(1 << 1);
478 //         COM_send_string("Frame sync - Config A\r\n");
479 //     }
480 //
481 //     // End of message - Config A
482 //     if (istate & (1 << 3)) {
483 //         // delay(5000);
484 //         rssi.prx = tda5340_transfer(TDA6, READ_FROM_CHIP, RSSIPRX, 0xFF);
485 //         rssi.ppl = tda5340_transfer(TDA6, READ_FROM_CHIP, RSSIPPL, 0xFF);
486 //
487 //         tda5340_set_mode_and_config(TDA6, SLEEP_MODE, 0);
488 //         COM_send_string("sleep-Mode\r\n");
489 //         if (!tda5340_receive(TDA6, rx_data, &length)) {
490 //             led_on(LED2);
491 //             COM_send_string("set led high\r\n");
492 //             if (length > 32)

```

```

493 //          length = 32;
494 //
495 //          led2_ctr = 400000;
496 //      }
497 //
498 //          tda5340_set_mode_and_config(TDA6, RX_MODE, 0);
499 //          istate &= ~(1 << 3);
500 //      }
501 //
502 //      //
-----
503 //
504 //      // LED-Timeout-Turnoff
505 //      if (led1_ctr) {
506 //          led1_ctr--;
507 //
508 //          if (!led1_ctr)
509 //              led_off(LED1);
510 //      }
511 //
512 //      if (led2_ctr) {
513 //          led2_ctr--;
514 //
515 //          if (!led2_ctr)
516 //              led_off(LED2);
517 //      }
518 //
519 //      //
-----
520 // }
521 //
522 // tda5340_set_mode_and_config(TDA6, RX_MODE, 0); // ANPASSEN AN MEJHRE TDAs
523 //
524 //TESTMODUL-ENDE
-----

526 //function for general test purposes
527 //general_test();
528 while (1) {

530 }

532 }

```

6.5.2 ISRs.c

Quellcode 6.2: Interrupt Service Routinen des Softwareentwurfs

```

1  /*
2  * ISRs.c
3  *
4  * Created on: Jul 7, 2016
5  * Author: student06
6  */

8  #include "Header_general.h" //including all Header files

10 // ISR für TDA1 (ERU1 OGUO IRQ)
11 extern void ERU1_0_IRQHandler(void) {
12     query_interruptTDA1_flag = 1;
13     COM_send_string("INTERRUPT1\r\n");
14 }
15 // ISR für TDA2 (ERU0 OGUO IRQ)
16 extern void ERU0_0_IRQHandler(void) {
17     query_interruptTDA2_flag = 1;

```

```

18 // COM_send_string("INTERRUPT2\r\n");
19 }
20 // ISR für TDA3 + TDA6 (ERU0 OGU1 IRQ)
21 extern void ERU0_1_IRQHandler(void) {
22     //XMC_ERU_ETL_ClearStatusFlag(XMC_ERU0, 1);
23     // COM_send_string("ISR 3 und 6 \r\n");

25     // //Check which Interrupt has occurred
26     // uint32_t status_tda3 = XMC_GPIO_GetInput(PORT_PP2_TDA_3, PIN_PP2_TDA_3);
27     // uint32_t status_tda6 = XMC_GPIO_GetInput(PORT_PP2_TDA_6, PIN_PP2_TDA_6);
28     // if ((!status_tda3) && (status_tda6)) {
29     //     COM_send_string("INTERRUPT3\r\n");
30     //     query_interruptTDA3_flag = 1;
31     // }
32     // if ((status_tda6 == 0) && (status_tda3 != 0)) {
33     //     COM_send_string("INTERRUPT6\r\n");
34     //     query_interruptTDA6_flag = 1;
35     // }
36     // if ((status_tda6 == 0) && (status_tda3 == 0)) {
37     //     COM_send_string("INTERRUPT 3&6 \r\n");
38     //     query_interruptTDA3_flag = 1;
39     //     query_interruptTDA6_flag = 1;
40     // }
41     query_interruptTDA3_flag = 1;
42     query_interruptTDA6_flag = 1; //beide setzen egal welcher ankommt -> es werden
        beide ausgelesen
43 // led_on(LED7);
44 }
45 // ISR für TDA4 (ERU1 OGU1 IRQ)
46 extern void ERU1_1_IRQHandler(void) {
47     query_interruptTDA4_flag = 1;
48     // COM_send_string("INTERRUPT4\r\n");
49 }
50 // ISR für TDA5 (ERU0 OGU2 IRQ)
51 extern void ERU0_2_IRQHandler(void) {
52     // COM_send_string("INTERRUPT5\r\n");
53     query_interruptTDA5_flag = 1;
54 }
55 // ISR für TDA6 (ERU0 OGU3 IRQ)
56 extern void ERU0_3_IRQHandler(void) {
57     led_on(LED5);
58     query_interruptTDA6_flag = 1;
59 }

```

6.5.3 Init.c

Quellcode 6.3: Initialisierung des Softwareentwurfs

```

1  /*
2   * Init.c
3   *
4   * Created on: Jun 16, 2016
5   * Author Christof Pfannenmüller (student06)
6   */
7  #include "Header_general.h" //including all Header files

9  //additional functions
10 void delay(unsigned long delay) {
11     while (delay--) {
12         __NOP();
13     }
14 }

16 //init

18 void init(void) {

20     //sets LED Pins as Outputs

```



```

21  XMC_GPIO_SetMode(PORT_LED_1, PIN_LED_1, XMC_GPIO_MODE_OUTPUT_PUSH_PULL); //LED1
22  XMC_GPIO_SetMode(PORT_LED_2, PIN_LED_2, XMC_GPIO_MODE_OUTPUT_PUSH_PULL);
23  XMC_GPIO_SetMode(PORT_LED_3, PIN_LED_3, XMC_GPIO_MODE_OUTPUT_PUSH_PULL);
24  XMC_GPIO_SetMode(PORT_LED_4, PIN_LED_4, XMC_GPIO_MODE_OUTPUT_PUSH_PULL);
25  XMC_GPIO_SetMode(PORT_LED_5, PIN_LED_5, XMC_GPIO_MODE_OUTPUT_PUSH_PULL);
26  XMC_GPIO_SetMode(PORT_LED_6, PIN_LED_6, XMC_GPIO_MODE_OUTPUT_PUSH_PULL);
27  XMC_GPIO_SetMode(PORT_LED_7, PIN_LED_7, XMC_GPIO_MODE_OUTPUT_PUSH_PULL); //LED7
28  //set LED Pins high (active low);
29  XMC_GPIO_SetOutputHigh(PORT_LED_1, PIN_LED_1);
30  XMC_GPIO_SetOutputHigh(PORT_LED_2, PIN_LED_2);
31  XMC_GPIO_SetOutputHigh(PORT_LED_3, PIN_LED_3);
32  XMC_GPIO_SetOutputHigh(PORT_LED_4, PIN_LED_4);
33  XMC_GPIO_SetOutputHigh(PORT_LED_5, PIN_LED_5);
34  XMC_GPIO_SetOutputHigh(PORT_LED_6, PIN_LED_6);
35  XMC_GPIO_SetOutputHigh(PORT_LED_7, PIN_LED_7);

37  //set P_ON Pins as Output
38  XMC_GPIO_SetMode(PORT_P_ON_TDA_1, PIN_P_ON_TDA_1, XMC_GPIO_MODE_OUTPUT_PUSH_PULL);
39  //TDA1
40  XMC_GPIO_SetMode(PORT_P_ON_TDA_2, PIN_P_ON_TDA_2, XMC_GPIO_MODE_OUTPUT_PUSH_PULL);
41  XMC_GPIO_SetMode(PORT_P_ON_TDA_3, PIN_P_ON_TDA_3, XMC_GPIO_MODE_OUTPUT_PUSH_PULL);
42  XMC_GPIO_SetMode(PORT_P_ON_TDA_4, PIN_P_ON_TDA_4, XMC_GPIO_MODE_OUTPUT_PUSH_PULL);
43  XMC_GPIO_SetMode(PORT_P_ON_TDA_5, PIN_P_ON_TDA_5, XMC_GPIO_MODE_OUTPUT_PUSH_PULL);
44  //TDA6
45  //P_PON low -> TDAs off state
46  XMC_GPIO_SetOutputLow(PORT_P_ON_TDA_1, PIN_P_ON_TDA_1);
47  XMC_GPIO_SetOutputLow(PORT_P_ON_TDA_2, PIN_P_ON_TDA_2);
48  XMC_GPIO_SetOutputLow(PORT_P_ON_TDA_3, PIN_P_ON_TDA_3);
49  XMC_GPIO_SetOutputLow(PORT_P_ON_TDA_4, PIN_P_ON_TDA_4);
50  XMC_GPIO_SetOutputLow(PORT_P_ON_TDA_5, PIN_P_ON_TDA_5);
51  XMC_GPIO_SetOutputLow(PORT_P_ON_TDA_6, PIN_P_ON_TDA_6);

52 }
53 void send_serialnumber_to_com(void) {
54     CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
55     CDC_Device_SendData(&VirtualSerial_CDC_Interface, "Serial Number TDA1\r\n", 20);
56     COM_send_int_as_string(tda5340_get_serial_number(TDA1));
57     CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
58     CDC_Device_SendData(&VirtualSerial_CDC_Interface, "Serial Number TDA2\r\n", 20);
59     COM_send_int_as_string(tda5340_get_serial_number(TDA2));
60     CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
61     CDC_Device_SendData(&VirtualSerial_CDC_Interface, "Serial Number TDA3\r\n", 20);
62     COM_send_int_as_string(tda5340_get_serial_number(TDA3));
63     CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
64     CDC_Device_SendData(&VirtualSerial_CDC_Interface, "Serial Number TDA4\r\n", 20);
65     COM_send_int_as_string(tda5340_get_serial_number(TDA4));
66     CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
67     CDC_Device_SendData(&VirtualSerial_CDC_Interface, "Serial Number TDA5\r\n", 20);
68     COM_send_int_as_string(tda5340_get_serial_number(TDA5));
69     CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
70     CDC_Device_SendData(&VirtualSerial_CDC_Interface, "Serial Number TDA6\r\n", 20);
71     COM_send_int_as_string(tda5340_get_serial_number(TDA6));
72     CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
73 }
74 void general_test(void) {
75     //uint8_t i = 0;
76     //i=tda5340_transfer(0, 0x05, 0xD0 , 0);
77     //
78     //if(i==0){
79     //    led_on(i);i=0;
80     //}
81     //test für TDA Lib von Felix

82
83     //tda5340_gpio_init(0);
84     //tda5340_init(0);
85     //uint32_t serialnumber = tda5340_get_serial_number(0);

86
87     //serialnumber -> LEDs

```

```

88 //for (int var = 0; var < 32; var++) {
89 //    if (serialnumber & (1 << var)) {
90 //        led_on(5);
91 //    }
92 //    led_on(6);
93 //    delay(4000000);
94 //    led_off(5);
95 //    led_off(6);
96 //}

98 set_TDA_status(0, 1);
99 delay(40000);

101 spi_init(spi_master_ch);
102 delay(40000);

104 tda5340_transfer(5, READ_FROM_CHIP, IS2, 0xFF);
105 delay(40000);
106 tda5340_transfer(5, READ_FROM_CHIP, 0xDB, 0);

108 tda5340_transfer(5, READ_FROM_CHIP, IS2, 0xFF);
109 delay(40000);
110 tda5340_transfer(5, READ_FROM_CHIP, 0xDB, 0);

112 //
113 //
114 //
115 //    uint16_t spi_array_tx[20] = { 0 };
116 //    spi_array_tx[0] = 0x05;
117 //    spi_array_tx[1] = 0xD3;
118 //    uint16_t spi_array_rx[20] = { 0 };
119 //    led_on(2);
120 //
121 //
122 //    set_TDA_status(0,1);
123 //
124 //
125 //    led_on(6);
126 //    led_on(7);
127 //
128 //        if (spi_array_rx[0] == 0 &&spi_array_rx[1] == 0&&spi_array_rx[2] ==
129 //            0&&spi_array_rx[3] == 0&&spi_array_rx[4] == 0&&spi_array_rx[5] == 0
130 //                && spi_array_rx[6] == 0 &&spi_array_rx[7] == 0&&spi_array_rx[7] ==
131 //                    0&&spi_array_rx[8] == 0&&spi_array_rx[9] == 0 ) {
132 //            led_off(6);
133 //        }
134 //
135 //    spi_init(spi_master_ch);
136 //    spi_transfer(spi_master_ch, 4, spi_array_tx, spi_array_rx, 20);
137 //
138 //
139 //    for (int var = 0; var < 20; var++) {
140 //        if (spi_array_rx[var] != 0) {
141 //            led_on(5);
142 //        }
143 //    }

149 // CDC_Device_SendData(&VirtualSerial_CDC_Interface, "Start Reading FIFO TDA 3\r\n
150 //    ", 26);
151 //
152 //    int8_t data_send[10] = { 1, 3, 5, 7, 9, 2, 4, 6, 8, 10 };
153 //    int8_t data_rec[10];
154 //    uint8_t lenght = 10;

```

```
154 // tda5340_fifo_rw(TDA3, 1, data_send, &lenght);
155 // tda5340_fifo_rw(TDA3, 0, data_rec, &lenght);
156 // COM_send_int_as_string(data_rec[0]);
157 // CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
158 //
159 // COM_send_int_as_string(data_rec[1]);
160 // CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
161 //
162 // COM_send_int_as_string(data_rec[2]);
163 // CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
164 //
165 // COM_send_int_as_string(data_rec[3]);
166 // CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
167 //
168 // COM_send_int_as_string(data_rec[4]);
169 // CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
170 //
171 // COM_send_int_as_string(data_rec[5]);
172 // CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
173 //
174 // COM_send_int_as_string(data_rec[6]);
175 // CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
176 //
177 // COM_send_int_as_string(data_rec[7]);
178 // CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
179 //
180 // COM_send_int_as_string(data_rec[8]);
181 // CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
182 //
183 // COM_send_int_as_string(data_rec[9]);
184 // CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
185 //
186 // COM_send_int_as_string(data_rec[10]);
187 // CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);
188 //
189 // CDC_Device_SendData(&VirtualSerial_CDC_Interface, "\r\n", 2);

191 CDC_Device_USBTask(&VirtualSerial_CDC_Interface);

193 // set_TDA_status(TDA_ALL, 0);
194 // set_TDA_status(TDA_ALL, 1);
195 //
196 // set_TDA_status(TDA_ALL, 0);
197 // set_TDA_status(TDA_ALL, 1);
198 // set_TDA_status(TDA_ALL, 0);
199 // set_TDA_status(TDA_ALL, 1);

204 }
```

Literatur
