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

Zur Lokalisierung von mobilen Sensorknoten, welche in einem Sub-GHz-Frequenzbereich von 868 MHz arbeiten, sollte eine energieeffiziente Art der Ortung umgesetzt werden. Dazu sollte mithilfe einer auf Feldstärke basierten Ortung die genaue Position der Sensorknoten festgestellt werden. Um dies zu ermöglichen wurde eine Basisstation mit sechs Transceiver-ICs der Bauart TDA5340 entworfen. Die Steuerung übernahm ein Mikrocontroller der Baureihe XMC4500. Diese Basisstation sollte dabei ein Auslesen der empfangen Daten sowohl über den USB-Standard, als auch über Ethernet als zweite Kommunikationsschnittstelle ermöglichen. Die dabei verwendete Hardware basierte zum Großteil auf Bauteilen des Herstellers Infineon. Die Platine der Basisstation wurde mit Altium Designer entwickelt und umgesetzt. Dabei wurde die Verbindung zwischen der Steuereinheit und den Transceivern mit dem SPI-Protokoll umgesetzt. Die einzelnen Sende-/Empfangseinheiten wurden dabei gleichmäßig in alle Raumrichtungen zeigend angeordnet, sodass auch die Abstahlung der Antennen über alle Raumrichtungen gleichförmig verteilt ist. Die Funksegmente der Platine wurden so gestaltet, das diese bei Bedarf abgetrennt und mit einer Kabelverbindung weiter voneinander entfernt werden konnten. Die Peripherie der verwendeten TDA5340 Transceiver generierte im Programmablauf nach einer Kommunikation mit dem Sensor ein Interrupt Signal. Dies erlaubte dem Mikrocontroller die Daten der einzelnen Empfangseinheiten auszulesen, zu speichern und zu einem späteren Zeitpunkt weiterzuleiten. Beim Messen der Feldstärke wurde ausgenutzt, dass die vorliegende Feldstärke bereits durch den TDA5340-Empfänger zur Verfügung gestellt wurde.

Abstract

Current localization measurements have been complex and consume plenty of energy. However almost every Receiver has knowledge of the elektrical field strength, correlating to distance vom 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. Therefor six identical Transceiver-ICs TDA5430 were arranged over all horizontal directions in space. By use of focused antenna the recognition of transmission direction could be corrected. Coordination of the ICs would be realized by a XMC4500 µC 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 Boards

EDA Electronic Design Automation

SPI Serial Peripheral Interface

DRC Design-Rule-Check

FIFO First In – First Out

SMA Sub-Miniature-A

LQFP Low Profile Quad Flat Package

BGA Ball Grid Array

JTAG Joint Test Action Group

TVS Transient Voltage Suppressor

LDO Low Drop-Out

SOT Small Outline Transistor

IC Integrierter Schaltkreis

SMD Surface Mounted Device

STEP Standard for the Exchange of Product Model Data

CAD Computer-aided design

NC Numerical Control

IDE integrated development environment

GUI Graphical User Interface

SDK Software development kit

CPU Central Processing Unit

USIC Universal Serial Interface Channel

ETH Ethernet MAC

USB Universal Serial Bus

GPIO General Purpose Input/Output

ISR Interrupt Service Routine

IRQ Interrupt Request

ERU Event Request Unit

ERS Event Request Select

ETL Event Trigger Logic

OGU Output Gating Unit

NVIC Nested Vectored Interrupt Controller

CMSIS Cortex Microcontroller Software Interface Standard

ASCII American Standard Code for Information Interchange

PLL Phasenregelschleife (phase-locked loop)

AGC automatic gain control

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

1.2 Zieldefinition

1.3 Projektmanagement

Als Versionskontrolle für das Projekt wurde Github eingesetzt.

Kapitel 2

Platinenaufbau

2.1 Vorüberlegungen

Um Sicherzustellen, das alle Antennen gleichmäßig in die sechs vorgegebenen Raumrichtungen abstrahlen, sollte bereits die Platine symmetrisch aufgebaut werden. Dazu wurden zuerst das Layout der sechs identischen Transceiver-Einheiten mit dem TDA5340 Baustein und den Antennen erstellt und anschließend gleichmäßig um die weiten für die Schaltung notwendigen 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 Boardss (PCBs). Ein solches Programm wird auch als Electronic Design Automations (EDAs) oder ECAD für electronic Computer-aided design (CAD) bezeichnet, da es den Entwickler bei der Umsetzung der Anforderungen in einen Schaltplan unterstützen soll. Wie viele andere EDA-Programme ist auch Altium Designer so aufgebaut das 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 Bibliotheken verwendete, 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 und Lötpads festgelegt. Da es Bauteile, wie den verwendeten Mikrocontroller, in verschiedenen Gehäusen geben kann,

besteht somit auch die Möglichkeit hier verschiedene Footprints zu wählen. Da viele Gehäuse herstellerübergreifend genormt sind, konnten teilweise bestehende Footprints genutzt werden oder diese mehrfach verwendet werden.

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 Ausmaße 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.

Wegen der Größte des Projekts wurde zur besseren Übersicht ein so genanntes „Multi-Sheet-Design“ erstellt. Dadurch war es möglich die verschiedenen funktionellen Blöcke der Basistation auf getrennte Blätter des Schaltplans zu verteilen. Der Mikrocontroller, die Spannungsversorung und der Transceiver sowie die für eine Netzwerkkommunikation notwendigen Bauteile wurden dabei auf getrennten Seiten angeordnet und die elektrischen Verbindungen erstellt. Da der Tranciever und die entsprechende Peripherie sechs mal genau identisch verwendet wurde und auch auf dem PCB-Substrat mehrfach mit Leiterbahnen verbunden und angeordnet werden musste, wurde hierfür ein so gennantes „Multi-Channel“-Design gewählt. In Altium Designer können mit diesem Feature identische Schaltungsteile einmal angeordnet und mit Leiterbahnen verbunden werden 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. Mit Hilfe eines übergeordneten Sheet-Symbols wurde der Schaltplan des Transceivers in den Schaltplan des Mikrocontroller eingefügt und diesem somit untergeordnet. Über Ports welche zum Schaltplansymbol hinzugefügt werden, lassen sich elektrische Verbindungen zwischen Netzen innerhalb des Schaltplans für Transceiver und Mikrocontroller erstellen. Dabei wird im Transceiver-Schaltplan ein Port und hinzugefügt, der mit elektrischen Netzen verbunden werden kann. Auf dem Schaltplan-Symbol 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 dem selben Pin des XMC4500 und den selben 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 Transceivereinheiten mit einen anderen Anschluss des Mikrocontroller verbunden sein mussten mussten deswegen mit dem Repeat-Kommando erstellt werden. So wird im untergeordneten Schaltplan, in diesem Fall dem des TDA, der Port beliebig benannt, etwa als „NCS“. Der auf dem Schaltplansymbol erstellte Port wird dagegen in „Repeat(NCS)“ umbenannt. 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>)“ aufweißen. 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 und einen Bereich in dem diese angeordnet sind und erlaubt das Anordnen aller enthaltenden Elemente auf der Platine.

Altium Designer nummeriert automatische die verwendeten Bauelemente durch um eine genaue Identifikation eine Bauteiles zu erlauben und erstellt automatische einen Aufdruck neben dem Lötpäd 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, das 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 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 im DRC sich überschneidende und falsch verbundene Leiterbahnen angezeigt werden, muss kein extra „Layout vs. Schematic“-Test durchgeführt werden, da durch den DRC Unterschiede zwischen Schaltplan und dem PCB-Layout bereits angezeigt würden.

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^{TM} Produktfamilie die energiesparende Lösungen für Funkanwengungen im Frequenzspektrum unterhalb von einem Gigaherz 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 auf drei gemeinsamen Leitungen übertragen, eine vierte Leitung dient dem XMC zur Auswahl des gewünschten Slaves für die Kommunikation. Diese „not Chip select“-Leitung (NCS) arbeitet active-low, sodass der jeweilige TDA5340 eine Interaktion akzeptier sobald diese vom XMC-Baustein auf Erdpotential gezogen wird.Von dein 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 XMC getriebenes Clock-Signal. Bei dem auf MISO und MOSI anliegenden Signal handelt es sich um ein unipolar kodiertes non-return-to-zero

Signal, welches einer logischen 0 bei Erdpotential 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 Signalpakete zwischengespeichert. Diese Zwischenspeicherung soll den Mikrocontroller entlasten, so können entsprechende Datenpakete dem TDA5340 mitgeteilt werden und dieser übernimmt selbsttätig eine korrekte Modulation und Übertragung mit den eingestellten Parametern.

Der TDA5340 kann sowohl mit einer Spannungsversorgungsspannung von $5V$ als auch bei $3{,}3V$ arbeiten. Da aber der XMC nur bei letzterer betrieben werden kann, wurde der TDA-Baustein und die externe Beschaltung einfachheitshalber auch auf $3{,}3V$ 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 könnten die gesamte Baugruppe von der Mutterplatine entfernt werden, was unter Umständen notwendig gewesen wäre um größere Unterschiede in der Signalstärke an den einzelnen Transceivern zu erhalten, und somit eine bessere Auflösung in der Ortung zu erlauben. Dazu wurden Anschlussleisten im Rastermaß $2{,}54mm$ an beiden Seiten der Sollbruchstelle vorgesehen. Die Verbindung der Transceiver-Einheiten mit der Hauptplatine wurde über diese Sollbruchstelle mit Leiterbahnen gewähleistet. Nach dem Abtrennen der TDA-Teilplatine, an der durch Bohrungen vorgesehenen Bruchstelle, wäre die elektrische Verbindung durch Kabel sichergestellt worden. Da es sich bei den zu übertragenden ausschließlich um digitale Signale handelt hätte dies unproblematisch mit ungeschirmten Flachbandkabeln ermöglicht werden können. Ein solches Vergrößern des Abstandes der Antennen war jedoch nicht notwendig. 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 dem XMC4500 über die Buchsenleisten zur Verfügung gestellt.

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 50Ω angepasst ist. Durch den Koaxialsteckverbinder konnte sichergestellt werden, das alle notwendigen Frequenzen auch korrekt und ungedämpft passieren können. Das Anpassnetzwerk zwischen dem integrierten Transceiver und der verwendeten SMA-Buchse diente der Leistungsanpassung zwischen den Pins des TDA5340 und der 50Ω-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 2,5pF wurde die korrekte Anpassung sichergestellt.

Zur Verbesserung der Hochfrequenzeigenschaften wurden die nach dem Freiräume zwischen den Leiterbahnen mit einer Kupferfläche gefüllt, die mit dem Masseanschluss kontaktiert 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 Bandweite 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 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 elektrischen Verbindungen zwischen den beiden Eingängen des Low Noise Amplifier und dem Anpassnetzwerk wurden mit dem „Differential Pair Routing"-Feature von Altium Designer erstellt. Durch ein im Schaltplan auf die postitive und die negative elektrischen Verbindung zwischen dem TDA5340 und dem Anpassnetzwerk wird das Leitungspaar als differentiell markiert. Anschließend kann mit dem interatkiven „Diffential Pair Routing" einer der beiden Leitungen begonnen werden. Altium Designer wird dabei selbstständig versuchen die zweite Leiterbahn des Paares so anzuordnen, das die beiden Leiterbahnen symmetrisch und parallel zueinandern liegen. Durch die Verwendung des „Diffential Pair Routing" versucht Altium Designer auch die Länge der beiden Verbindungen anzugleichen. Durch die Anpassungen der differentielle Leiterbahnen wird eine gleichmäßige Übertragung 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 das für den Oszillator benötigte Quarz mit einer Frequenz $f_{Crystal} = 21{,}948717$ MHz zwischen dem Integrierter Schaltkreis (IC) und vorgesehenen Stiftleiste platziert. Die Frequenz des benötigten Quarzes ergibt sich aus dem Zusammenhang

$$f_{Crystal} = f_{IF2} * 80 = \frac{f_{IF1}}{39} * 80 = \frac{10{,}7Mhz}{39} * 80 = 21{,}948717MHz \tag{2.1}$$

wobei die Zwischenfrequenz der ersten Stufe 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

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 Kern Cortex™-M4 des britischen Herstellers ARM™. Daneben ist der Mikrocontroller auf die von Infineon selbst entwickelte Entwicklungsumgebung DAVE™ angepasst. 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 ein Low Profile Quad Flat Package (LQFP)-Gehäuse verbaut. Durch die Wahl diese 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 verlötet werden.

Zur Verteilung der entstehenden Abwärme wurde auch in diesem Bereich der Platine frei gebliebener 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 $T_J = 150°C$ 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 $\overline{\textbf{PORST}}$-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 der in IEEE1149.1 festgelegten Standarts übereinstimmt. Verwendet wird hierfür die achpolige Variante des Debug-Steckers bei dem der Platine vom JTAG-Adapter Versorgungsspannung und Masse sowie Signale für Reset, Systemtakt und die Steuerleitung übergeben wird.

Für die Kommunikation des Mikrocontroller 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 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 führt. 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. 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.

2.3.3 Ethernet

Die Ethernetschnittstelle der Basisstation basiert auf dem RelaxKit 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 verbaute Stufe zur Interruptgenerierung nur über einen schwachen Pull-Up Widerstand verfügt, musste ein externer Widerstand von $1k\Omega$ 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, das 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 gewähleistet 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 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, wurde 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-Mikrocontroller nicht umgesetzt. Da jedoch ein entsprechendes Softwareprojekt für das RelaxKit von Infineon zur Verfügung gestellt wird

2.3.4 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 2,54mm 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 5V Ausgangsspannung, durch den Aufbau mit den beiden verwendeten Schottky-Dioden und die mögliche Eingangsspannung des nachfolgenden Reglers wäre jedoch eine angeschlossene 6V-Stromquelle (bei vernachlässigtem Spannungsabfall an der Diode) unproblematisch. Wegen der bereits erwähnten notwenigen Versorungspannung von 3,3V für den XMC4500 und die Transceiver wurde diese mit einem Low Drop-Out (LDO)-Regler angepasst. Dieser verwendete Spannungsregler der Bauart MCP1826S von Microchip Technology Inc. sollte die Eingangspannung 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 3,3V Output kontaktiert, beim verwendeten LDO mit Ground. Da es sich bei dem Spannungsregler um ein Surface Mounted Device (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 Kupferfläche. So wird die erzeugte Wärme gespreizt und kann gut an die Umgebung abgegeben werden. Bei Verwendung des zuerst eingeplanten IFX1117MEV33 wäre somit eine Kupferfläche auf 3,3V Potential zum Kühlen notwendig, welche elektrisch isoliert sein müsste. Wegen des XMC4500 und den anderen Bauteilen wäre somit nur eine Kupferfläche mit Abmessungen von etwa 15mm auf 16mm möglich, da die elektrischen Verbindungen bestehender Bauteile des Bereiches nicht unterbrochen werden sollten. 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 Kühlfäche 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 dies 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 Ausgabe aller Bauteile 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 Formen 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 Fertig der Platine durch den Hersteller erfolgte durch erstellte Gerber Daten, die ebenfals aus dem Output-Job-Manager generiert werden. 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.

Kapitel 3

Software

3.1 DAVE Entwicklungsumgebung

Das Programm DAVE™ (Digital Application Virtual Engineer) wird von Infineon Technologies AG entwickelt. Sie 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™ greift bei der Programmierung von Mikrocontrollern der XMC-Serie auf die so genannten 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 es 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 nun noch die Pins ebenfalls grafisch den Aufgaben zugeordnet werden, generiert DAVE™ den Programmcode mit den in den Apps enthaltenen Informationen-Mithilfe des DAVE™ Software development kit (SDK) können nicht nur die Parameter der APPs beim programmieren , sonder auch diese selbst grundlegend angepasst werden und das entwickeln eigener APPs ist möglichIm Verlauf dieser Arbeit wurden DAVE™ APPs jedoch nur in einem bereits existierenden Softwareprojekt für ein RelaxKit 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 Blocke 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 Universal Serial Interface Channel (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).

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 ebenfalls über die Peripheriebrücke PBA1 an die Bus Matrix und somit den Cortex M4 Kern angebunden. Das Modul stellt für jeden Pin die erste Funktionsauswahl bereit. Im „Port Input/Output Control Register“ des Moduls wird für jeden Pin festgelegt ob er als Eingang oder Ausgang belegt ist. Das momentane elektrische Potential am Eingang wird dann mit einem Schmitt-Trigger in ein binäres logisches Signal übersetzt. Ist der Pin ein Eingang, so kann dieser dort zusätzlich invertiert werden. Wird ein Pin des XMC als Ausgang konfiguriert, so kann gewählt 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 den einen Treiber in Open-Drain Konfiguration oder durch Push-Pull erzeugt werden soll. Zur Nutzung eines Ausgangs mit der 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**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.

3.2.2 USIC

Die ICs der XMC-Familie verfügen über ein Modul zu Kommunikation über diverse serielle Protokolle, den 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 verweden. Die Mikrocontroller unterstützen somit die folgenden Protokolle UART/SCI, SPI in einfach, doppelt und quad-Ausführung, IIC/I2C, IIS/I2S und LIN. Für diese Arbeit wurde alle Kommunikation mit einem gemeinsamen 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 und nicht über die Bibliothek erfolgen müssen.

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 eine 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 Eingänge 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 diesen 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 den Interrupt und sorgt dafür, das 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 sowohl als Host oder als USB-Slave arbeiten. In diesem Fall wurde der IC als Slave betrieben. Das USB-Modul besitzt über eine eigene Interruptsteuerung und ist damit eine der gerade 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. Diese 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 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 der Auswahl des entsprechenden Transceivers über das Slave-Select-Signal. Dazu initalisiert die bibliothek zuerst den USIC entsprechend den in der Headerdatei vorgegebenen Pins für MISO, MOSI und den Pins für den jeweiligen Slave.

3.3.2.1 SPI Übertragung

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, was die Lesbarkeit erheblich beeinflusst. 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 notwenig, 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, das 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™. Die Bereitstellung des virtuellen seriellen Ports erfolgt auf Seiten des XMC über die LUFA (Lightweight USB Framework for AVRs)-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, das 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.

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 Bereits hier wurde die Übertragung übe den seriellen COM-Port initialisiert. Nach dem 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 den 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 initalisiert und anschließend in den Resceiver-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.

3.4.1 Konfiguration der Funkmodule

Bei der Initialisierung erhalten diese die gewünschten Werte für die Sende- und Empfangs-Frequenzen. Diese werden über die Teilerrate für die Phasenregelschleife (PLL) übergeben und ermittelt. Über die entsprechenden Register wird auch das Verhalten bei

3.4.2 interruptbasierte Datenerfassung

Die Erfassung der Daten erfolgt im Programmablauf innerhalb der dauerhaften Ablaufschleife. In der Interrupt Service Routine, 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, das 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, das 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 notwenig, da es sich dabei um so genannte „Read-Clear“-Register handelt, welche nach dem Auslesen über SPI automatisch zurückgesetzt werden. 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, da eine erneute Übertragung diese überschreiben würde. Jedoch ist das Auslesen der Werte bei weitem nicht so zeitkritisch wie die ankommenden Interrupts. Das Abfragen der empfangenen Daten aus dem FIFO-Speicher hat eine noch geringer Priorität, da dieser die Datenpakete bis zum Auslesen behält. 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.

3.4.3 Weiterleitung der erfassten Daten

Das 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 dieser 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 15 zu erkennen. Die Zeichenfolge $\backslash r \backslash n$ stellt dabei den Übergang in eine neue Zeile in der Ausgabekonsole dar. Wie in C üblich werden Strings in doppelten Anführungszeichen im Quelltext eingefügt.

```
1 COM_send_string("#################### Übertragung erkannt ####################\r\n");
2 COM_send_string("Übertragung Nummer ");
3 COM_send_int_as_string(transfernumber);
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 115200Bd und acht Daten Bits pro Zeichen empfangen wurde und die entsprechenden Einstellungen im Programm vorgenommen 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 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 **??** erkennen. Zur Darstellung einer eingegangenen Übertragung an der Basisstation sollte die LED Nummer 7 nach jedem Empfangen kurz rot aufblinken.

Kapitel 4

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 ... und eine Empfangsfrequenz von ... programmiert was durch die Werte für die PLL im TDA5340 eingestellt wurde. Die Basisistation wurde mit sechs Antennen, die einen Verstärkungsfaktor von 3,6dBi und eine Mittenfrequenz von 868MHz aufwiesen, bestückt. 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 wurden die zuvor einprogrammierte Zeichenkette 1,2,3,4,5,6,7,8,9 ausgesendet. Der Abstand zur Basisstation betrug im ersten Test 3,30m und wurde nach jeder Übertragung um 30cm verringert. In einem zweiten Test wurde ebenfalls mit einem Abstand von 3,3m gestartet. Nach jeder Übertragung wurde der Sender 30cm 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 dieser verhindert werden. Die Basisstation wurde auf einem 70cm 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, das zwar in jedem Test zehn mal durch das Drücken des Tasters eine Übertragung ausgelöst wurde, jedoch öfter eine Übertragung an der Basistation registriert wurde. Im Ersten Feldtest wurden fünfzehn, im zweiten sogar sechzehn gültige Übertragungen von der Basisstation an den Hostcomputer ausgegeben. 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, das es sich bei diesen Werten noch um die Messungen aus dem ersten Transfer handelt. Somit wäre zu folgern, das 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 -100dBm, lediglich bei der letzten Übertragung, welche bei einem Abstand von 30cm stattfand, konnte hier ein Wert von -97dBm gemessen werden. Da diese sehr schwachen Empfangsleistungen in vorherigen Tests nicht auftraten ist zu vermuten, das 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 -114dBm entsprach dem minimal möglichen Ausgabewert der dafür verwendeten Funktion. Es ist also davon auszugehen, das 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, das einzelne Transceiver eine Übertragung nicht erkennen konnten.

Zu beachten ist, das 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 den Taster 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 eine Ausgabe der Sendungsnummer am Relax Kit und ein neuer zu sendender Datensatz für jede Übermittlung hilfreich.

Kapitel 5

Zusammenfassung und Ausblick

Zusammenfassend kann man sagen, das 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 Eruieren 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 verweden. 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 notwenigen 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, sonder auch über die ebenfalls mit dem XMC verbundenen PP0 und PP1 Pins. Somit wäre ein verteilen auf einzelne Kanäle der ERU wahrscheinlich möglich.

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Kapitel 6

Anhang

EINFÜGEN: Layout Stefan Erhard Bilder Altium 3D Modelle Dateien und Dateiverzeichnis (s. XMC Peripheral lib startseite)

6.1 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

6.2 3D-Daten

6.2.1 Gehäuse

6.3 Layout TDA5340 Aufsteckboard

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.

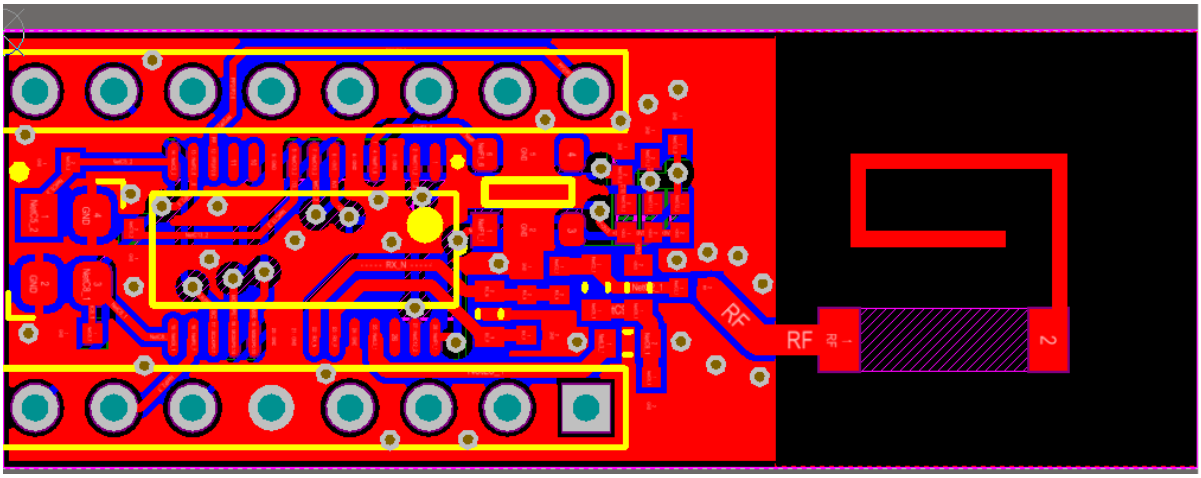

Abb. 6.1: Layout des Aufsteckboards mit dem TDA5340

6.4 Quellcode

6.4.1 Main.c

```
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);
```

```
  delay(40000);
  tda5340_gpio_init(TDA_ALL);
  spi_init(spi_master_ch);

  delay(500);
  delay(500);

  led_on(LED_ALL);
  delay(4000000);
  led_off(LED_ALL);
  led_on(LED1);
  delay(4000000);
  led_off(LED_ALL);
  led_on(LED1);

//  set_TDA_status(TDA1, 1);
//  delay(4000);
//  set_TDA_status(TDA2, 1);
//  delay(4000);
//  set_TDA_status(TDA3, 1);
//  delay(4000000);
//  set_TDA_status(TDA4, 1);
//  delay(4000);
//  set_TDA_status(TDA5, 1);
//  delay(4000000);
//  set_TDA_status(TDA6, 1);
//  delay(4000);

  delay(40000);
  tda5340_init(TDA1); //Verzögerung nach set Status muss gro  genug sein bis SPI Kom möglich ist    delay(45000); müsste das richtige sein
  tda5340_set_mode_and_config(TDA1, RX_MODE, 0);

  delay(40000);

  //für gesamte Platine:
  tda5340_init(TDA2);
  tda5340_set_mode_and_config(TDA2, RX_MODE, 0);
  tda5340_init(TDA3);
  tda5340_set_mode_and_config(TDA3, RX_MODE, 0);
  tda5340_init(TDA4);
  tda5340_set_mode_and_config(TDA4, RX_MODE, 0);
  tda5340_init(TDA5);
  tda5340_set_mode_and_config(TDA5, RX_MODE, 0);
  tda5340_init(TDA6);
  tda5340_set_mode_and_config(TDA6, RX_MODE, 0);

//  Ablaufschleife START ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  COM_send_string("Initialisierung beendet - ");
  uint8_t data_recieved = 0;
  uint32_t istateTDA1 = 0, istateTDA2 = 0, istateTDA3 = 0, istateTDA4 = 0, istateTDA5 = 0, istateTDA6 = 0;
  uint8_t lengthTDA1 = 0, lengthTDA2 = 0, lengthTDA3 = 0, lengthTDA4 = 0, lengthTDA5 = 0, lengthTDA6 = 0;
  uint32_t transfernumber = 0;
  uint32_t led_ctr = 0;
  char rx_data_TDA1[36] = { 1, 0, 3, 0, 5, 6, 7, 8, 9 };//TODO: Test ob er auch wiklich gesetzt wird; aktuell sind vor und nach dem empfangen inhalt der variable gleich
  char rx_data_TDA2[36] = { 1, 0, 3, 0, 5, 6, 7, 8, 9 };
  char rx_data_TDA3[36] = { 1, 0, 3, 0, 5, 6, 7, 8, 9 };
  char rx_data_TDA4[36] = { 1, 0, 3, 0, 5, 6, 7, 8, 9 };
  char rx_data_TDA5[36] = { 1, 0, 3, 0, 5, 6, 7, 8, 9 };
  char rx_data_TDA6[36] = { 1, 0, 3, 0, 5, 6, 7, 8, 9 };
  struct rssi {
    uint8_t pmf;
    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)
              >> 1;
144       istateTDA1 &= ~(1 << 1);
145     }
146     if (istateTDA2 & (1 << 1)) {
147       rssiTDA2.pmf = tda5340_transfer(TDA2, READ_FROM_CHIP, RSSIPMF, 0xFF);
148       rssiTDA2.rx = tda5340_transfer(TDA2, READ_FROM_CHIP, RSSIRX, 0xFF);
149       rssiTDA2.agc = (tda5340_transfer(TDA2, READ_FROM_CHIP, AGCADRR, 0xFF) & 0x06)
              >> 1;
150       istateTDA2 &= ~(1 << 1);
151     }
152     if (istateTDA3 & (1 << 1)) {
153       rssiTDA3.pmf = tda5340_transfer(TDA3, READ_FROM_CHIP, RSSIPMF, 0xFF);
154       rssiTDA3.rx = tda5340_transfer(TDA3, READ_FROM_CHIP, RSSIRX, 0xFF);
155       rssiTDA3.agc = (tda5340_transfer(TDA3, READ_FROM_CHIP, AGCADRR, 0xFF) & 0x06)
              >> 1;
156       istateTDA3 &= ~(1 << 1);
```

```
157     }
158     if (istateTDA4 & (1 << 1)) {
159       rssiTDA4.pmf = tda5340_transfer(TDA4, READ_FROM_CHIP, RSSIPMF, 0xFF);
160       rssiTDA4.rx = tda5340_transfer(TDA4, READ_FROM_CHIP, RSSIRX, 0xFF);
161       rssiTDA4.agc = (tda5340_transfer(TDA4, READ_FROM_CHIP, AGCADRR, 0xFF) & 0x06)
              >> 1;
162       istateTDA4 &= ~(1 << 1);
163     }
164     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]);

308         }
309         COM_send_string("\r\n\r\n");

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]);

332         }
333         COM_send_string("\r\n\r\n");

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]);

356         }
357         COM_send_string("\r\n\r\n");
```

```
      COM_send_string("TDA5:");
      COM_send_string("\r\nPMF:");
      COM_send_int_as_string(rssiTDA5.pmf);
      COM_send_string("\r\nPRX:");
      COM_send_int_as_string(rssiTDA5.prx);
      COM_send_string("\r\nRX:");
      COM_send_int_as_string(rssiTDA5.rx);
      COM_send_string("\r\nPPL:");
      COM_send_int_as_string(rssiTDA5.ppl);
      COM_send_string("\r\nAGC:");
      COM_send_int_as_string(rssiTDA5.agc);
      COM_send_string("\r\nEmpfangsleistung (dBm):");
      if (dig_to_dbm(rssiTDA5.ppl, rssiTDA5.agc) < 0) {
        COM_send_string("-");
      }
      COM_send_int_as_string(abs(dig_to_dbm(rssiTDA5.ppl, rssiTDA5.agc)));
      COM_send_string("\r\n");
      COM_send_string("Empfangene Daten:");
      for (int i = 0; i <= lengthTDA5; ++i) {
        COM_send_int_as_string(rx_data_TDA5[i]);

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

      COM_send_string("TDA6:");
      COM_send_string("\r\nPMF:");
      COM_send_int_as_string(rssiTDA6.pmf);
      COM_send_string("\r\nPRX:");
      COM_send_int_as_string(rssiTDA6.prx);
      COM_send_string("\r\nRX:");
      COM_send_int_as_string(rssiTDA6.rx);
      COM_send_string("\r\nPPL:");
      COM_send_int_as_string(rssiTDA6.ppl);
      COM_send_string("\r\nAGC:");
      COM_send_int_as_string(rssiTDA6.agc);
      COM_send_string("\r\nEmpfangsleistung (dBm):");
      if (dig_to_dbm(rssiTDA6.ppl, rssiTDA6.agc) < 0) {
        COM_send_string("-");
      }
      COM_send_int_as_string(abs(dig_to_dbm(rssiTDA6.ppl, rssiTDA6.agc)));
      COM_send_string("\r\n");
      COM_send_string("Empfangene Daten:");
      for (int i = 0; i <= lengthTDA6; ++i) {
        COM_send_int_as_string(rx_data_TDA6[i]);

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

      led_ctr = 400000;
      led_on(LED7);
      data_recieved = 0;
    }

    if (led_ctr) {
      led_ctr--;

      if (!led_ctr)
        led_off(LED7);
    }
  }
//  Ablaufschleife ENDE
    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

////TESTMODUL-START
    ----------------------------------------------------------------------------------------------
//  char tx_data[36] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

```
423 //  char rx_data[36] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
424 //  COM_send_string("aktuell Test mit TDA6\r\n");
425 //  uint8_t length = 0;
426 //
427 //  struct {
428 //    uint8_t pmf;
429 //    uint8_t prx;
430 //    uint8_t rx;
431 //    uint8_t ppl;
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 Iterruots 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 }
```

6.4.2 ISRs.c

```
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 OGU0 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 OGU0 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 }
```

6.4.3 Init.c

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

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

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]  ==
        0&&spi_array_rx[3]  ==  0&&spi_array_rx[4]  ==  0&&spi_array_rx[5]  ==  0
129 //            && spi_array_rx[6]  ==  0 &&spi_array_rx[7]  ==  0&&spi_array_rx[7]  ==
          0&&spi_array_rx[8]  ==  0&&spi_array_rx[9]  ==  0  ) {
130 //          led_off(6);
131 //        }
132 //
133 //
134 //  spi_init(spi_master_ch);
135 //    spi_transfer(spi_master_ch, 4, spi_array_tx, spi_array_rx, 20);
136 //
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
            ", 26);
150     //
151     //  int8_t data_send[10] = { 1, 3, 5, 7, 9, 2, 4, 6, 8, 10 };
152     //  int8_t data_rec[10];
153     //  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