Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Tom Rostock

Entwurf und Realisierung eines FPGA basierten Software Defined Radios mit Echtzeit-Streaming-Schnittstelle zum PC

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Tom Rostock

Entwurf und Realisierung eines FPGA basierten Software Defined Radios mit Echtzeit-Streaming-Schnittstelle zum PC

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer nat Jürgen Reichardt
Zweitgutachter : Prof. Dr. -Ing. Jürgen Missun

Abgegeben am 11. August 2011

Tom Rostock

Thema der Bachelorthesis

Entwurf und Realisierung eines FPGA basierten Software Defined Radios mit Echtzeit-Streaming-Schnittstelle zum PC

Stichworte

Software-Defined-Radio, Echtzeit, FM-Rundfunk, Field-Programmable-Gate-Array, digitale Signalverarbeitung, Microblaze, PowerPC, USB, Ethernet, UDP

Kurzzusammenfassung

Heutige digitale Hardware wird immer leistungsstärker, günstiger und kann auf einer abstrakteren Ebene beschrieben werden. Hierzu gehören auch Field-Programmable-Gate-Arrays (FPGA), welche es erlauben, komplexe Signalverarbeitung zu implementieren und diese später an geänderte Spezifikationen anzupassen. Software-Defined-Radios (SDR) nutzen diesen Vorteil der digitalen Hardware, um flexible Plattformen zu schaffen, die direkt mit der analogen Hardware der untersten Übertragungsschicht zusammenarbeiten und so den Rechenaufwand minimieren. Mit dieser Bachelorarbeit soll die Implementierung eines SDR gezeigt werden, welches auf Basis eines FPGA erlauben soll, FM-Rundfunksignale zu empfangen und diese in Echtzeit an einen PC für eine FM-Demodulation zu übermitteln.

Tom Rostock

Title of the paper

Design and Implementation of an FPGA-based software defined radio with real-time streaming interface for a PC

Keywords

Software Defined Radio, Realtime, FM-Broadcasting, Field Programmable Gate Array, Digital Signal Processing, Microblaze, PowerPC, USB, Ethernet, UDP

Abstract

Today's digital hardware is becoming more powerful, cheaper and can be described on a higher level. This includes also Field Programmable Gate Arrays (FPGA), which allows the implementation of complex signal processing algorithms with the option to modify it afterwards. Software Defined Radios (SDR) use this advantage of digital hardware to provide a flexible platform, interfacing directly with the analog hardware of the lowest transmission layer, to reduce the computational effort. This bachelorthesis shows the design and implementation of a SDR, which allows to receive FM-Broadcasting signals and its real-time transmission to a PC for FM demodulation.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einführung

Eine Welt ohne Funkübertragung ist kaum vorstellbar. Ob es Mobiltelefone, Kraftfahrzeuge, PC-Peripherie oder auch Radio-Frequency-Identification-Chips (RFID) im Supermarkt sind, sie alle nutzen elektromagnetische Wellen (Funktechnik), um digitale/analoge Daten von einem Ort zum Anderen zu übertragen ohne eine Kabelverbindung zu nutzen. Für den Anwender ist Funk eine attraktive Technik und wird somit in Zukunft immer stärker weiterentwickelt werden. Das Bestreben nach besserer Funktechnik wird heute nur durch die zu Grunde liegende Hardware und die verfügbaren Frequenzbänder begrenzt. Moderne Hardware wird immer leistungsfähiger und kann höherfrequente Signale effizienter verarbeiten, doch neue Frequenzbänder für mehr Kapazität sind begrenzt. Zum Einen werden diese von der Bundesnetzagentur verwaltet. Zum Anderen setzt die Erschließung von höheren Frequenzbereichen eine bessere Signalverarbeitung und schnellere Signalverarbeitung voraus. Diese Arbeit setzt an den Punkt *bessere Signalverarbeitung* an, um mit einer langsameren Signalverarbeitung eine effiziente Signalverarbeitung von Signalen aus hohen Frequenzbändern zu erreichen.

Funkübertragungssysteme bestehen aus analoger Hardware was zur Folge hat, dass diese fest verbaut und verdrahtet ist. Ein analoges Filter zum Beispiel müsste über Drehpotentiometer konfiguriert und abgeglichen werden. Somit muss eine aufwändige Anpassung manuell durch den Menschen durchgeführt werden, bis das Filter auf eine neue Spezifikation angepasst ist. Teilweise ist dies auch gar nicht möglich, sodass direkt eine neue Entwicklung nötig ist.

Analog-Digital-Umsetzer (ADC) ermöglichen die Umsetzung analoger Signale in digitale, sodass am Ausgang des ADC ein digitaler Datenstrom erzeugt wird. Ein digitaler Datenstrom kann auch mit Sicherungsmaßnahmen versehen werden, die es erlauben, Fehler zu erkennen oder diese sogar zu beheben. Bei einer analogen Übertragung ist dies schwer möglich.

Heutige Geräte, die Funktechnik nutzen, sind meist digitale Systeme (Mobiltelefone, Navigationsgeräte, PC, usw.), welche im Kern digitale Signale verarbeiten können. Digitale Systeme werden heutzutage durch steigende Taktraten und bessere Systemlösungen immer schneller und schaffen einen immer höheren Datendurchsatz bei der Verarbeitung von digitalen Datenströmen. Des Weiteren kann heutige digitale Hardware flexibel gestaltet werden. Ein digitaler Signalprozessor (DSP) zum Beispiel, ist ein spezieller Prozessor der durch seine

Architektur für die digitale Signalverarbeitung ausgelegt wurde und erreicht heute Taktraten von 1.5 GHz (Texas Instruments (TI) Integra [1]). Die Arbeit des DSPs ist durch eine Software definiert, das heißt eine schnelle Anpassung an geänderte System-Spezifikation ist einfach durch eine neue Programmierung möglich. Dies gilt ebenso für Embedded Prozessoren, Mikrocontroller (uC) oder Field-Programmable-Gate-Arrays (FPGA). FPGAs arbeiten jedoch nicht mit einem sequentiellen Programm, vielmehr kann die innere Struktur mit digitaler Logik beschrieben werden und somit parallel Daten verarbeiten.

Über Funk zu übertragene Daten werden mit Hilfe geeigneter Modulationsverfahren in Frequenzbänder aufgeteilt, um parallel mit anderen Daten übertragen zu werden und so das zur Verfügung stehende Frequenzband[2] effizient ausnutzen. Für modulierte Signale in höheren Frequenzbändern bedeutet dies allerdings auch, dass das digitale Verarbeitungssystem mit einem höherem Takt arbeiten muss. Es sind viele Proben des analogen Signals nötig, um die Informationen des Ursprungssignals im digitalen zu erhalten (Shannon-Nyquist-Abtasttheorem). Auch wenn heutige Signalverarbeitungssysteme mit diesen hohen Verarbeitungsraten umgehen könnten, ist dieses nicht effizient. Eine Reduzierung der Verarbeitungsrate des gewünschten Frequenzbereiches kann unnötige Signalinformationen eliminieren und den Datenstrom verlangsamen. Hierfür werden die Frequenzbänder analog oder digital verschoben, um dann bequem von einem digitalen System mit einer niedrigen Taktrate verarbeitet zu werden. Somit ist es möglich eine leistungsschwächere und auch günstigere Hardware, bei vergleichbarer Signalverarbeitungsqualität, einzusetzen.

Digitale Systeme sind flexibel anpassbar, steuerbar, fehlertoleranter und zudem auch günstig, sodass es sinnvoll ist, so viel und so nah an der physikalischen Schicht wie möglich die Signalverarbeitung von der analogen Seite in die digitale zu verlagern. Hieraus ist der Begriff Software Defined Radio (SDR) ableitbar. *Software Defined* steht für flexible Umprogrammierbarkeit und *Radio* für die Funkübertragung.

1.1 Ziel der Thesis

Motiviert wurde diese Arbeit von einer früheren Masterarbeit [29] eines Studenten der HAW-Hamburg. Dieser hat sich damit beschäftigt, einen Intellectual-Property-Core (IP-Core) als Soft-Core, den DDC-Core, zu realisieren mit dem es möglich ist, über einen ADC, Signale aus höheren Frequenzbändern auf eine Zwischenfrequenz (ZF) oder das Basisband zu mischen. Hierbei wird gleichzeitig eine IQ-Demodulation, sowie Tiefpassfilterung und Dezimierung der zwei Kanäle (I und Q) erreicht (nach dem Digital-Down-Conversion (DDC) Prinzip.

[1] Informationen zum TI Integra: http://focus.ti.com/docs/prod/folders/print/tms320c6a8168.html.

[2] Die Frequenzbänder werden in Deutschland von der Bundesnetzagentur verwaltet. http://www.bundesnetzagentur.de.

Siehe Kapitel 2.5). Ein kapselndes Embedded System im FPGA mit Universal-Serial-Bus (USB) Schnittstelle zum PC dient als Testsystem für den DDC-Core und ermöglicht das Auslesen der Daten zur Analyse in MATLAB.

Für Laborübungen an der HAW-Hamburg in Fakultät Technik und Informatik soll ein SDR-System entstehen, um Studenten ein System zu bieten, mit dem Algorithmen auf reale Funksignalen (zum Beispiel eine Demodulation) angewendet und dargestellt werden können. Hierfür ist es notwendig, dass der bestehende DDC-Core in ein System integriert wird, das reale Funksignale empfangen und diese über den DDC-Core in Echtzeit an den PC senden kann. Die Übertragung sollte von möglichst vielen PC-Systemen (plattformübergreifend) erreichbar sein und die Möglichkeit bieten, um mehrere Datenströme erweitert zu werden. Ein Schnittstelle zu MATLAB muss Studenten ermöglichen, Daten über ein begrenztes Zeitintervall aufzunehmen und diese dann mit den gewünschten Algorithmen in MATLAB zu testen. Des Weiteren soll ein PC-basiertes FM-Rundfunkradio entstehen, das die Echtzeitfähigkeit des Systems bestätigen soll (Demonstrator).

Diese Arbeit wird auf Basis eines Xilinx FPGA-Boards (Xilinx ML507 mit Virtex-5 FPGA) und dem LTC2206 ADC umgesetzt werden, die bereits Bestandteil der Masterarbeit [29] waren. Es soll ein einfaches SDR-System auf Basis des DDC-Cores erstellt werden, welches die gewünschte Echtzeitübertragung und die Schnittstelle zu MATLAB implementiert. Zudem soll ein einfaches analoges Frontend reale Funksignale, hier FM-Rundfunk (87,50 MHz bis 108,00 MHz - VHF-2) als Testumgebeung, dem DDC-Core bereitstellen. Ein FM-Rundfunkradio, als Demonstrator, auf dem PC soll den Datenstrom kommend vom SDR in Echtzeit dekodieren, sowie hörbar machen. Für weitere Frequenzbänder kann später ein komplexes, steuerbares Frontend entstehen.

1.2 Aufbau dieses Dokuments

Diese Arbeit beinhaltet Informationen oder setzt Grundlagen voraus, die aus verschiedenen Fachgebieten kommen. Somit ist der gesamte Text mit Fußnoten durchzogen. Eine Fußnote gilt nicht als Hinweis auf eine zitierte Informationsquelle. Eckige Klammern mit einer Ziffer kennzeichnen weiterführende Referenzen zum Literaturverzeichnis.

In der *Einführung* (1) wurde ein Einstieg in die Thematik der Arbeit gegeben, sodass der Leser weiß worum es in dieser Arbeit geht.

Voraussetzungen zum Verständnis der Arbeit sind dem Kapitel *Grundlagen* (2) zu entnehmen. Hierzu gehört die Erklärung des Begriffes *SDR* oder zum Beispiel auch *FPGA*. Es wird jedoch nicht auf Grundlagen der Signalverarbeitung eingegangen, diese Informationen können zum Beispiel in [6] oder Sekundärquellen nachgelesen werden.

Das *Konzept* zur Realisierung und die nötigen *Analyse*n hierzu sind im Kapitel 3 zu finden. Es wird gezeigt werden, wie sich das System aus [29] zusammensetzt und dieses arbeitet. Eingegangen wird auf die möglichen Übertragungsmöglichkeiten für dieses SDR und es wird geklärt, welche praktische Übertragungsraten erreicht werden können. Des Weiteren werden Fragen über das analoge Frontend und die Signalverarbeitung beantwortet. Abschließend ist das Konzept für dieses SDR gezeigt.

Eine Beschreibung über die Vorgehensweise und die Realisierung der Teil- und des Endsystems sind im Kapitel *Realisierung* (4) untergebracht. Teilsysteme werden sofort getestet, um später aufsetzende Systeme nur mit getesteten Teilsystemen zusammen zu bringen.

Im Kapitel *Systemtest* (5) werden alle bereits durchgeführten Teiltests kurz zusammengefasst, um ein Gesamtergebnis über das realisierte System zu erhalten.

Eine abschließende *Zusammenfassung* (6) der gesamten Arbeit macht erkennbar, was gefordert war und wie, beziehungsweise ob, dieses umgesetzt wurde. Ein *Ausblick* zeigt auf, was aus dem Thema der Arbeit oder auch der Arbeit selbst in Zukunft gemacht werden könnte. Hierzu gehört die Verfeinerung des Systems oder auch wie neue Bachelorarbeiten anknüpfen könnten.

2 Grundlagen

2.1 Field Programmable Gate Array

Digitale Systeme bestehen zu großen Teilen aus logischen Gattern wie AND, OR, NOT und Flip-Flops. Programmable-Logic-Devices (PLD), wie Field-Programmable-Gate-Arrays (FPGA), bieten, sehr einfach gesagt, eine Plattform die es erlaubt, ein solches digitales System umzusetzen. FPGAs sind eine sehr flexible Untergruppe der PLDs, bei welchen die Logik durch kleine RAM Speicher, LUTs, definiert wird. Complex-Programmable-Logic-Devices (CPLD) gehören ebenfalls zu der Gruppe der PLDs, hierbei wird die Logik jedoch durch eine programmierbare UND-ODER-Logik Struktur definiert. FPGAs sind moderne Chips, um komplexe digitale Systeme flexibel zu realisieren. CPLDs hingegen sind mehr für Glue Logic geeignet und können als das Scratchpad unter den PLDs angesehen werden.

Im allgemeinen bestehen FPGAs aus einer flexiblen Verbindungsmatrix und mehreren Logikblöcken (LB) mit Look-Up-Tabellen (LUT). LUTs sind als Speicher ausgelegt und liefern zu jeder binären Kombination am Eingang einen Ausgangswert. Den LUTs sind meist, je nach Hersteller, zusätzliche Logik wie Flip-Flops, Treiber und Multiplexer nachgeschaltet, um eine flexibel nutzbare Struktur der Logik zu erreichen. Durch eine flexibel programmierbare Verbindungsmatrix können alle Blöcke innerhalb eines FPGA verschaltet werden. FPGAs beinhalten zudem Ein- und Ausgangsblöcke (IOB) zur Kommunikation mit der Außenwelt, Speicher oder gar ganze Hard-Cores[1] wie CPUs oder Peripherie, die durch ihre feste Implementierung bereits stark optimiert sind. Da die LUTs aus flüchtigem Speicher aufgebaut sind, zeigt sich ein großer Nachteil. Die programmierte Logikstruktur ist nur solange existent, wie eine Spannungsversorgung angelegt ist. Das heißt, zu einem FPGA muss immer ein extra Festwertspeicher mit Programmierlogik bestehen, wodurch eine Initialisierung des Systems vorgenommen wird.

FPGA werden meist mit einer Hardware-Description-Language (HDL) wie Very-High-Speed-Integrated-Circuit-HDL (VHDL) oder Verilog beschrieben, welches eine Hardware-Synthese erfordert. Synthesierter VHDL-Code wird in Netzlisten abgebildet, welche anschließend auf

[1] Stellen bereits in Hardware vorhandene Subsysteme im FPGA dar. Zum Beispiel CPUs die durch ihre Implementierung in Hardware sehr stark optimiert sind und mit wenig Aufwand über die Verbindungsmatrix im FPGA genutzt werden können.

die jeweilige FPGA-Architektur gemapped und durch einen Routing-Prozess über die Verbindungsmatrix verbunden werden. Für diesen Zweck stellt Xilinx die ISE Design Suite. Sie dient zu der Beschreibung in VHDL/Verilog, bietet jedoch ebenso die Möglichkeit auf einer höheren Abstraktionsschicht zu arbeiten. Dieses beschleunigt den Entwurfsvorgang und sorgt für mehr Übersicht (siehe EDK[2] und Systemgenerator for DSP[3]).

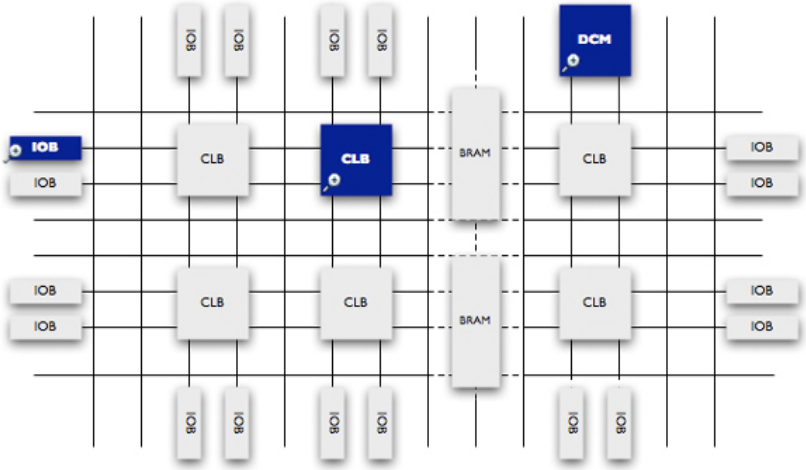

Abbildung 2.1: Innere Struktur eines Xilinx FPGA. *Quelle: [42]*

Als Beispiel für den internen Aufbau eines FPGA soll eine allgemeine FPGA-Struktur von Xilinx dienen (Abbildung 2.1). Eine systemweite und programmierbare Verbindungsmatrix liegt unter dem gesamten FPGA. An den Rändern des Chips liegen die IOBs für externe Signale. Digital-Clock-Manager (DCM) im System erlauben das Erzeugen, Kontrollieren und Regeln von Taktsignalen. Sie beinhalten Phase-Locked-Loops (PLL) und Delay-Locked-Loops (DLL) zur Regelung und Generation der Taktsignale. CLBs sind durch die gesamte Verbindungsmatrix verteilt und implementieren die Möglichkeit, Logik abzubilden. CLBs selbst bestehen wiederum aus Slices, die eine kleine Unterstruktur im FPGA bilden. Diese Unterstruktur ist speziell zur unkomplizierten Bildung von Schieberegistern oder Rechenwerken (optimiertes Routing für die Weitergabe von Überträgen) aufgebaut worden. In den Slices sitzen die eigentlichen LUTs mit Flip-Flops und zahlreichen Bauteilen zur Signalrückkopplung und Weiterleitung. Außer diesen Standardblöcken implementiert Xilinx auch, je Produkt, verschiedene Hard-Cores im FPGA. Im Virtex-5 FXT von Xilinx wäre dies zum Beispiel eine PowerPC440 CPU. Diese native Implementierung bietet ein sehr großes Potenzial für hohe Taktraten, da die Verdrahtung im Inneren des Cores nicht über die Verbindungsmatrix laufen muss.

[2] Teil der ISE Design Suite zum Entwurf von Embedded Systemen auf einer abstrakten Ebene.
[3] Simulink-Toolbox als Teil der ISE Design Suite zum Entwurf von FPGA-basierten Systemen auf einer abstrakten Ebene.

Xilinx ist nicht der einzige Hersteller von FPGAs, hält allerdings den größten Marktanteil. Andere Hersteller von FPGAs sind Altera, Actel, QuickLogic, SiliconBlue, Abound Logic, Atmel, Lattice, Aeroflex und Achronix angeboten. Abound Logic, SiliconBlue und QuickLogic bieten sehr stromsparende FPGAs an. Diejenigen von Aeroflex und Actel sind unter anderem auf Strahlungsresistenz optimiert. Deutlich schnellere FPGAs, im Gegensatz zu den anderen Herstellern, sind von Achronix Semiconductor erhältlich.

FPGAs sind durch ihre flexible Programmierbarkeit und der Möglichkeit Signale parallel zu verarbeiten, sehr für die digitale Signalverarbeitung wie in Software-Defined-Radios (SDR) geeignet. Eine Signalverarbeitung kann so auch nach der Implementierung im FPGA geändert werden und ermöglicht die Integration von mehreren parallelen Datenstromverarbeitungen in einem Chip.

2.2 Datenstrom und Echtzeit

Echtzeit (Realtime - RT) heißt kurz gesagt nur, dass die bei der Verarbeitungseinheit ankommenden Daten (Samples) mindestens so schnell verarbeitet werden müssen, wie diese übertragen werden. Ob dies nur ein Abtastwert (Sample) oder ein ganzer Block von Samples ist, hat für den Begriff Echtzeit erst einmal keine Bedeutung. Es ist nur eine Frage der Takt-basierten Verzögerung (Latenz). Als Beispiel soll ein Rundfunk-Audio-Datenstrom und ein Voice-Over-IP-Datenstrom dienen, die eine digitale Filterung erfahren und anschließend von einem Menschen gehört werden. Vorher muss jedoch geklärt, werden was es mit dem Begriff Datenstrom (Stream) auf sich hat.

Ein Datenstrom ist nichts weiter als eine Folge von zufälligen Samples. Jedoch haben diese Samples einen zeitlichen Bezug zueinander. Zum Beispiel hat ein Audiostream eine bestimmte Abtastrate, die zur Wiedergewinnung des analogen Audiosignals benötigt wird. Somit ist der Datenstrom auf diese Abtastrate fixiert. Des Weiteren ist ein Datenstrom von der Dauer nicht begrenzt. Bei dem Kopiervorgang einer Datei von einer Festplatte auf die andere werden einzelne Datenstücke dieser Datei über einen Bus geschickt. Dies geschieht zeitlich begrenzt und die Datenblöcke haben keine zeitliche Korrelation zu einander, zudem sind die Daten nicht zufällig. Dies wäre nach der obigen Definition somit kein Datenstrom.

Für ein Rundfunk-Signal macht es für den Hörer keinen Unterschied, ob der Radio-Moderator vor 1 ms oder 3 s gesprochen hat. Bei einem Telefongespräch über Voice-Over-IP macht es sehr wohl einen Unterschied, denn der Hörer wird mit großer Wahrscheinlichkeit antworten wollen, ohne dem Gesprächspartner ins Wort zu fallen. Demnach ist über die Verzögerung im jeweiligen Anwendungsfall nachzudenken, aber auf den Begriff Echtzeit hat dies keinen Einfluss.

Bei einem Datenstrom der in Blöcken von zum Beispiel 512 Byte und mit einer Übertragungsrate von 1 MHz übertragen wird, soll eine digitale Verarbeitung stattfinden. Am Eingang einer Verarbeitungseinheit werden 512 Byte gepuffert und erst verarbeitet, wenn alle Bytes angekommen sind. In der gleichen Zeit wird ein zweiter Puffer am Ausgang mit 512 bereits verarbeiteten Werten mit 1 MHz wieder kontinuierlich versendet versendet. Für die Verarbeitungseinheit bedeutet dies, sie muss einen 512 Byte großen Block in $\frac{512}{1\ MHz} = 512\ \mu s$ verarbeiten. Ist dies zu jedem Zeitpunkt gewährleistet, kann man dies System echtzeitfähig nennen, auch wenn eine Pufferung Latenz erzeugt.

Echtzeit ist nicht nur in der Datenstromverarbeitung präsent, sondern zum Beispiel auch in der Regelungstechnik. Hier kann ein Sensor einem Rechner über den Füllstand eines Wasserbeckens informieren, der dann nach einiger Zeit dem Aktor (vielleicht ein Ventil) sagt, dass es auf/zu drehen muss. In bestimmten Anwendungsbereichen kann so etwas sehr zeitkritisch sein und es kommt darauf an, dass das Sensor-/Aktor-Zusammenspiel in einer fest definierten Zeit geschieht. Wird diese Zeit zu jedem Zeitpunkt eingehalten ist auch dieses ein Echtzeit-System.

2.3 Bandpassunterabtastung

Bandpassunterabtastung ist eine Technik, um beim Prozess der Analog-Digital-Umsetzung implizit eine Mischung zu erreichen (digitale Mischung). Die folgenden Unterkapitel sollen den Begriff Bandpassunterabtastung weiter erläutern und zeigen unter welchen Randbedingungen die Bandpassunterabtastung eingesetzt werden kann.

Abtastung

Eine Analog-Digital-Umsetzung kann mit einem Analog-Digital-Umsetzer (ADC) erreicht werden. Ein ADC nimmt in festgelegten Zeitabständen (definiert durch die Abtastrate f_s) Proben des analogen Signals und weist diesen einen Zahlenwert zu. Der Zahlenwert wird zum Beispiel binär kodiert, wobei die Anzahl an Bits, die für die Kodierung genutzt werden, die Auflösung des ADCs angibt. Das heißt, um so mehr Bits, um so genauer wird das analoge Signal in der digitalen Domäne abgebildet. Der Ausgang des ADC liefert einen konstanten digitalen Datenstrom. Das heißt, bei einer Abtastrate von 80 MHz und einer Auflösung von 16 Bit wird dieser mit 160 MB/s übertragen.

Durch die Systemtheorie lässt sich nachweisen, dass das entstandene Ausgangsspektrum des digitalen Datenstroms die mit der Abtastfrequenz periodische Wiederholung des Eingangsspektrums ist. Das bedeutet, es ist möglich, dass sich die periodischen Wiederholungen überschneiden (Aliasing-Effekt). Nach dem Nyquist-Shannon-Theorem ist die Abtastfre-

quenz so zu wählen, dass keine Aliasing-Effekte im Spektrum auftauchen. Diese Frequenz muss mindestens doppelt so groß sein wie die größte Frequenz des Eingangsspektrums (Nyquist-Frequenz: $f_s > 2 \cdot f_1$). Wird diese Bedingung eingehalten, kann das analoge Eingangsspektrum nach einer Digital-Analog-Umsetzung mit Hilfe eines Tiefpasses wieder zurück gewonnen werden (Abbildung 2.2). Das ursprüngliche analoge Spektrum (orange) wird periodisch mit f_s fortgesetzt und es kommen neue Anteile, um vielfache von f_s hinzu (grün). Ein Tiefpass (lila) nach einem Digital-Analog-Umsetzer (DAC), zum Übergang in die analoge Domäne, kann nun dafür sorgen, das Ursprungsspektrum zurück zu gewinnen. Abbildung 2.3 zeigt eine Abtastung mit Aliasing-Fehler (rot). Aliasing führt so bei einer Rekonstruktion mit einem Tiefpass zu Verzerrungen im Vergleich zum Ursprungsspektrum.

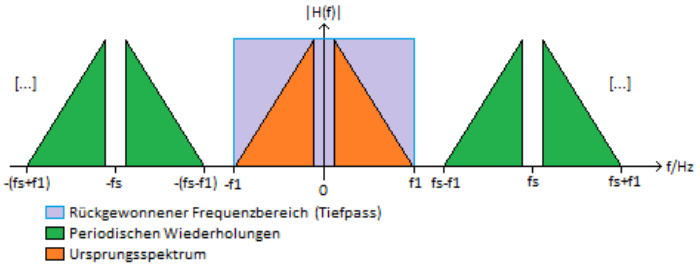

Abbildung 2.2: Abtastung im Spektrum (kein Aliasing)

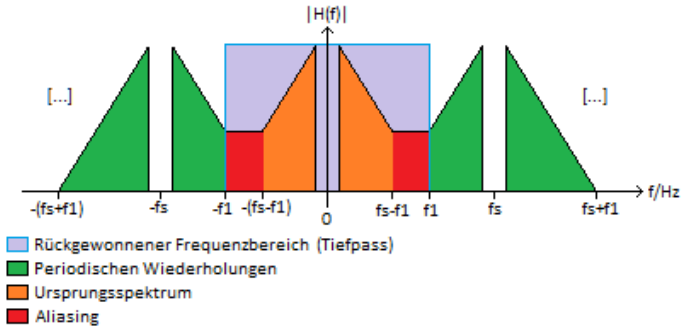

Abbildung 2.3: Abtastung im Spektrum (mit Aliasing)

Reale elektrische Signale sind nie ideal, das heißt es ist immer ein Rauschsignal additiv überlagert oder Signale in anderen Frequenzbändern sind zusätzlich präsent. Es ist somit vor der Abtastung mit einem analogen Tiefpass (Antialias-Filter) dafür zu sorgen, dass der Frequenzbereich größer $\frac{f_s}{2}$ hinreichend gedämpft ist und kein Aliasing-Effekt auftreten kann.

Wenn die Abtastfrequenz größer als die Nyquist-Frequenz gewählt wird, spricht man von Überabtastung. Wenn diese kleiner ist, spricht man demnach von der Unterabtastung. Eine

Überabtastung stellt rein prinzipiell kein Problem dar, außer dass der Datenstrom mehr Daten als nötig befördern muss.
Bei der Unterabtastung schaut dies schon anders aus, denn bei falscher Wahl der Abtastfrequenz kommt es zu den genannten Überlappungen. Wenn eine Unterabtastung bedacht durchgeführt wird, kann hieraus allerdings ein großer Vorteil gewonnen werden.

Bedachte Unterabtastung

Ein ADC mit einem Abtasttakt von 80 MHz darf nach dem Shannon-Nyquist-Theorem nur ein Signal bis 40 MHz in die digitale Domäne umgesetzt werden. Das würde bedeuten, um ein Frequenzband wie VHF-2 (87,5 MHz bis 108 MHz) mit diesem ADC umzusetzen, ist eine Abtastrate von mindestens 216 MHz nötig. Höhere Abtastraten erfordern jedoch teurere ADCs und leistungsstärkere Signalverarbeitungssysteme, da mehr Daten zu verarbeiten sind. Der VHF-2 Frequenzbereich könnte vor der Wandlung analog unter 40 MHz herunter mischen, doch dies würde, durch die individuelle Anpassung des analogen Mischers (Abgleich der analogen Schaltung), nicht zu der Philosophie des SDRs (flexible Anpassung) passen. Eine analoge Mischung kann jedoch durch den Vorteil einer bedachten Unterabtastung (Bandpassunterabtastung) umgangen werden.

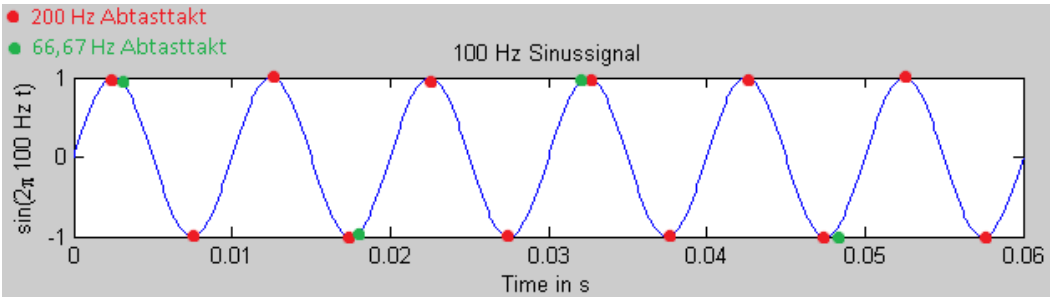

Abbildung 2.4: Abtastzeitpunkte bei Unterabtastung

Ein 100 Hz Sinussignal müsste laut dem Shannon-Nyquist-Theorem mit minimal 200 Hz abgetastet werden. Diese Abtastzeitpunkte könnten in den Maxima und Minima des Sinus liegen (1 und -1). Eine Abtastung mit 66,66 Hz würde allerdings das gleiche erzielen, jedoch mit einer Abtastfrequenz kleiner der Signalfrequenz (Abbildung 2.4). Zudem ist das 100 Hz Sinussignal von 100 Hz auf 33,34 Hz verschoben worden (digitale Mischung) ohne einen analogen Mischer zu nutzen. Eine geeignete Bandpassunterabtastung für ein Bandpasssignal von f_1 bis f_2 ($B = f_2 - f_1$) erreicht man durch die Einhaltung der folgenden Kriterien [30]:

- $f_{smin} = \frac{2 \cdot f_2}{m+1}$

- $f_{smax} = \frac{2 \cdot f_1}{m}$
- $m = floor(\frac{f_1}{f_2 - f_1})$

Anschaulich sagt m aus, wie oft mal die Bandbreite B zwischen 0 Hz und f_1 Platz findet ohne das sich diese überschneiden. Ist m eine gerade Zahl befindet sich der Frequenzbereich B in Regellage (negative und positive Frequenzen des Basisbandsignals von B sind original) im Ausgangssignalspektrum enthalten. Ist m jedoch ungerade, ist B in Kehrlage im Ausgangssignalspektrum, also der negative Frequenzbereich des Basisbandsignals vom Eingang, befindet sich im positiven Frequenzbereich und der positive im negativen des Ausgangssignals. f_{smin} und f_{smax} geben nun die gültigen Grenzen der Abtastrate an, bei welcher sich die periodischen Spektralbereiche gerade eben berühren würden, aber nicht Überlappen.

Das folgende Beispiel soll die Vorteile der Bandpassunterabtastung noch einmal verdeutlichen. Das VHF-2-Band (87,5 MHz bis 108 MHz) soll in einem digitalen System verarbeitet werden. Eine Abtastung nach Nyquist und Shannon würde einen Antialias-Tiefpass von 0 MHz bis 108 MHz und eine Abtastrate von $f_s = 2 \cdot 108\ MHz = 216\ MHz$ vorsehen. Bei einem 16 Bit ADC würden pro Abtastwert 2 Byte Daten anfallen. Das heißt ein Datenstrom von $v = 2 \cdot 216\ MHz = 432 \cdot 10^6\ Byte/s \approx 412\ MiB/s$ würde die Signalverarbeitungseinheit erreichen.
Mit einer geeigneten Unterabtastung müsste ein analoges Antialias-Bandpass den Bereich von $f_1 = 87,5\ MHz$ bis $f_2 = 108\ MHz$ selektieren. Die minimale Abtastfrequenz ergibt sich mit $m = floor(\frac{f_1}{f_2 - f_1}) = 4$ zu $f_{smin} = \frac{2 \cdot f_2}{5} = 43,2\ MHz$ und $f_{smax} = \frac{2 \cdot f_1}{4} = 43,75\ MHz$. Bei einer Festlegung zu $f_s = 43,5\ MHz$ würde sich das VHF-2-Band nun bei $abs(87,5\ MHz - floor(\frac{87,5\ MHz}{43,5\ MHz}) \cdot 43,5\ MHz) = 0,5\ MHz$ bis $abs(108\ MHz - floor(\frac{108\ MHz}{43,5\ MHz}) \cdot 43,5\ MHz) = 21\ MHz$ im digitalen Ausgangssignalspektrum befinden. Für die Signalverarbeitung heißt dies, dass die Informationen im Spektrum woanders liegen und nur noch ein Datenstrom von $v = 2\ Byte \cdot 43,5\ MHz = 47 \cdot 10^6\ Byte/s \approx 45\ MiB/s$ verarbeitet werden muss. Dies ist ein Rechenzeitgewinn von ca. 90% ohne einen analogen Mischer nutzen zu müssen. Jedoch befindet sich das Signal nun in Kehrlage, welches durch eine mehrfache Unterabtastung (m = 3) umgangen werden kann (Abbildung 2.5).

2.4 Direct Digital Synthesis

Für das Herab- oder Heruntermischen (Anwendung Kapitel 2.5) eines Digitalsignals wird eine sinusförmige Mischerfrequenz benötigt, was somit einen digitalen Signalgenerator voraussetzt. Direct-Digital-Synthesis (DDS) ist eine Technik, die diese Signalgeneration von periodischen Signalen erlaubt.

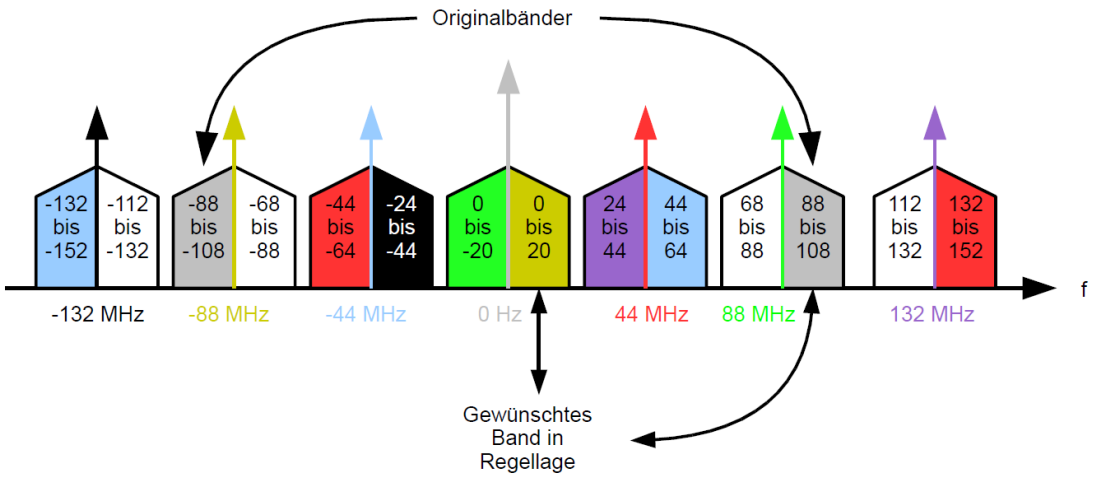

Abbildung 2.5: Bandpassunterabtastung von VHF-2 mit m=4. *Quelle: [30]*

VCOs sind analoge Bauelemente, welche ein sinusförmiges Ausgangssignal mit einer zur Eingangsspannung proportionalen Frequenz erzeugen. Die Bauteile werden zum Beispiel in Phase-Locked-Loop (PLL) Schaltungen eingesetzt, um eine Regelung der Phase eines Eingangssignals zu erreichen. NCOs sind in der digitalen Domäne das Pendant zu den VCO der analogen Domäne. Jedes digitale und in der Frequenz durch eine digitale Zahl steuerbares digital Signal, wird von einem NCO erzeugt. DDS ist eine spezielle Umsetzung eines NCO, bei der direkt aus dem Zählerstand eines periodischen Zählers (z.B. Modulo X) ein Ausgangssample generiert wird.

Prinzipiell besteht ein DDS-System aus einem Phasenakkumulator und einer Look-Up-Tabelle (LUT). Als Eingang des Systems dient ein konstanter Phaseninkrementwert und als Ausgang das gewünschte Signal. Siehe hierzu das Prinzip Schaltbild in Abbildung 2.6.

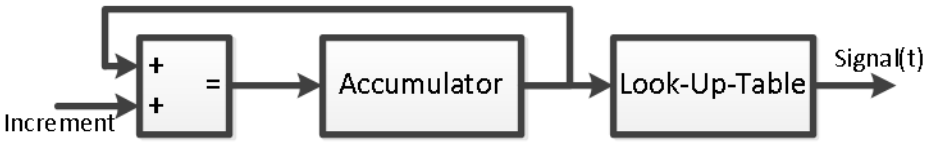

Abbildung 2.6: Struktur eines DDS-Systems

Nach einem Reset des DDS ist der Akkumulator mit dem Wert 0 initialisiert und es würde bei jedem folgenden Takt der Wert des Phaseninkrementwerts aufaddiert werden. Da die Akkumulator-Bitbreite begrenzt ist, wird der Akkumulator periodisch überlaufen und von unten wieder weiter zählen. Aus einem Teil des Akkumulatorwertes, als Adresse für die LUT, wird bei jedem Takt ein Element aus der LUT selektiert und erscheint am Ausgang. Da der Akkumulator periodisch Werte enthält, wird auch der Ausgang von der LUT periodisch

sein. Sollten Cosinus-Samples hinterlegt sein, erscheint eine digitale Cosinus-Zahlenfolge am Ausgang.

Es gibt drei wichtige Kennwerte die von Bedeutung sind. Dies sind die Bitbreite des Akkumulators, die Anzahl der LUT Elemente und die Bitbreite der Werte in der LUT. Die Bitbreite des Akkumulators legt fest, wie genau eine Periode des Signals erzeugt werden kann. Zum Beispiel würde mit einer Bitbreite von 32 Bit mit einem Systemtakt von 100 MHz eine Genauigkeit von $g = \frac{100\ MHz}{2^{32}} \approx 0,023\ Hz$ möglich sein. Für ein System mit einem X Bit breiten Datenstrom sollte eine LUT mit X Bit breiten Werten gewählt werden, damit diese kompatibel sind. Des weiteren gibt die Anzahl der LUT-Elemente vor, wie genau eine Schwingung dargestellt werden kann.

Als Beispiel soll eine Frequenz von 13 MHz bei einem 100 MHz System mit einem 32 Bit Phasenakkumulator am Ausgang erscheinen (wobei die Zahlen willkürlich gewählt wurden), könnte man wie folgt vorgehen. Die Frequenzgenauigkeit ergibt sich zu $g = \frac{100\ MHz}{2^{32}} \approx 0,023\ Hz$. Der Phaseninkrement zu $i = \frac{13\ MHz \cdot 2^{32}}{100\ MHz} = 558345748$, das heißt ein Überlauf des Akkumulators tritt nach ca. $\frac{2^{32}}{558345748} = \frac{100\ MHz}{13\ MHz} = 7,69$ Systemtakten auf, was impliziert, dass nur 7 Samples pro Periode erscheinen. Auch hier ist das Abtasttheorem einzuhalten, sodass minimal zwei Samples pro erzeugter Frequenzperiode erscheinen müssen.

2.5 Digital Down Conversion

Digital-Down-Converter (DDC) sind digitale Systeme, die ein hochfrequentes Eingangssignal digital im Spektrum verschieben und hierbei gleichzeitig eine Zerlegung in ihre orthogonalen Signalkomponenten erreichen (I und Q).

Wird zum Beispiel ein hochfrequentes Eingangssignal, moduliert auf 90,3 MHz, für eine Demodulation auf einer Zwischenfrequenz von 100 kHz erwartet, kann dies durch eine Multiplikation von dem Eingangssignal (S_X) und einem phasengleichen sinusförmigen (90,3 - 0,1 = 90,2) MHz Signal (S_Y), die Mischerfrequenz, geschehen. Denn aus der Trigonometrie ist bekannt, dass $S_Z = S_X \cdot S_X = \cos(X) \cdot \cos(Y) = \frac{1}{2}(\cos(X-Y) + \cos(X+Y))$ ist. Der erste Term erzeugt die Verschiebung des Eingangssignals von $90,3\ MHz - 90,2\ MHz = 100\ kHz$, der zweite die Verschiebung um den gleichen Abstand (90,2 MHz) nach rechts. Dieser Prozess wird auch als Mischen bezeichnet. Das herab gemischte Eingangsspektrum kann nun am Ausgang genutzt werden, wenn der hochfrequente Term durch einen Tiefpass eliminiert wird.
Wenn die Mischerfrequenz und das Trägersignal nicht genau in Phase liegen, führt das zu einer Dämpfung. Diese Dämpfung kann bis zu ∞ gehen, wenn die Phase 90° beträgt und das Ausgangssignal komplett auslöschen.
Um keine Mischung mit einer phasengleichen Mischerfrequenz zum Sender durchführen

zu müssen, kann die IQ-Demodulation ([29] und [34]) angewendet werden. Für die IQ-Demodulation existiert ein zweiter Mischerpfad der mit der um 90° verschobenen Cosinusschwingung mischt, um den orthogonalen Signalanteil des Eingangs zu erhalten (Quadratur-Anteil: Q). Der erste Mischer wird als Inphasenmischer bezeichnet und erhält somit seine Kurzbezeichnung I. I und Q werden auch als Quadratursignale bezeichnet.
Die Quadratursignale können zwei verschiedene Signale sein, die von einem Sender AM-moduliert wurden und orthogonal zusammengefasst worden sind. Die IQ-Modulationstechnik wird zum Beispiel für die Modulationsart QAM genutzt, bei der die komplexen Komponenten einer komplexen Zahl je in I und Q moduliert werden. Somit können komplexe Zahlen auf einem Träger übertragen werden, welche bei QAM in einer Komplexenebene als Zeiger dargestellt werden und je einem digitalen Symbol entsprechen. QAM wird zum Beispiel bei DVB-T oder auch UMTS eingesetzt, um digitale Signale zu übertragen.

Ein DDC besteht prinzipiell aus einem Mischer und einem Tiefpass je I bzw. Q Signalpfad (Abbildung 2.7). Die Tiefpässe sorgen für die Auslöschung der hochfrequenten Mischreste des jeweiligen Mischerpfades. Die Mischer werden von einem DDS-Bausein gespeist, der eine möglichst genaue Cosinus bzw. Sinus Schwingung erzeugt. Der const-Eingang gibt dem DDS einen Wert vor, der der Frequenz entspricht, um welche das gewünschte Band gemischt wird. Durch den Mischvorgang werden die Signalinformationen meist in eine niedriges Frequenzband verschoben, sodass die bestehende Abtastrate meist größer wie nötig ist. Aus diesem Grund wird eine Dezimierung eingefügt, die den ausgehenden Datenstrom auf ein Minimum verkleinert.

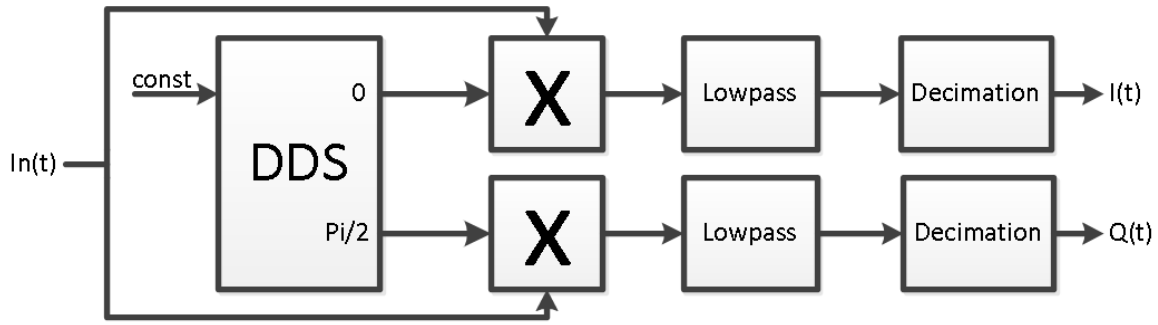

Abbildung 2.7: Struktur eines DDC

2.6 Software Defined Radio

Der Begriff SDR ist nicht neu. Bereits 1996 wurde das *Wireless Innovation Forum™* [41] ins Leben gerufen, um geschlossen an dem Thema SDR zu arbeiten. Deshalb existieren bereits zahlreiche Projekte/Hersteller, die sich mit diesem Thema befassen.

Aus der Bezeichnung Software-Defined-Radio (SDR) und der Einleitung dieser Arbeit kann bereits die wesentliche Bedeutung des SDR erkannt werden. Einfach gesagt ist es ein Funkempfänger bzw. -sender der flexibel umkonfigurierbar oder umprogrammierbar ist. Dies ermöglicht die schnelle und flexible Anpassung des Funksystems an neue oder geänderte Spezifikationen. Es bietet dem Hersteller eines Funksystems eine Kostenreduzierung, da Änderungen während des Entwicklungsprozesses, oder im Feld, schnell eingearbeitet werden können. Erreicht werden kann dies in der digitalen Domäne mit DSPs für sequentielle ablaufende Programme oder mit FPGAs zur flexiblen Implementierung von parallelen digital Logik-Strukturen. Eine frühe Analog-Digital-Umsetzung der Signale wird bei SDRs angestrebt, um die analoge Schaltung, und die damit verbundenen Abgleichsungenauigkeiten, klein zuhalten und das System besser steuerbar bzw. konfigurierbar zu machen.

Allgemeine SDR-Struktur

Für das erste Verständnis wird im Folgenden ein allgemeines SDR-System (Abbildung 2.8) für Bildungs- und Entwicklungszwecke gezeigt, welches als Grundlage für viele der bereits existierenden SDR-Systeme aus Kapitel 2.6 gilt (Funkempfang).

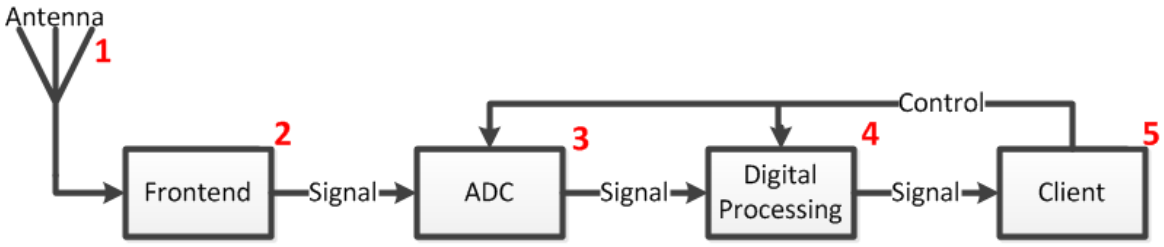

Abbildung 2.8: Allgemeine SDR-Struktur

1. Die Antenne empfängt die Funksignale und setzt diese in elektrische Signale um. Hiernach ist das Funkspektrum als elektronisches Signal der nächsten Verarbeitungsebene verfügbar. Mit welcher Qualität die jeweiligen Frequenzbereiche im Ausgangssignal enthalten sind, hängt von der verwendeten Antenne ab die das Empfangsverhalten definiert.
2. Dieser Abschnitt wird als Frontend bezeichnet und besteht aus analoger Elektronik. Aufgabe dieser Elektronik ist die Anpassung des empfangenen Signals von der Antenne an den ADC, um Aliasing zu unterdrücken und den ADC voll aus zu steuern (für volle Wandlungsqualität).
3. Das vom Frontend präparierte Ausgangssignal wird vom ADC in ein digitales Signal umgesetzt. Eine wichtige Eigenschaft des ADC sollte sein, eine möglichst große Band-

breite wandeln zu können, um so flexibel wie möglich zu bleiben. Auch eine dynamische Abtastrate sollte erlaubt sein, um eine Vorpositionierung des gewünschten Frequenzbandes vornehmen zu können oder die Verarbeitungsgeschwindigkeit des Nachfolgeblocks zu mindern.

4. Ein digitales System (zum Beispiel ein FPGA oder DSP) nimmt den digitalen Datenstrom entgegen und wendet eine Signalverarbeitung auf ihn an. Welche Signalverarbeitung dies ist und ob diese vom Block 4 oder Block 5 ausgeführt wird, sollte dem Nutzer des SDR überlassen werden. Eine echtzeitfähige Übertragungsstrecke zu einem Client würde diesem den verarbeiteten Datenstrom bereitstellen.

5. Als Client könnte ein PC angesehen werden. Dieser konfiguriert das SDR für den gewünschten Frequenzbereich oder implementiert eine Signalverarbeitung im Block 4 des SDR. Anschließend empfängt dieser die SDR-Daten und kann diese nutzen oder an ein weiteres Programm leiten.

Stand der Technik

Der folgende Abschnitt zeigt ausgewählte Projekte/Produkte, die in die genannte allgemeine SDR-Struktur passen. Es sind Projekte die für Bildungs- und Entwicklungszwecke, sowie aber auch im kommerziellen Raum genutzt werden. Grob kann eine Gliederung in zwei Bereiche beobachtet werden. Transceiver und PC-Software. Der Transceiver übernimmt die Aufgaben der Blöcke 1, 2, 3 und 4. Zudem ist in diesem ein Sendepfad implementiert, welcher in dieser Arbeit nicht berücksichtigt wird. Der Block 5 kann der PC-Software zugeordnet werden.

GNU Radio [9]

GNU Radio ist ein freies Open Source Entwicklungs-Kit für SDR-basierte Signalverarbeitung auf dem PC, welches zur Zeit hauptsächlich für Entwicklungs- und Bildungszwecke zum Einsatz kommt. Es wird in Verbindung mit Radio-Frequency(RF)-Transceivern genutzt, wie dem USRP (s.u.) oder auch dem SDR4ALL (s.u.), um Signalverarbeitung auf dem PC zu betreiben. Zudem bietet das GNU Radio eine Schnittstelle zu Comedi-kompatibler Hardware (s.u.). Auf Hardware-Transceiver ist das GNU Radio allerdings nicht zwingend angewiesen, denn es liefert zum Beispiel auch Generatoren in Software, um erfolgreich arbeiten zu können. Entwickelt wurde die Software ursprünglich für Linux-Maschinen, dennoch ist sie in Verbindung mit Cygwin[4] auch auf Windows-PC's nutzbar.

[4] Freies Softwarepaket für Windows-basierte PCs, welches diesem eine Linux-kompatibles API bereitstellt. http://www.cygwin.com

Comedi [4]

Für eine Kommunikation von Software mit den Datenerfassungsgeräten (DAQ) bietet das Comedi-Projekt eine ganze Reihe an Treibern, die für den Linux-Kernel verfügbar sind. Diese Treiber basieren auf Kerneltreibern und sind somit sehr gut für Echtzeit-Anwendungen geeignet. GNU Radio bietet eine Schnittstelle zu diesen Treibern, sodass GNU Radio mit jedem Comedi-kompatiblen Transceiver arbeiten kann. Es gibt weitere zahlreiche Applikationen die auf den Comedi-Treibern basieren. Dies ist zum Beispiel comedirecord, um Daten aus dem DAQ zu loggen.

Winrad [40]

Winrad ist eine Windows Software die von Jeffrey Pawlan entwickelt wurde und genutzt werden kann, um digitale Daten eines SDR-Transceivers zu analysieren und darzustellen. Es handelt sich um eine freie Software und ist daher frei herunterladbar. Treiber für kompatible Transceiver sind durch Dynamic-Link-Library[5] (DLL) Dateien gegeben, sodass auch für eigene Transceiver eine einfache Schnittstelle programmiert werden kann. Zur Zeit existieren unter anderem Schnittstellen DLL für Perseus (s.u.) und einige RFspace Transceiver (s.u.).

MATLAB/Simulink [18]

MATLAB ist eine kommerzielle Mathematiksoftware der Firma Mathworks mit der es möglich ist, numerische Lösungen für mathematische Probleme zu erstellen und diese entsprechend darzustellen. Man kann in einer Konsole arbeiten oder komfortable Skripte schreiben. Weiter bietet MATLAB die Möglichkeit, Anwendungen nativ zu kompilieren. Zahlreiche Toolboxen (Erweiterungen) werden verwendet, um die Funktionalität zu erweitern. Zu MATLAB gehört eine Software Namens Simulink, welche für eine grafische Simulation eines Problems/Systems genutzt werden kann. Auch für Simulink sind die erwähnten Toolboxen verwendbar.

Einige SDR-Transceiver Hersteller/Projekte bieten solche Toolboxen an, um mit deren Geräten komfortabel zu kommunizieren (siehe SDR4ALL auf Seite 26). Weitere Toolboxen von Mathworks bieten zahlreiche Schnittstellen zu anderen Medien, wie zum Beispiel TCP/IP oder aber ein eigenes Blockset für den USRP-Tranceiver (s.u.).

MATLAB-Skripte sind nicht Multithreading fähig und werden nur interpretiert ausgeführt. Um dies zu umgehen, gibt es MEX-Dateien. Diese beinhalten schnellen nativen Maschinencode

[5] Dateien die kompilierten Maschinencode enthalten, um diesen dynamisch zwischen Programmen teilen zu können.

zur direkten Ausführung durch MATLAB. MEX-Dateien können in Verbindung mit einer MEX-Bibliothek in der Programmiersprache C/C++ erzeugt werden. Eine Generation von nativen Code aus MATLAB-Skripten und Simulink-Modellen ist mit dem Embedded Coder von Mathworks möglich, sodass eine Rechenzeit-optimierte Ausführung möglich ist.

LabView [22]

LabView ist eine kommerzielle grafische Entwicklungsumgebung. Ein Programm wird nicht durch einen Quellcode wie C erstellt, sondern durch das Positionieren, Verbinden und Verschachteln von grafischen LabView-Blöcken. LabView bietet zu seinen Blöcken die Programmieransicht und eine GUI-Ansicht (VI), die den jeweiligen LabView-Block abstrakt von außen darstellt und eine Bedienung zu lässt. Genau wie auch bei MATLAB besteht die Möglichkeit das Programm zu erweitern oder es nativ zu kompilieren. Das SDR Hardware-Projekt USRP ist bereits dabei, einen Treiber für seine Hardware zu schreiben, damit dieses kompatibel mit LabView wird. Werden Übertragungsmedien wie TCP/IP genutzt, kann eine DAQ-Kommunikation auch direkt ohne Treiber möglich sein.

SDR4ALL [24]

SDR4ALL ist ein freies Open Source SDR-Projekt, welches die allgemeine SDR-Struktur abdeckt, also von Transceiver bis hin zur Auswertungssoftware auf dem PC. Es wurde von Mitarbeitern der Firma Supélec[28] ins Leben gerufen, um eine einfache und freie Struktur zu schaffen, welche für die Ausbildung und Entwicklung genutzt werden kann. Zur Zeit ist der Transceiver noch in der Entwicklung, dennoch kann das Software-Toolkit bereits mit dem URSP-Transceiver (s.u.) genutzt werden. Der Transceiver soll dafür ausgelegt werden, im Industrial-Scientific-and-Medical (ISM) Band von 2,4 GHz bis 2,5 GHz zu arbeiten.

Das Software-Packet von SDR4ALL bietet ein komplettes Backend für MATLAB in der Form einer Toolbox. Des Weiteren werden DLL's gestellt, mit denen eine SDR-Kommunikation mit einer eigenen Software realisiert werden kann. Als SDR-Backend kommt ein Serverprogramm zum Einsatz, das MATLAB oder auch anderen Clients eine einfache Schnittstelle über einen Socket bietet.

Perseus [21][2][3][23]

Die Italienische Firma microtelecom[20] bietet mit dem Perseus ein modernes SDR-Komplettpaket an. Der analoge Hardwareteil beschränkt sich beim Perseus Projekt auf eine Pegelanpassung aus schaltbaren Dämpfungsgliedern von 0 bis 30 dB (Intervall: 10 dB),

einen Bandfilter für eine Vorselektion des Frequenzbandes, einem Tiefpass der als Antialiasing Filter eingesetzt werden kann und einem Verstärker zur Anpassung des Signals an den ADC mit nachgeschalteter Rauschunterdrückung. Der Bandfilter ist digital wählbar. Das heißt es ist möglich, mit Relais einen von zehn Bandfiltern (aus dem Bereich von 0 MHz bis 30 MHz) zu wählen oder den Bandpass komplett zu umgehen. Es sind somit Eingangsspektren bis zur Tiefpassfrequenz (30 MHz) nutzbar. Nach der Digitalisierung wird die DDC-Technik zur IQ-Mischung des Signals eingesetzt. Eine Übertragung über USB 2.0 an den PC ermöglicht diesem den Datenstrom zu empfangen und zu verarbeiten. Die Softwarelösung zur Analyse des Datenstroms ist eine Eigenentwicklung speziell für Perseus und erlaubt es eine Analyse des Datenstroms vorzunehmen. Es sind allerdings auch Treiber für Winrad verfügbar, die das Arbeiten mit Perseus und Winrad erlauben.

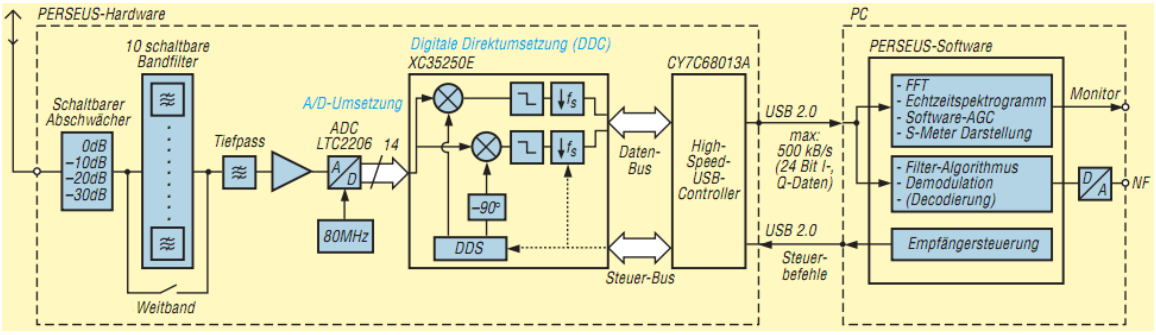

Abbildung 2.9: microtelecom Perseus SDR Lösung. *Quelle: [2]*

Quicksilver und SDRMAX [26]

SRL QuickSilver QS1R ist ein DDC-basierter SDR-Transceiver und gehört zu einem Open Source Komplettpaket von Software Radio Raboratory LLC (SRL) [26]. Er beinhaltet einen 16-Bit ADC von Linear Technology (LT2208) mit einer Samplerate von 125 MSPS. Ein Cyclon-FPGA von Altera hinter dem ADC beinhaltet bereits einen vorkonfigurierten DDC zur IQ-Mischung. Der FPGA ist von außen komplett konfigurierbar und auch umprogrammierbar, sodass eine Signalverarbeitung direkt in dem FPGA mit VHDL realisiert werden kann. Ein Audio-Ausgang, wie auch digitale Schnittstellen (I2C und SPI), können als Ausgang zu einem Client genutzt werden. Eine Verbindung zum PC ist durch USB 2.0 möglich, welche in Verbindung mit der Analyse-Software SDRMAX einfach für Bildungs- und Entwicklungszwecke genutzt werden kann.

SDRMAX abstrahiert das SDR über ein lokales Serverprogramm, sodass mehrere Betriebssystemprozesse Zugriff auf das SDR haben. Hiermit ist es auch für eigene Programme möglich, mit der SDR Hardware zu kommunizieren. Als Alternative zu SDRMAX wird auch Winrad unterstützt.

Am Eingang des Transceivers ist bereits ein 55 MHz Tiefpassfilter untergebracht worden, um den Alias-Effekt zu vermeiden. Allerdings kann auch die Unterabtastung genutzt werden, um Signale bis zu einer Frequenz von 500 MHz verarbeiten zu können. Hierfür ist ein Direkteingang geschaffen worden, an den ein eigenes analoges Frontend angeschlossen werden kann, sodass keine Filterung des Eingangssignals vom Transceiver ausgeführt wird.

Das Quicksilver-Projekt selbst sieht sich als verbesserte Version des Perseus-Projektes, da es flexibler genutzt werden kann (Vergleich siehe [25]). Dieses schließt unter anderem ein, ein eigenes Frontend zu nutzen, um auch mit der Unterabtastung arbeiten zu können. Zudem ermöglicht es im Gegensatz zu Perseus eine Integration von Signalverarbeitung in den FPGA, um so eine schnellere Signalverarbeitung, im Gegensatz zu einer langsamen MATLAB-Verarbeitung auf dem PC, zu gewährleisten.

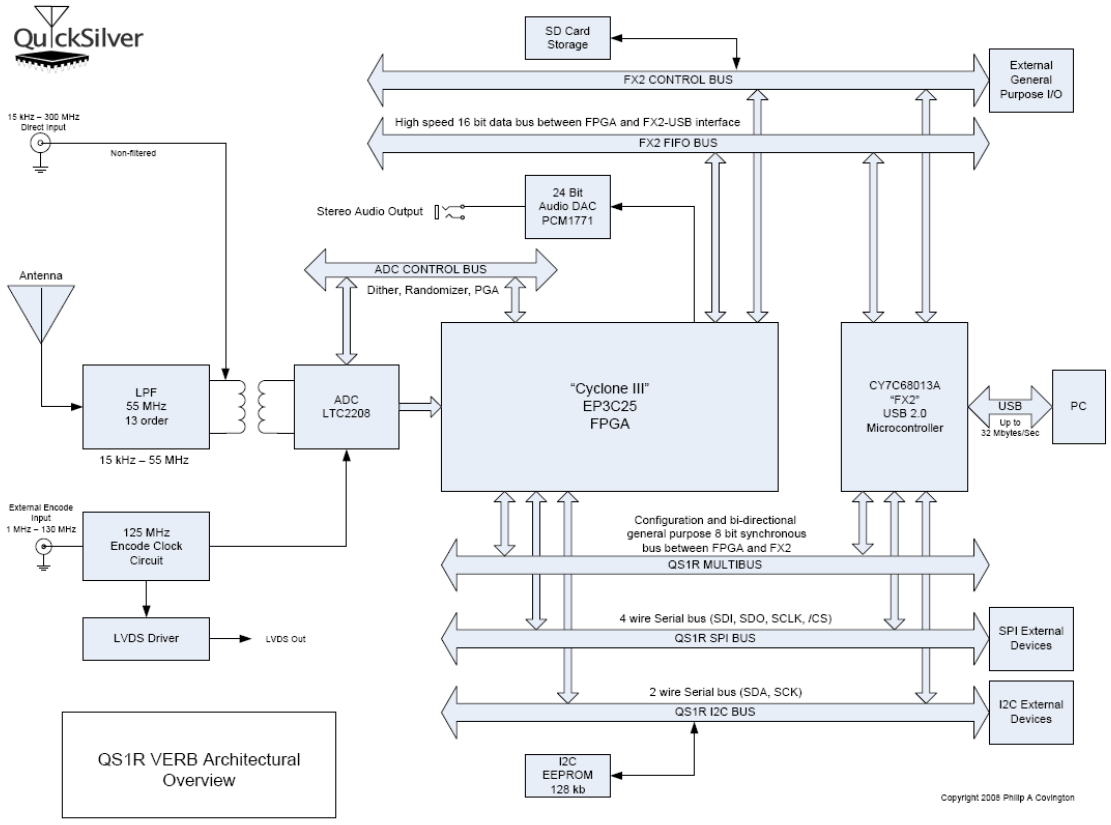

Abbildung 2.10: SRL QuickSilver QS1R Lösung. *Quelle: [26]*

URSP [8]

URSP (Universal Software Radio Peripheral) ist eine Open Source SDR-Transceiver-Lösung von Ettus Research LLC [8] und kann in Kombination mit dem GNU-Radio, LabView oder auch MATLAB/Simulink genutzt werden. Wenn es mit dem GNU Radio genutzt wird, ist dies eine komplette Open Source Lösung, für die jegliche Quellcodes und Schaltpläne zur Verfügung stehen. Ein extra Treiber für alle gängigen Betriebssysteme (Universal Hardware Driver) kann genutzt werden, um die SDR-Ressource direkt mit eigener Software zu nutzen.

Es gibt zahlreiche Versionen des Transceivers und ebenso zahlreiche Zusatzkarten, die für die analoge Vorverarbeitung für das gewünschte Frequenzband sorgen. Im USRP lassen sich diese Zusatzkarten vor dem ADC/DAC anbringen. Der durch den DDC umgesetzte Datenstrom wird über USB 2.0 einem PC bereitgestellt. Der USRP2 erweitert den USRP unter anderem durch eine Ethernet-Schnittstelle die es erlaubt, einen 50 MHz Datenstrom an einen Client zu versenden bzw. empfangen. Des Weiteren ist auf dem FPGA noch Platz, um eine eigene Signalverarbeitung zu realisieren und so die Last vom PC zu nehmen. Genauere Informationen hängen sehr von dem gewähltem Transceiver ab und sind den jeweiligen Datenblättern zu entnehmen.

Cool USB Radio [5]

Etwas überschaubarer gestaltet sich das Cool USB Radio Projekt. Enthalten ist die Transceiver-Hardware und ein Software Development Kit (SDK), sowie Beispielanwendungen, mit denen ein einfacher Einstieg gewährleistet wird. Den Analogeingang des Transceivers bildet ein konfigurierbarer Bandfilter, nach dem das Signal für den Altera Cyclone FPGA über einen Analog Device AD9215 digitalisiert wird. Die Übertragung zum PC ist auch hier über USB 2.0 realisiert, was zudem eine Möglichkeit bietet, eigene Software in den USB-Controller laden zu können. Auch das Senden ist möglich, wofür ein AD9742 DAC mit nachfolgendem analogen Bandpass genutzt wird. Laut dem Datenblatt soll eine Unterabtastung möglich sein, wenn das Eingangsfilter dementsprechend ausgelegt wird.

Sonstige

Es gibt zahlreiche andere Projekt/Produkte, die sich mit dem Thema SDR befassen, softwareseitig sowie auch hardwareseitig. All diese zu erwähnen würde den Rahmen sprengen, sodass einige folgende Stichworte einen Verweis auf weiterführende Informationen geben: FlexRadio, RFspace, Softrock, LinRadio ,WinRadio, Linrad, Genesis Radio, SDRadio, Rocky, Spectraview, PowerSDR, ...

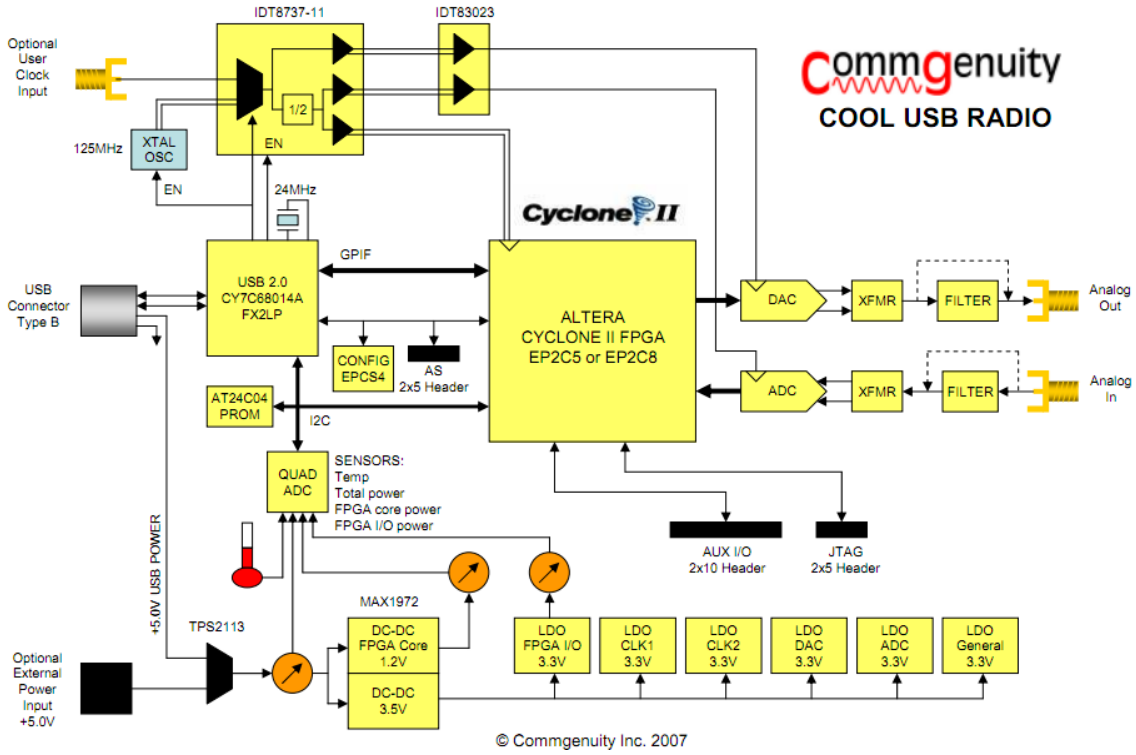

Abbildung 2.11: Cool USB Radio. *Quelle: [5]*

2.7 Very High Frequency II

Mit Very-High-Frequency II (VHF-2) wird der, bei der Bundesnetzagentur registrierte, Frequenzbereich für FM-Rundfunkradio bezeichnet (87,5 MHz bis 108 MHz). Für diese Arbeit wird dieser genutzt werden, um ein Rundfunkradio als Demonstrator zu realisieren. Somit soll dieses Kapitel zeigen wie sich VHF-2 zusammensetzt und welche Informationen ein FM-Rundfunksender tragen kann.

Jeder Rundfunksender ist einem Ausschnitt aus dem VHF-2 Bereich zugeordnet und zentriert sich um dessen Mittenfrequenz (Kanäle). Die Mittenfrequenz gibt die jedem bekannte Frequenz eines Senders an, zum Beispiel für NDR 2 ist es 87,6 MHz in Hamburg. Die verschiedenen Kanäle im VHF-2 Band sollen in einem 300 kHz Raster angelegt sein, laut [37] ist dieses in der Praxis jedoch nicht so. Es wird eher versucht, einen Abstand von 100 kHz bis 200 kHz zwischen den einzelnen Sendern einzuhalten. In Abbildung 2.12 wurde mit einer DVB-T Antenne und einem Spektrumanalysator der VHF-2 Frequenzbereich gemessen. Die Spitzen in dem Bereich von 87,5 MHz bis 108 MHz stellen die einzelnen frequenzmodulierten Sender dar. Der Sender NDR 90.3 ist zum Beispiel bei der markierten Stelle auf 90,3 MHz zu finden.

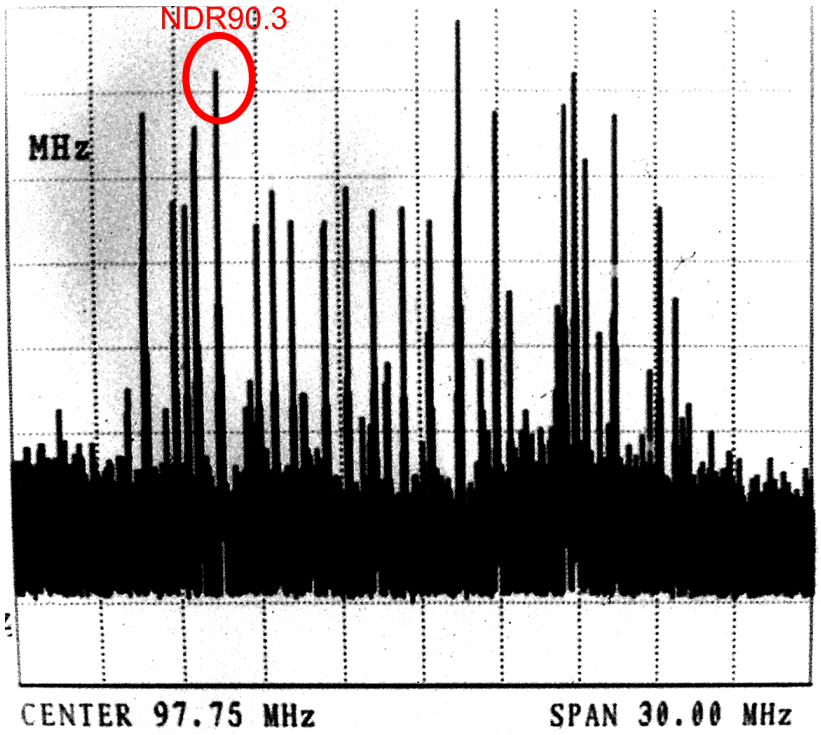

Abbildung 2.12: Gemessenes Spektrum des VHF-2 Bereiches am Standort Hamburg

Sollte die Mittenfrequenz eines nicht frequenzmodulierten Senders auf 0 Hz (Sender ist im Basisband) liegen, besteht das Frequenzspektrum aus weit mehr als nur Audiosignalen (Abbildung 2.13). Es beinhaltet unter anderen einen Bereich mit digitalen Radio-Data-System (RDS) bzw. Radio-Broadcasting-Data-System (RBDS) Signalen, um Informationen zu einem gerade spielenden Lied oder auch Verkehrsmeldungen für Navigationsgeräte (TMC - Traffic Message Channel) zu übermitteln. RBDS ist der offizielle Name in den USA und unterscheidet sich vom RDS-Standard nur sehr gering [35]. Weiter können auf den 67 kHz und 92 kHz Unterträgern (Subcarrier) weitere analoge oder digitale Signale übertragen werden. Welche dies sind, hängt vom jeweiligen Land ab. In Nord Amerika wird auf dem 67 kHz Subcarrier DirectBand übertragen. DirectBand ist ein Protokoll von Microsoft, um zum Beispiel GPS-Geräte oder Uhren mit Daten zu versorgen [31].

Im Basisband des Senders liegt ein Teil des eigentlichen 15 kHz breiten Audiosignals. Um auch Audio in Stereo empfangen ([32]) zu können und älteren Empfänger das Nutzen von Mono-Radio weiter hin zu ermöglichen, ist dieser Bereich (0 kHz bis 15 kHz) eine Addition aus linkem und rechtem Stereokanal (L+R) des ursprünglichen Stereosignals. Mono-Empfänger können diesen Bereich einfach herausfiltern und direkt auf einen Lautsprecher geben. Um die Stereosignale wiederzugewinnen ist ein wenig mehr Aufwand nötig. Ein möglichst reiner Pilotton (Sinusförmig) bei 19 kHz kennzeichnet die Hälfte der Mittenfrequenz, auf

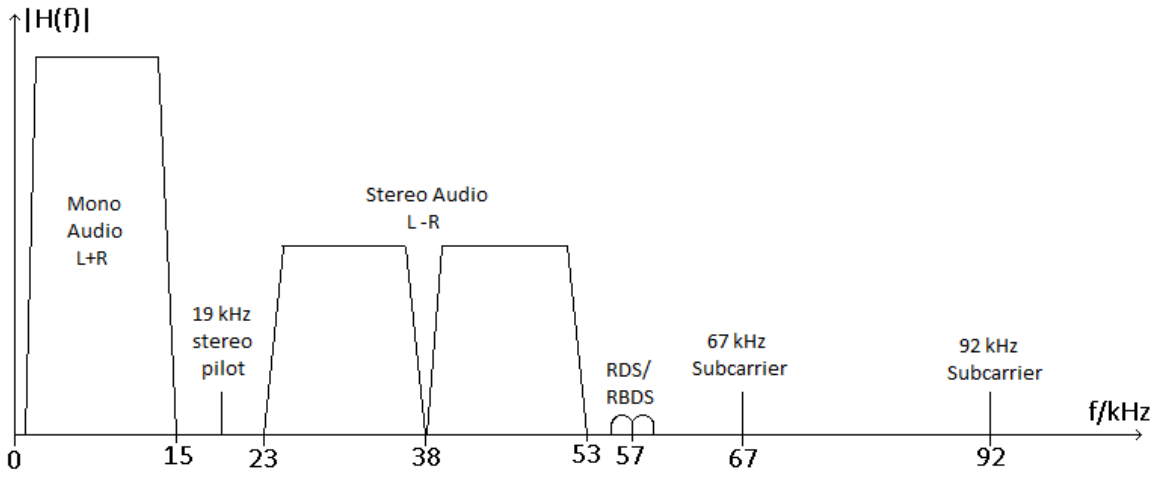

Abbildung 2.13: Frequenzband eines FM-Rundfunksenders. *Quelle: Wikipedia*

welcher die restlichen Teile des Stereosignals liegen. Dies ist das hoch gemischte Spektrum der Differenz aus linkem und rechtem Stereokanal (L-R). Der Pilotton kann genutzt werden, um die Mittenfrequenz zu erzeugen und durch Mischen an den restlichen Stereoanteil (L-R) im Basisband zu kommen. Aus den Basisbandsignalen L+R und L-R können nun die Stereosignale dekodiert werden. (L+R)+(L-R) ergibt den linken Stereokanal und (L+R)-(L-R) den rechten [32].

Die jeweilige Sendestation eines Senders setzt das Frequenzspektrum aus Abbildung 2.13 im Basisband durch digitale Modulation (für RDS, etc.) und Mischen (AM-Modulation) zum Positionieren der Kanalanteile zusammen. Anschließend wird der gesamte Frequenzbereich des Senders FM-moduliert und an die angestammte Stelle im VHF-2 Frequenzband verschoben.

3 Analyse und Konzept

Dieser Abschnitt der Arbeit soll einige Themen analysieren die wichtig für die Kozeptionierung des zu realisierenden SDRs sind. Dies ist zum Beispiel der bestehende DDC-Core und dessen Testumgebung. Oder die zu verwendende Übertragungsart. Zusätzlich soll in diesem Kapitel geklärt werden, wie sich das zu realisierende SDR-System zusammensetzt und dieses funktioniert.

3.1 Bestehendes System

Als Grundlage für diese Arbeit dient die Masterarbeit [29]. Es wurde der DDC-Core und ein zugehöriges Testsystem entwickelt, welches im Folgenden kurz erläutert und im Bezug auf diese Arbeit bewertet wird.

3.1.1 Aufbau und Funktion

Durch [29] wurde ein IP-Core für PLB-basierte Embedded Systeme entwickelt (dem DDC-Core), mit dem es möglich ist, ein hochfrequentes digitales Eingangssignal, nach dem Direct-Down-Conversion (DDC)-Prinzip, in seine orthogonalen Komponenten (I und Q) zu zerlegen und dabei diese herab zu mischen.
Intern arbeitet der DDC-Core mit einem DDS-Core zur Erzeugung der Mischerfrequenz und den eigentlichen Mischern für die Erzeugung der In-Phase (I) und Quadrature-Phase (Q) Datenströme (siehe Kapitel 2.5). Je I und Q Pfad ist eine Tiefpassfilter-Kaskade mit einer festen Grenzfrequenz von 200 kHz nachgeschaltet, um die hochfrequenten Reste des Mischvorgangs zu entfernen und gleichzeitig eine Dezimierung des Datenstroms von 64 zu implementieren. Eine Dezimierung von 64 führt am Ausgang des DDC zu einer Abtastrate von $f_{sa} = \frac{80\ MHz}{64} = 1,25\ MHz$, welches bei der festgelegten Tiefpassfrequenz deutlich dem Shannon-Nyquist-Abtasttheorem genügt.

Zur AD-Wandlung der Signale wird ein LTC2206 ADC (16 Bit @ 80 MHz ; [12]) von Linear Technology eingesetzt. Dieser verfügt über ein sehr breites Wandelspektrum (bis 700 MHz). Somit ist es möglich, auch ein 700 MHz Signal, durch 8-fache Unterabtastung und einem

Abtasttakt von 80 MHz, ein digitales Mischen auf $(700\ MHz - floor(\frac{700\ MHz}{80\ MHz}) \cdot 80\ MHz - 80\ MHz) = -20\ MHz$ zu erreichen. Die Schnittstelle zum ADC ist fest in den DDC-Core integriert, sodass dieser nur mit dem LTC2206 zu nutzen ist. Nach der IQ-Demodulation und der Herabmischung, werden die beiden Datenströme (I und Q) über einen 4 Bit tiefen und 32 Bit breiten FIFO und dem Native-Port-Interface (NPI) direkt im RAM abgelegt. Mit dem 16 Bit Samples einer Abtastrate von 80 MHz und einer Dezimierung von 64 muss der FIFO somit eine Übertragungsrate von $f_{sa} \cdot 2\ Samples \cdot 2\ Byte = 1,25\ MHz \cdot 4\ Byte = 5\ \frac{MB}{s}$ befördern.

Die Konfiguration des DDC-Cores kann Memory-Mapped von einem mit PLB angebunden Prozessor erfolgen. Es ist möglich, den DDC-Core zu starten, zu stoppen, die Mischerfrequenz einzustellen und den RAM-Bereich zur Speicherung der Daten festzulegen.

Der vom DDC zu nutzenden RAM-Bereich wird über eine 32-Bit-Speicheradresse als Startadresse und einen Chunkzähler (je Chunk 512 Byte) als Kapazität definiert. Der Chunkzähler definiert die Kapazität des RAMs, der für den DDC-Core als Datenspeicher dient. Der Core nutzt den definierten Speicher als Ringspeicher, das heißt, ist der Core an der obersten Adresse des zugewiesen Speichers angekommen, schreibt dieser unten weiter. Eine Konfiguration der Tiefpassfilter-Kaskade oder der Dezimierung ist nach der Synthese nicht mehr möglich.

Damit eine Verfolgung des genutzten Speichers, beziehungsweise zuletzt geschriebenen Speichers möglich ist, wird vom Core eine Interruptleitung getriggert, sobald 32 KiB Daten geschrieben worden sind. Die Firmware die diesen Core nutzt, kann diese Interrupts mitzählen und so errechnen, welcher Speicher als nächstes von der Software ausgelesen werden muss.

Für den weiteren Verlauf wird der DDC-Core als gegeben angenommen, sodass die Masterarbeit [29] als Datenblatt für den Core genutzt werden kann.

Zum Test des Systems wurde durch [29] eine USB-Übertragung eingerichtet (Abbildung 3.1), die auf Basis eines USB-Massenspeichers funktioniert. Das heißt, wird der FPGA über USB an den PC angeschlossen, wird dieser als USB-Masserspeicher erkannt. Im Massenspeicher erscheint eine Datei, die als Shared Memory zwischen FPGA und PC fungiert. Diese hat eine feste Größe von 20 MB + 512 Byte. Die 20 MB sind aufgeteilt in vier Segmente je 5 MB, dies entspricht bei einer Abtastrate von 80 MHz und einer Dezimierung von 64 genau einer Sekunde. Die ersten 512 Byte definieren den Control-Block. Hierüber wird eine Steuerung des DDC-Cores erreicht. Im Control-Block ist neben der Start- und Stop-Möglichkeit ein Segmentzähler angesiedelt, der vom FPGA durch Zählen der Interrupts erzeugt wird. Der PC kann diesen nutzen, um zu erkennen, welcher der vier Segmente gelesen werden muss. Diese Datei im Massenspeicher ist als FAT-Datei realisiert. Das heißt, ein Zugriff ist nur chunkweise (512 Byte) möglich und bedeutet, um den 32-Bit Fragmentzähler auslesen zu können, müssen immer die gesamten 512 Byte des Control-Blocks gelesen werden.

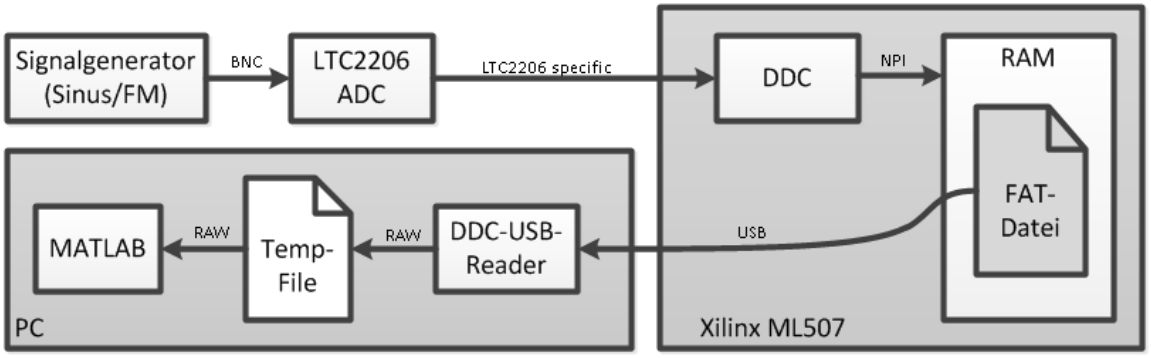

Abbildung 3.1: Überblick des bestehenden Testsystems ([29])

Der Zugriff vom PC geschieht über ein C++-Programm, das aus der Kommandozeile bedient werden muss. Ist dieses gestartet worden, startet es den DDC-Core über den Control-Block und fragt diesen jede 100 ms ab (100 ms polling). Hierüber stellt das Programm fest, wo Daten aus dem Shared-Memory gelesen werden müssen und führt dieses aus. Es speichert die zeitbegrenzten Daten in eine temporäre Datei, welche dann durch ein MATLAB-Skript wieder eingelesen werden.

Ein Test des Systems erfolgte mit einem Signalgenerator, mit dem ein reiner oder ein FM-modulierter Sinus auf das System gegeben wurde. Ein MATLAB-Skript kann genutzt werden, um Daten von der Größe eines definierten FFT-Fensters auszulesen und in der Zeit- und Frequenzdomäne darzustellen.

3.1.2 Bewertung der bestehenden Lösung

Für das in dieser Arbeit zu realisierende SDR ist das bestehende Testsystem unter Berücksichtigung der folgenden Punkte einsetzbar:

- Die Zwischenspeicherung der empfangenen Daten in eine temporäre Datei, das 100 ms polling des 512 Byte großen Control-Blocks und die Auslegung als FAT-Datei ist stark CPU-lastig.
- Durch die USB-Massenspeicherlösung ist das bestehende System zwar plattformunabhängig, bietet jedoch kaum Potenzial schnellere Datenströme zu führen (maximal $10\ \frac{MB}{s}$). Somit ist es schwierig mit dieser Lösung eine Erweiterbarkeit des zu realisierenden Systems, um weitere, parallele Datenströme, zu erreichen.
- Bis jetzt ist das bestehende System nur mit Testsignalen aus dem Generator gespeist worden und nicht mit einem realen Funksignal einer Antenne.

- Ob die realisierte Übertragung überhaupt die richtigen Daten empfängt, konnte mit dem bestehenden MATLAB-Skript aus [29] nicht festgestellt werden. Grund ist, dass nur so viele Daten gelesen wurden, wie in ein FFT-Fenster (vordefiniert ist maximal 327680 Samples ; $\approx \frac{327680}{1{,}25\ MHz} = 262\ ms$) passen. Das bedeutet, dass nie Daten über ein Segment (1 Sekunde) hinaus gelesen wurden und somit eine Verifikation unmöglich ist.
- Die Tiefpassfilter-Kaskade ist fest auf eine 200 kHz Grenzfrequenz dimensioniert und somit nicht flexibel einstellbar, um zum Beispiel Signale mit einer Bandbreite größer als 200 kHz zu empfangen.
- Bedingt durch die feste Tiefpassfilter-Kaskade ist die Dezimierung fest zu 64 definiert. Hiermit ist eine dynamische Reduzierung des Ausgangsdatenstroms nicht möglich und erfordert weitere Schritte außerhalb des Cores.
- Durch die spezielle NPI Anbindung an das RAM, ist der MPMC Speicher-Controller zwingend erforderlich.
- Der DDC-Core ist nur mit einem LTC2206 ADCs, oder zu diesem kompatible, nutzbar.

Damit sichergestellt ist, dass der DDC-Core auch in Verbindung mit dem bestehenden Testsystem funktioniert, wird zu Beginn der Realisierung dieser Arbeit ein initialer Test durchgeführt werden.

3.2 SDR-Struktur

Ein SDR-System für den Zweck dieser Arbeit könnte sich grob in vier Teile (Abbildung 3.2) gliedern lassen. Dies ist zum Einen das analoge Frontend, um die gewünschten Funksignale (VHF-2) für eine erfolgreiche Digitalisierung mit dem ADC vorzubereiten (Aliasing-frei bei Unterabtastung und Fullscale-Aussteuerung für gute Signalqualität). Zum Anderen die Konvertierung (Frequenzverschiebung) der Informationen des Funksignals in das Basisband, oder auf eine Zwischenfrequenz, durch den DDC-Core. Weiter wird der digitale Datenstrom über ein Übertragungsmedium an den PC geleitet. Zuletzt folgt das Auslesen und die Verarbeitung der Signale mit dem PC.

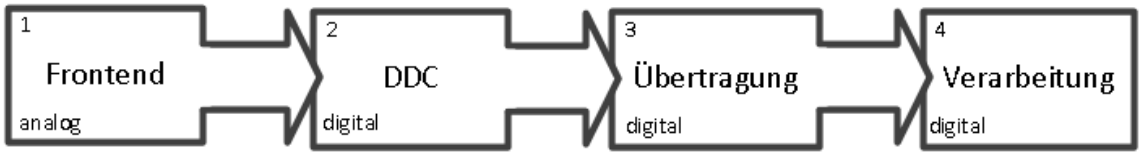

Abbildung 3.2: Überblick des zu realisierenden Gesamtsystems

Der DDC wird vom bestehenden DDC-Core [29] gestellt und in ein CPU-basiertes Embedded-System auf einem FPGA des Xilinx ML507 FPGA-Board als CPU-Peripherie implementiert. Der DDC-Core arbeitet mit dem LTC2206 ADC fest zusammen, sodass das zugehörige Evaluations-Board LT918C [13] weiter genutzt wird. Zudem muss das Embedded System eine Schnittstelle zum PC aufweisen (siehe nächstes Kapitel).

Für eine bessere Übersicht des Systems kann das Embedded System auf Basis des FPGA-Boards als einfaches Input-Output-System gesehen werden (Abbildung 3.3). Es bietet einen analogen Eingang mit ADC, einen digitalen Ausgang für die Daten, einen digitalen Steuerkanal und weitere digitale Ausgänge für die Steuerung eines analogen Frontends. Dieses System ermöglicht, in Abhängigkeit von den Steuerbefehlen des Steuerkanals, die Steuerung der Frequenzverschiebung im Spektrum des digitalisierten Analogeingangsspektrums, die Dezimierung des Datenstroms und die Anpassung des Analogsignals an den ADC, falls ein komplexes Frontend verwendet wird. Weitere Einstellungsmöglichkeiten sind natürlich möglich, jedoch für dieses System nicht nötig. Diese könnten später einfach hinzugefügt werden.

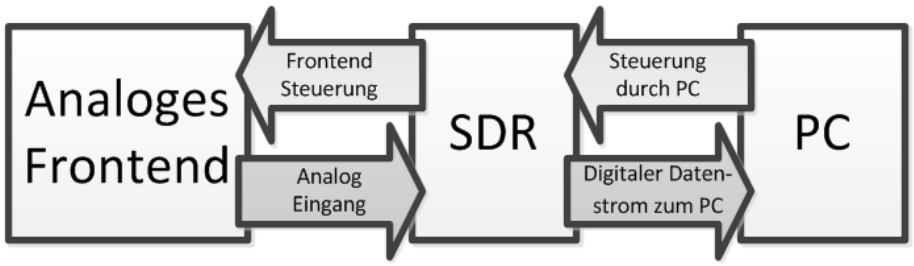

Abbildung 3.3: Embedded System als Input-Output-System

Generiert werden Steuerbefehle von einem plattformunabhängigen PC-Programm (SDR-Client), welches von einem PC-Benutzer bedient wird. Der SDR-Client muss das SDR auf eine grafische Benutzeroberfläche (GUI) abstrahieren, sodass eine einfache und benutzerfreundliche Bedienung möglich ist. Zu den Steuerungsmöglichkeiten gehört das Administrieren des bestehenden DDC-Cores und eines einfachen VHF-2 Frontends.

Zur Verwendung des SDR könnte ein Benutzer ein Frontend mit Kompatibilität zu der jeweilig implementierten Frontendschnittstelle und einer entsprechenden Antenne an das SDR anschließen. Anschließend das SDR mit dem PC verbinden und über den Steuerkanal dem SDR mitteilen, einen Datenstrom anhand der konfigurierten Einstellungen zu senden. Dies würde dem PC ermöglichen, den SDR-Datenstrom zu empfangen und zu verarbeiten.

3.3 Datenübertragung zum PC

Die zu implementierende Übertragungsart ist ein wichtiger Faktor in einem System bei dem es darauf ankommt, Daten in Echtzeit zu übermitteln. In diesem Teil der Arbeit soll gezeigt werden, welche Übertragungsarten für das zu implementierende SDR auf Basis des Xilinx ML507 FPGA-Boards, in Verbindung mit dem PC, in Fragen kommen.

3.3.1 Vorgaben

Eine wichtige Eigenschaft der Übertragungsart ist die Unabhängigkeit vom Client-System, hier der PC. Es muss möglich sein das SDR als eine Einheit zu nehmen, an einen anderen Platz zu bringen und mit geringem Aufwand in Betrieb zu nehmen. Da das SDR-System auch in den Laboren der HAW-Hamburg zum Einsatz kommen soll, ist ein Zugriff von mehreren Client-Systemen im Labor wünschenswert. So würde nicht jeder Student ein eigenes SDR-Transceiver benötigen.

Weiter ist die mögliche Übertragungsgeschwindigkeit (Datenrate) eine essentielle Eigenschaft, die zu beachten ist. Als minimales Maß ist die Datenrate des bestehenden Systems von 5 MB/s pro IQ-Datenstrom zu sehen. Dies ist das Maximum, welches sich, bei der fest implementierten Tiefpassfrequenz von 200 kHz, einer Dezimierung von 64, einem Abtasttakt von 80 MHz und 2 Byte je I-/Q-Sample ergibt ($\frac{f_s}{64} \cdot 4\ Byte = 5\ MB/s$) [29]. Das SDR soll die Möglichkeit bieten, später um mehrere Datenströme erweitert zu werden, sodass zum Beispiel zwei verschiedene Funkbänder unabhängig von einander in Echtzeit empfangen werden können.

Als Basis dieses SDR dient das Xilinx ML507 FPGA-Board, welches einige Schnittstellen bereitstellt, um eine Echtzeit Datenübertragung zum PC zu realisieren. Hierzu gehören USB, PCI-Express, Serial-ATA und Ethernet (TCP/IP und UDP/IP).

3.3.2 Übertragungsarten

USB

USB 2.0 würde sich für eine Übertragung anbieten, da hierfür der PHY[1] auf dem Board bereits vorhanden ist. Zudem bietet das paketorientierte serielle USB, im Bezug auf die benötigte minimale Bandbreite von 5 MB/s, mit einer Datenübertragungsrate von theoretisch

[1] Ein Chip der die untersten Schichten des jeweiligen Übertragungsprotokolls umsetzt.

maximal 60 MB/s, eine gute Übertragungsgeschwindigkeit [38]. Bedingt durch den Protokoll-Overhead wird in der Praxis jedoch nur eine Übertragungsrate von 30-35 MB/s erreicht [39]. Bei der USB 3.0 ist jedoch die 10-fache Übertragungsgeschwindigkeit möglich.

USB unterscheidet zwischen vier verschiedenen Übertragungsmodi: Bulk-Modus für zeitunkritische große Datenmengen, zum Beispiel ein USB-Massenspeicher. Interrupt-Modus für kleine nicht periodische Datenmengen als Trigger für den Host. Control-Modus für die Gerätekonfiguration. Und zuletzt der Isochrone-Modus für Datenstrom-Übertragungen wie Audio-Daten, bei welchen eine garantierte Bandbreite für die Übertragung festgelegt werden kann. Allerdings erfolgt beim letzten Modus auch keine Sicherung der Daten über zusätzliche Handshake-Pakete, wie es bei den anderen Modi der Fall ist. Für eine Echtzeit-Übertragung ist dies kein Problem, denn bei einem eventuellen Datenverlust hätte der Host sowieso keine Zeit, diese wieder aufzuarbeiten.

Des Weiteren spricht für USB, dass heutzutage jeder PC-Nutzer mit einem USB-Stecker etwas anfangen kann, denn jeder PC verfügt über eine USB-Schnittstelle. Die meisten Betriebssysteme bringen ein Subsystem für die Nutzung von USB und generische Treiber mit sich. Die generischen Treiber sind nicht für jede Hardware verfügbar, sodass speziell für das jeweilige Betriebssystem Treiber entwickelt werden müssen. Für ein plattformunabhängiges SDR wäre dies nicht von Vorteil.

Dem Abschnitt 3.1 ist zu entnehmen, dass für das bestehende System eine USB-Übertragung auf Basis eines generischen USB-Massenspeicher-Treibers genutzt wurde. Diese Lösung wird von den meisten PC-Client-Systemen ohne zusätzlichen Treiber unterstützt. In dieser Implementierung bietet diese allerdings nur eine geringe Übertragungsrate von 10 MB/s und belastet die PC-CPU deutlich durch Polling des Control-Blocks im Shared-Memory des SDR.

PCI-Express

Als Übertragungsart mit mehreren bidirektionalen seriellen Lanes[2], ist PCI-Express (Peripheral Component Interconnect Express) eine weitverbreitete und skalierbare Übertragungsart. Je Endpunkt können mehrere Lanes gebündelt werden, um so über eine Switch[3]-Struktur paketorientiert Daten zu übertragen.

Mit PCI-Express ist theoretisch eine maximale Übertragungsrate (PCIe 3.0 und x32 Slot) von 32000 MB/s möglich [33]. Jedoch wird vom ML507 FPGA-Board nur eine PCIe 1.0 Lane bereitgestellt, welches die Übertragungsrate auf effektiv 250 MB/s beschränkt [44].

[2] Eine Lane entspricht einem Sende- und einem Empfangskanal

[3] Die Verbindungen der Übertragungswege können umgeschaltet werden. Wie ein Gleis bei der Bahn.

Dies ist eine respektable Übertragungsgeschwindigkeit für die SDR-Anbindung. Jedoch ist PCI-Express eine PC interne Schnittstelle und das FPGA-Board muss somit auch in den PC montiert werden. Dieses würde die mobile Nutzung des SDR sehr einschränken.

Serial-ATA

Genau wie PCI-Express ist Serial-ATA (SATA - Serial Advanced Technology Attachment) eine geräteinterne Übertragungsart und würde die mobile Nutzung des SDR ebenfalls einschränken. SATA 2.0 erlaubt eine theoretische maximale Übertragungsrate von 300 MB/s und mit Version 3.0 sogar das Doppelte [36]. SATA beinhaltet seit SATA 2.0 die eSATA-Spezifikation (External Serial-ATA). eSATA spezifiziert die Anschlussleitungen auch für den Betrieb außerhalb eines PC-Gehäuses. Heutige PCs sind jedoch nicht standardmäßig mit eSATA-Schnittstellen ausgestattet, wie es bei USB oder PCI-Express der Fall ist.

Ethernet

Ethernet stammt aus den 1970ern und ist schon relativ alt. Dennoch hat sich Ethernet (beschrieben im IEEE (Institute of Electrical and Electronics Engineers) Standard-Paket IEEE 802.3[4]) mit seinen vielen Spezifikationen schnell und dominant durchgesetzt.

Heutzutage kommen in Local-Area-Network (LAN) Netzwerken meist die 100 Mb/s- und die 1000 Mb/s-Varianten (GigE) zum Einsatz. Genau diese werden auch von dem PHY des ML507 FPGA-Boards unterstützt und kommen für das zu realisierende SDR in Frage.

GigE ist eine bidirektionale serielle Übertragungsart, die nicht für den geräteinternen Gebrauch bestimmt ist. Vielmehr ist sie für LAN-Netzwerke, die sich in einem Gebäude über mehrere hundert Meter erstrecken, gedacht. Eine GigE-Leitung kann bis zu 100 m betragen und mit Hilfe von Hubs[5] die Endpunkt-zu-Endpunkt Länge erhöhen. Ethernet arbeitet paketorientiert mit einer variablen Paketgröße. Standard-Pakete sind 1500 Byte groß. Es gibt auch die Möglichkeit eine Erweiterung in der Spezifikation zu nutzen, welche größere Pakete erlaubt (Jumbo-Frames). Sie erlauben dem Übertragungssystem, mehr Nutzdaten in ein Paket unterzubringen und reduzieren so stark den Protokoll-Overhead.

Im OSI-Schichtenmodell[6] ist Ethernet auf den beiden untersten Schichten (Bitübertragungs- und Sicherungsschicht) angesiedelt. Somit wird durch Ethernet beschrieben, wie das Übertragungsmedium zu dimensionieren ist und sich die zu übertragenen Symbole auf dem Über-

[4] Weitere Informationen zum IEEE 802.3 Standard: http://www.ieee802.org/3/

[5] Netzwerk-Gerät, welches auf dem OSI-Layer 1 arbeitet und Ethernet-Endpunkte verbindet.

[6] Beschreibt ein Übertragungsprotokoll in sieben abstrakten Schichten, wobei nur die unmittelbar benachbarten Schichten miteinander kommunizieren können.

tragungsmedium abbilden lassen. Die zweite Schicht gibt vor, wie diese zeitlich anzuordnen sind und welche Sicherungsmaßnahmen für eine erfolgreiche Übertragung zu treffen sind.

Ethernet alleine (Raw-Ethernet) ist seriell aufgebaut und kann nur eine bidirektionale Endpunkt-Endpunkt-Kommunikation aufbauen, also das physikalische Medium mit einem Kanal belegen. Es ist aber möglich einen TCP/IP-Stack[7] auf Ethernet aufzusetzen, sodass eine paketorientierte Kommunikation auch in höheren OSI-Modell-Schichten möglich ist. Diese Kommunikation kann dann nicht nur zwischen Ethernet-Endpunkten erfolgen, sondern auch zwischen abstrakteren Endpunkten über der Sicherungsschicht.

Der Kern dieses Protokoll-Stacks sind die Protokolle Internet Protocol (IP), User Datagram Protocol (UDP) und Transmission Control Protocol (TCP). IP bildet die unterste Schicht und arbeitet auf der Schicht 3 (Vermittlungsschicht) des OSI-Modells. Es sorgt für eine Kommunikation von IP-Endpunkten (gekennzeichnet durch IP-Adressen) im Netzwerk, welches sich über eine beliebige Netzwerktopologie bestehend aus Switchen[8] und Routern[9] erstrecken kann. So entstehen mehrere Kanäle über ein Ethernet-Medium.

Eine Schicht über IP arbeiten UDP und TCP. Sie sorgen für die Möglichkeit, mehrere parallele Kommunikationswege (Ports) zwischen Software-Prozessen herzustellen. In Abbildung 3.4 ist ein Ausschnitt aus dem OSI-Modell gezeigt. Es soll verdeutlichen, welche Protokolle wo im OSI-Modell ansetzten.

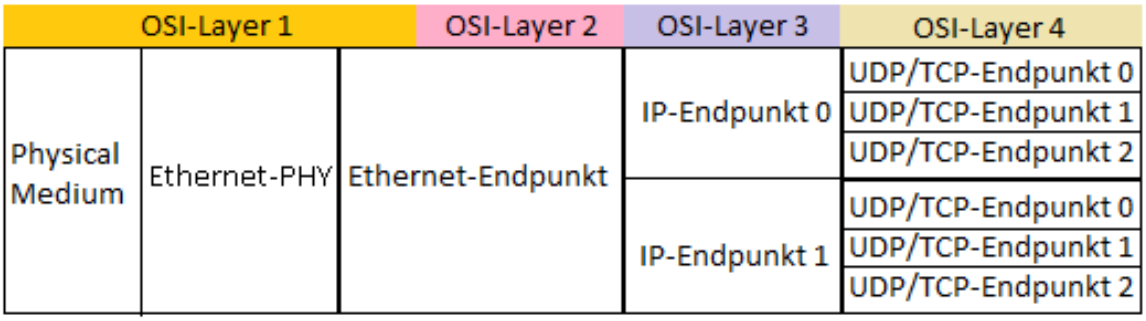

Abbildung 3.4: Ausschnitt aus dem OSI-Modell

Das UDP-Protokoll ist, im Vergleich zu TCP, sehr einfach aufgebaut und beschränkt sich mit seiner Tätigkeit nur auf das Austauschen von Daten zwischen Ports von IP-Endpunkten, mit Hilfe von einzelnen Datenpaketen. Auf eine Fehlersicherung für verlorengegangene Pakete

[7] Mehrere Protokolle die miteinander interagieren und eine paketorientierte Verbindung ermöglichen. Dieses erlaubt das Weiterleiten (routen) von Paketen durch das gesamte Netzwerk. Des Weiteren ist eine Prozess-Prozess Kommunikation möglich.

[8] Verbinden IP-Endpunkte und ermöglichen das Weiterleiten von Ethernet-Paketen in einem IP-Netzwerk.

[9] Leiten IP-Pakete zwischen verschiedenen IP-Netzwerken weiter.

wurde verzichtet. UDP ist ein verbindungsloses Protokoll, welches einzelne Datenpakete verschickt und nicht auf eine Reihenfolge achtet. Es kann somit passieren, dass ein später losgeschicktes Paket einen kürzen Weg durch das IP-Netz findet und somit früher am Endpunkt ankommt. Durch diese Eigenschaft spart sich UDP die Zeit, eine Reihenfolge wiederherzustellen und lässt mehr Rechenzeit frei. Echtzeitübertragungen müssen sicherstellen, dass eine konstante Bandbreite zur Verfügung steht, sodass auf Wiederholungen von Paketen nicht gewartet werden kann. Somit ist UDP für eine Echtzeitübertragung geeignet.

Eine abgesicherte Prozess-Prozess-Kommunikation über IP ist mit TCP möglich. Das Protokoll ist verbindungsorientiert und stellt während einer verbundenen Sitzung von TCP-Endpunkten sicher, dass alle Pakete in der richtigen Reihenfolge ankommen. Der dadurch entstehende Overhead übersteigt den von UDP, sodass TCP im Regelfall langsamer arbeitet.

Heutzutage einen PC ohne einen Ethernetadapter zu finden, ist sehr selten. Dies gilt ebenso für Betriebssysteme und die TCP/IP-Unterstützung. Jedes gängige Betriebssystem implementiert bereits einen TCP/IP-Stack und ermöglicht einen Zugriff auf die Protokollschichten über Subsysteme ohne spezielle Treiber erstellen zu müssen.

3.3.3 Praktische Ethernet und UDP/IP Übertragungsrate

Da UDP/IP over Gigabit-Ethernet mit seinen Eigenschaften ($\frac{1000Mb/s}{8} = 125MB/s$ Übertragungsrate für Echtzeit Übertragungen, Plattform- und Treiberunabhängigkeit und separate Port-Kanäle für parallel Streams) stark für dieses SDR in Frage kommt, soll in diesem Abschnitt die praktisch erreichbare UDP-Übertragungsgeschwindigkeit mit dem Ziel-System (ML507 FPGA-Board und Windows-PC) näher untersucht werden. Hierbei kommt es nicht auf eine optimierte Übertragung an. Vielmehr soll eine erste schnelle Analyse zeigen, was ein unoptimiertes System auf Basis von Ethernet und UDP/IP leisten kann.

Alle erstellten Systeme mit der jeweiligen Firmware können im Anhang (*3* und *4*) dieser Arbeit eingesehen werden.

Übersicht

Es kommen zwei Testsysteme zum Einsatz, um die Ethernet und UDP/IP Übertragungsgeschwindigkeit zu analysieren. Diese gliedern sich in das FPGA-System als Server, dem PC als Client und der eigentlichen Übertragungsstrecke. Als Übertragungsstrecke dient ein Cat5-Ethernet-Kabel. Es verbindet die Ethernetadapter des FPGA und des PCs direkt ohne über einen Netzwerkknoten zu gehen. Da verschiedene Paketgrößen, auch Jumbo-Frames, getestet werden sollen, sind Ethernetadapter mit der Option für Jumbo-Frames nötig. Durch

den PC Ethernetadapter *Intel Gigabit CT Desktop Adapter* und den des FPGA-Board ML507 ist dieses für beide Seiten gegeben. PC und FPGA können für diesen Test kontinuierlich Daten versenden(Tx) beziehungsweise empfangen(Rx), sodass die Übertragungsgeschwindigkeit ermittelt werden kann.

Testsysteme

Auf Basis des zur Verfügung stehenden Xilinx ML507 FPGA-Boards und mit Hilfe der Xilinx ISE Design Suite 12.4[10] sind zwei Central-Prozessing-Unit (CPU)-basierte Basis-Systeme entworfen worden (Microblaze und PowerPC440), die GigE unterstützen. Es sind einfache Basis-Systeme die sich auf das Wesentliche (Ethernet) beschränken und nicht optimiert sind (Abbildung 3.5). Für beide Testsysteme setzt es sich grob aus sechs Teilen zusammen (CPU, Random-Access-Memory (RAM), RAM-Controller, Direkt-Memory-Access-Controller (DMA-Controller), Peripherie-Bus und Ethernet-Core mit DMA).

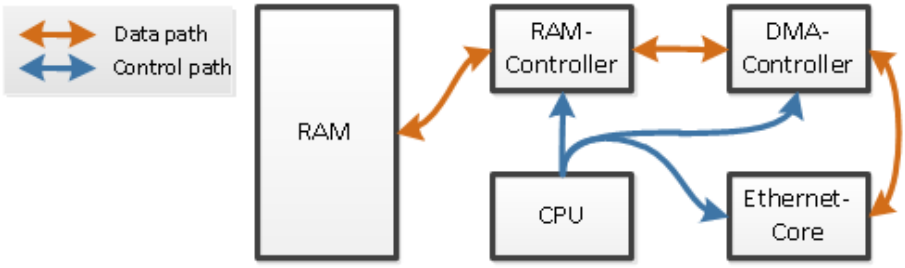

Abbildung 3.5: Übersicht des Ethernet/UDP Test-Systems im FPGA

Als CPUs der beiden Systeme dienen zum Einen ein Microblaze (MB) Soft-Core Prozessor, welcher mit 125 MHz getaktet ist. Zum Anderen kommt ein PowerPC440 (PPC) mit 400 MHz Takt zum Einsatz. Die Besonderheit beim PPC liegt in der Auslegung als Hard-Core. Das heißt, der Core ist im FPGA als Silizium-Chip enthalten und erreicht ohne spezielle Timing-Constrains hohe Taktraten. Zu dem PPC existiert ein RAM-Controller (ppc440mc_ddr2). Dieser kommt für das PPC Testsystem jedoch nicht zum Einsatz, da für das spätere SDR-Endsystem eine Native-Port-Interface[11] (NPI) Schnittstelle für den DDC-Core benötigt wird. NPI wird vom ppc440mc_ddr2 nicht unterstützt, sodass der Soft-Core mpmc[12] eingesetzt

[10] Xilinx Entwicklungstoolchain für Xilinx FPGAs und CPLDs (http://www.xilinx.com).

[11] Ist ein natives Interface des MPMC-RAM-Controllers und erlaubt den schnellstmöglichen Zugriff auf das RAM. NPI wird im MPMC selbst genutzt, um anderen Schnittstellen, wie PLB, den Zugriff auf das RAM zu erlauben.

[12] Soft-Core RAM-Controller mit zahlreichen Schnittstellen für den Zugriff auf das DDR2-RAM. Weitere Informationen im Xilinx Datenblatt DS643.

werden muss. Ein Testsystem ohne NPI würde keine brauchbaren Ergebnisse für diese Arbeit liefern.

Kontrolliert werden alle Peripherie-Cores über den Prozessor-Local-Bus (PLB)[13]. PLB wurde gewählt, da der DDC-Core bereits eine funktionierende PLB-Schnittstelle enthält und deshalb später im SDR-Endsystem zum Einsatz kommen wird.

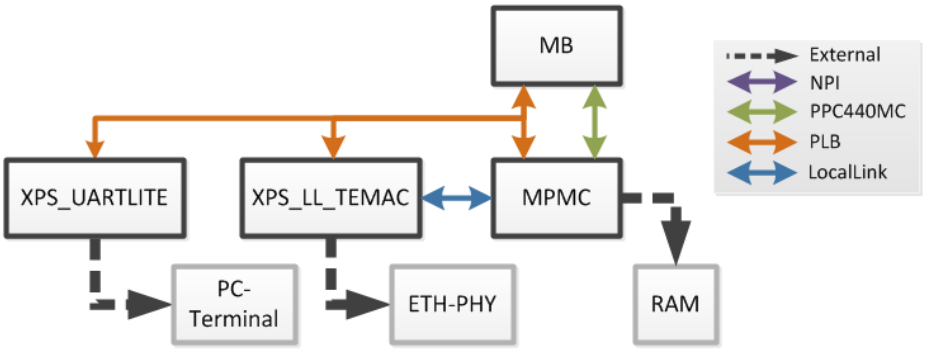

Abbildung 3.6: Aufbau des Microblaze Testsystems im FPGA

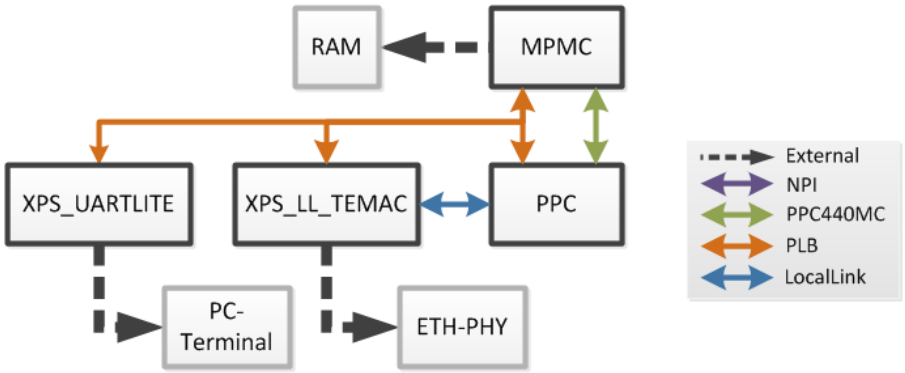

Abbildung 3.7: Aufbau des PowerPC440 Testsystems im FPGA

DMA-Transaktionen erfolgen über LocalLink[14]-Kanäle, welche Punkt-zu-Punkt Fifo-Kanäle von der Quelle (Ethernet-Core) zu dem Ziel (RAM) darstellen und so die Last vom PLB-Bus nehmen. Sie bieten zudem die Möglichkeit, Checksum-Offloading[15] in den Übertragungsweg zu integrieren. Dies kommt jedoch in diesem unoptimierten System vorerst nicht zum Einsatz.

[13] Zentrales Bussystem als Highspeed Element aus dem CoreConnect-Paket der Firma IBM.
[14] FIFO-basiertes Protokoll zur direkten Verbindung von Peripherie im Embedded System.
[15] Ermöglicht die Checksummenberechnung (UDP/TCP-Pakete) in Hardware statt in Software.

Als Ethernet-Core der Systeme, dient für beide Systeme der Hard-Core xps_ll_temac[16]. Damit DMA-Transaktionen zwischen Ethernet-Core und RAM-Controller möglich sind, wurden die Rx/Tx LocalLink Kanäle mit dem RAM-Controller verbunden. Um auch große Ethernet-Pakete (9 KiB) erfolgreich empfangen bzw. senden zu können wurden, die Tiefe der Rx und Tx Fifos auf 32 KiB gesetzt. So können mindestens drei komplette Ethernet-Pakete mit 9 KiB erfolgreich empfangen werden, bis eine DMA-Transaktion nötig ist.

Zu den Basiselementen aus Abbildung 3.5 wurde ein Universal-Asynchronous-Receiver-Transmitter (UART) in beide Systeme integriert, welcher es erlaubt, die Testsysteme zu steuern oder Debug-Information auszugeben.

Die zwei folgenden Abbildungen (3.6 und 3.7) zeigen das CPU-basierte System im FPGA. Für weitere Informationen zu den realisierten FPGA-Systemen sind die Xilinx EDK Projekt Dateien aus Anhang *3* und *4* in Augenschein zu nehmen.

Ethernet

Eine reine Ethernet-Übertragung kann mit den Testsystemen realisiert werden, indem der jeweilige Prozessor seine Daten über DMA aus dem Ethernet-Core (MAC[17]) verschickt. Für einen solchen Test, wie er hier durchgeführt werden soll, wurde von Xilinx eine Firmware (PerfApp[7]) entwickelt, die über eine UART-Konsole bedient werden kann. Hierzu gehört unter anderem eine Funktion mit dem Namen *ChannedTx*. Diese erlaubt es, in einem Sendetestdurchlauf, verschiedene Paketgrößen und Packet-Thresholds[18] zu kombinieren und zeigt die Ergebnisse in der Konsole. Eine manuelle Anpassung der Einstellungen ist während des Testdurchlaufs so nicht nötig. Die Abbildungen 3.8 und 3.9 zeigen die Ergebnisse für das MB und PPC-System.

Die Ergebnisse zeigen für verschiedene Ethernet-Paketgrößen und Paket-Thresholds die Übertragungsgeschwindigkeit in Mib/s, die Auslastung von Ethernet und die CPU-Aktivität. Der Paket-Threshold gibt hierbei an, wie viele Interrupts des Ethernet-MACs zusammengefasst werden, um so die CPU von einer hohen Interruptrate zu entlasten. Die Paketgröße stellt die Anzahl an Bytes im Datenbereich des Ethernetpakets dar und bestimmt hiermit den Protokoll-Overhead pro Paket.

Aus den Abbildungen 3.8 und 3.9 wird deutlich, dass mit steigendem Paket-Threshold und steigender Paketgröße die CPU-Aktivität sinkt, welches der CPU mehr Zeit für andere Programme/Threads lässt. Diese Zeit wird vom Testsystem genutzt, um schneller neue Pakete

[16] Wrapper für den xps_ll_temac Hard-Core Ethernet-MAC im Virtex-5 FXT. Weitere Informationen im Xilinx Datenblatt DS537.

[17] Medienzugriffssteuerung. Siehe Schicht 2 im OSI-Model.

[18] Die Interrupts des Ethernet-Cores können zusammengefasst werden und minimieren somit die Interrupt rate. Packet-Thresholds ist der Grad der Zusammenfassung.

```
**************************************************************************
Interrupt Driven SG DMA Performance
**************************************************************************

                Packet Threshold settings
          +------------------+------------------+------------------+------------------+
          |        1         |        2         |        8         |       64         |
          +------------------+------------------+------------------+------------------+
 Frame          Net   CPU          Net   CPU          Net   CPU          Net   CPU
  Size   Mbps Util  Util    Mbps Util  Util    Mbps Util  Util    Mbps Util  Util
  ----   ----- ----- ----    ----- ----- ----    ----- ----- ----    ----- ----- ----
    64   10.2   1.3  100     10.2   1.3  100     10.2   1.3  100     10.2   1.3  100
   128   20.5   2.4  100     20.5   2.4  100     20.5   2.4  100     20.5   2.4  100
   512   82.0   8.5  100     81.9   8.5  100     81.9   8.5  100     81.9   8.5  100
  1024  163.7  16.7  100    163.7  16.7  100    163.7  16.7  100    163.7  16.7  100
  1518  242.5  24.6  100    242.4  24.6  100    242.4  24.6  100    242.4  24.6  100
  2048  324.8  32.8  100    324.7  32.8  100    326.7  33.0  100    326.6  33.0  100
  4096  633.1  63.6  100    633.4  63.6  100    633.3  63.6  100    632.1  63.5  100
  8192  994.3  99.7  100    994.1  99.7  100    994.1  99.7  100    995.9  99.8   80
  9000  994.4  99.7  100    994.2  99.6  100    994.2  99.6  100    994.2  99.6   73
```

Abbildung 3.8: ChannedTx-Ergebnisse für das Microblaze-System

```
**************************************************************************
Interrupt Driven SG DMA Performance
**************************************************************************

                Packet Threshold settings
          +------------------+------------------+------------------+------------------+
          |        1         |        2         |        8         |       64         |
          +------------------+------------------+------------------+------------------+
 Frame          Net   CPU          Net   CPU          Net   CPU          Net   CPU
  Size   Mbps Util  Util    Mbps Util  Util    Mbps Util  Util    Mbps Util  Util
  ----   ----- ----- ----    ----- ----- ----    ----- ----- ----    ----- ----- ----
    64   665.0  87.3  100    665.1  87.3  100    664.6  87.2  100    664.6  87.2  100
   128   853.4  98.7  100    853.4  98.7  100    853.4  98.7   96    853.4  98.7   69
   512   957.0  99.4   89    957.0  99.4   53    957.0  99.4   26    957.0  99.4   18
  1024   978.7  99.8   48    978.8  99.8   28    978.8  99.8   14    978.7  99.8    9
  1518   985.7  99.9   31    985.7  99.9   19    985.7  99.9    9    985.7  99.9    6
  2048   989.4  99.9   24    989.4  99.9   14    989.4  99.9    7    989.2  99.9    5
  4096   994.6  99.9   12    994.6  99.9    7    994.6  99.9    3    994.5  99.9    2
  8192   997.2 100.0    6    997.2 100.0    3    997.2 100.0    2    996.6  99.9    1
  9000   997.5 100.0    6    997.5 100.0    3    997.5 100.0    2    997.2  99.9    1
```

Abbildung 3.9: ChannedTx-Ergebnisse für das PPC440-System

zu senden, sodass die Datenübertragungsrate ansteigt. Bei hohem Threshold ist darauf zu achten, die Hardware-Fifos im Ethernet-MAC groß genug zu halten, damit alle Daten zwischen dem Auftreten der Interrupts darin Platz finden.

Das MB-System, so wie auch das PPC-System, erreichen, mit großen Paketen und hohem Threshold, GigE bis 99% der theoretischen Möglichkeiten. Allerdings ist die CPU-Auslastung des MB-Systems, bedingt durch die geringere Taktfrequenz, deutlich größer (hier 73%) als die des PPC. Wenn die CPU noch andere Dinge tun soll, wie zum Beispiel das UDP Protokoll umsetzen, kann dies bei diesem MB-System zu erheblichen Geschwindigkeitseinbußen kommen und so auch höhere Protokollschichten bremsen.

UDP/IP

Wie für Ethernet liefert Xilinx mit der Application-Note XAPP1026 [27] eine Testfirmware die es ermöglicht UDP-Rx/Tx Übertragungsratentests durchzuführen. Diese Firmware setzt auf den Open Source Software TCP/IP-Stack lwIP[19] auf. lwIP wurde speziell für Embedded-Systeme mit wenig Resourcen und Speicher entwickelt und ist somit gut für diesen Test geeignet. Mit UDP, IP, Internet-Control-Message-Protocol[20] (ICMP) und Address-Resolution-Protokoll [21] (ARP) stellt lwIP genau die benötigten Protokolle bereit. Es gibt noch zahlreiche andere Stacks, wie zum Beispiel uIP (http://www.sics.se/ adam/old-uip/) oder auch Treck (http://www.treck.com/). Xilinx bietet lwIP als Bibliothek im EDK[22] bzw. SDK[23] mit Treiber für den xps_ll_temac an. So ist eine schnelle und unkomplizierte Nutzung des Stacks möglich.

Ein weiterer Stützpfeiler der Xilinx Firmware ist eine Kompatibilität zu iPerf[24] (eine grafische Oberfläche (GUI) zu iPerf existiert mit jPerf[25]). iPerf stellt PC-Programme (Client und Server) zur Messung der TCP/UDP-Übertragungsgeschwindigkeit von PC zu PC. Xilinx hat beide Programme (Server und Client) in seine Firmware integriert, sodass eine Messung der UDP-Übertragungsgeschwindigkeit von PC zu FPGA möglich ist. iPerf ist relativ einfach aufgebaut und verschickt als Server mit einer gewünschten Bandbreite Pakete an einen bestimmten UDP/IP-Endpunkt und fügt in die ersten 4 Byte einen vorzeichenlosen 32 Bit Identifikator (inkrementiert je Paket) ein, welcher dem iPerf-Client, sagt ob Pakete verloren gegangen sind.

Das Messen des UDP-Datendurchsatzes im Sendebetrieb (vom FPGA aus gesehen) ist mit Hilfe der eingefügten iPerf-Identifikatoren einfach möglich. Die Firmware des Systems sendet kontinuierlich Pakete in der jeweiligen Größe (plus iPerf-Identifikator) an den PC. Der PC lauscht auf dem konfigurierten UDP-Endpunkt und ermittelt die Geschwindigkeit. Für die Messung im Empfangsbetrieb sieht dies anders aus. Die System-Firmware empfängt iPerf-kompatible UDP-Pakete und wertet deren Identifikator aus. Sind mehr als 100 Pakete verloren gegangen, wird in der Debug-Konsole ein Fehler signalisiert. Somit muss am Paketsender (PC) die Paket-Sendefrequenz manuell angepasst werden, bis keine Fehler mehr

[19] Ist eine ressourcen- und speichersparende Implementierung des TCP/IP-Protokollstapels und implementiert die folgenden Protokolle: IP, ICMP, UDP, TCP, DHCP, PPP und ARP. Es gibt zwei signifikante API's, dies sind eine Socket-API für Systeme mit Betriebssystem und eine Raw-API für Geschwindigkeitsoptimierte Systeme - siehe http://savannah.nongnu.org/projects/lwip/

[20] Protokoll in der OSI-Schicht 3 zur Kontroll-Kommunikation von IP-Endpunkten

[21] Protokoll der OSI-Schicht 2 zur Auflösung von MAC-Adressen zu IP-Adressen.

[22] Ist Teil der ISE Design Suite und stellt ein Embedded Development Kit dar. Mit diesem ist der Entwurf und die Realisierung von Embedded Systemen in Xilinx FPGAs möglich.

[23] Ist Teil der ISE Design Suite und stellt ein Software Development Kit dar. Mit diesem ist es möglich auf, den im EDK erstellte,n Embedded Systemen eine Software zu entwickeln.

[24] http://sourceforge.net/projects/iperf/.

[25] Java-basierte GUI für iPerf. http://code.google.com/p/xjperf/

in der Konsole erscheinen. Aus diesem Grund sind die Messergebnisse im Empfangsbetrieb ein wenig ungenauer als die des Sendebetriebs, für diese Analyse jedoch ausreichend.

Aus der gegebenen Firmware wurden lediglich die Teile für eine UDP Messung mit einkompiliert, um das Gesamtsystem weniger zu belasten. Es ist jedoch eine kleine Modifikation am UDP-Testcode nötig, denn standardmäßig wird im Serverbetrieb ein lwIP-Buffer (pbuf) für alle ausgehenden UDP-Pakete erstellt. Das bedeutet, wenn der pbuf den lwIP-Stack durchläuft und seine Protokollinformation der jeweiligen Protokollschichten erhält, verändert dieser seine Größe und seinen internen Payload-Pointer[26]. Nachdem der pbuf durch den Stack an den DMA-Controller übergeben wurde, ist dieser noch nicht über Ethernet verschickt, aber für ein zweites neues UDP-Paket wieder verfügbar. Diese neue UDP-Transaktion würde mit dem pbuf arbeiten der im DMA-Kanal residiert und führt somit zu Fehlern in der Übertragung. Es muss also für jede neue Transaktion ein neuer pbuf erstellt werden.

Der lwIP wurde mit einem Threshold von 16 Interrupts und 32 DMA-Deskriptoren[27] betrieben. Die Speicherverwaltung ist so konfiguriert, dass laut den lwip-Statistiken während der Laufzeit nie der komplette Speicher verwendet wird. pbufs aus dem lwIP-Pool[28] haben eine Größe von 9000 Byte erhalten, um auch Jumbo-Frames direkt ohne Reallokierung aufnehmen zu können. Das in der Hardware verfügbare Checksum-Offloading ist bei diesen Messungen nicht zum Einsatz gekommen, denn eine Nutzungsmöglichkeit ist von lwIP nur mit TCP gegeben. Genauere Information zu den lwIP-Einstellungen sind den im Anhang (*3* und *4*) befindlichen *.mss-Dateien der EDK-Projekte zu entnehmen.

System	Paketgröße in Byte	Durchsatz in $\frac{Mb}{s}$ / $\frac{MB}{s}$ (TX)	Durchsatz in $\frac{Mb}{s}$ / $\frac{MB}{s}$ (RX)
MB	1024	$\approx 109/13,6$	$\approx 120/15,0$
MB	2048	$\approx 178/22,3$	$\approx 110/13,8$
MB	4096	$\approx 207/25,9$	$\approx 245/30,6$
MB	8192	$\approx 250/31,3$	$\approx 250/31,3$
PPC	1024	$\approx 430/53,8$	$\approx 120/15,0$
PPC	2048	$\approx 525/65,6$	$\approx 123/15,4$
PPC	4096	$\approx 588/73,5$	$\approx 245/30,6$
PPC	8192	$\approx 626/78,3$	$\approx 253/31,6$

Tabelle 3.1: UDP Datendurchsatz ohne Checksum-Offloading

Die Ergebnisse aus Tabelle 3.1 zeigen, wie unterschiedlich schnell die Testsysteme sind und dass die Tendenz der Ethernet-Übertragungsgeschwindigkeiten aus dem vorigen Abschnitt

[26] Pointer im pbuf auf den Start des jeweiligen (abhängig von der Protokollschicht) Datenbereiches.

[27] Bezeichnet vorallokierte Speicherbereiche für den DMA-Controller. Dieser verwaltet diesen Bereich, um DMA-Transaktionen abzulegen und diese für die eigentliche Transaktion, zwischen Peripherie und Speicher, zu nutzen.

[28] Durch lwIP allokierte pbufs zur Selbstverwaltung.

auch hier enthalten ist. Bei dem bestehenden System der Masterarbeit [29] (Kapitel 3.1) ist eine Übertragungsrate von 5 MB/s für einen IQ-Datenstrom nötig, sodass beide Testsysteme (unoptimiert) brauchbar wären. Für eine Erweiterung um mehrere Datenströme kommt jedoch nur das PPC-System in Frage, da hier deutlich höhere Tx Geschwindigkeiten zu erreichen sind. Der Unterschied von Tx und Rx ist für das SDR-Endsystem nicht weiter ein Problem, denn der Tx Pfad wird für das Senden des Datenstroms genutzt und der Rx Pfad nur für Kontroll-Informationen.

Wie bereits weiter oben erwähnt, bietet der Ethernet-MAC die Möglichkeit, Checksum-Offloading zu betreiben. Zur Zeit wird in den Testsystemen, zum Beispiel bei dem Senden eines UDP-Paketes, eine Checksumme über das ganze Paket berechnet, welches viel Zeit in Anspruch nimmt. Durch Checksum-Offloading wird dieses in Hardware ausgelagert. Zum versenden eines UDP-Pakets wird das Paket mit einer DMA-Transaktion über den TX-LocalLink-Kanal an den Ethernet-MAC geleitet. Während dieser Transaktion kann der Ethernet-MAC die Berechnung der Checksumme übernehmen. Hierfür ist vor jeder Transaktion dem Ethernet-MAC mitzuteilen, wie die Checksumme zu berechnen ist und wo diese im Paket eingefügt wird.

Leider existiert im lwIP beziehungsweise dem lwIP-Netzwerkadapter nur eine Unterstützung für TCP-Checksum-Offloading, welches prinzipiell das gleiche wie bei UDP darstellt. Im Zuge dieser Arbeit ist ein Patch für die Xilinx lwIP Library (v3.00.a) entstanden, welcher dem lwIP Checksum-Offloading auch das Senden von UDP Paketen ermöglicht (zu finden im Anhang *5*). Auf eine Implementierung von Checksum-Offloading für das Empfangen von UDP-Paketen wurde vorerst verzichtet, da der Sendebetrieb für diese Arbeit höher zu bewerten ist.

Checksum-Offloading im Sendebetrieb sollte den Datendurchsatz drastisch erhöhen und der CPU mehr Zeit für andere Threads liefern. Somit ist eine neue Messreihe im Sendebetrieb von verschiedenen UDP-Paketen nötig, welche im Folgenden dargestellt ist (Tabelle 3.2).

System	Paketgröße in Byte	Durchsatz in $\frac{Mb}{s}/\frac{MB}{s}$ (TX)	Durchsatzgewinn in %
MB	1024	$\approx 194/24,3$	≈ 78
MB	2048	$\approx 360/45,0$	≈ 102
MB	4096	$\approx 680/85,0$	≈ 229
MB	8192	$\approx 990/123,8$	≈ 296
PPC	1024	$\approx 937/117,1$	≈ 118
PPC	2048	$\approx 968/121,0$	≈ 84
PPC	4096	$\approx 980/122,5$	≈ 67
PPC	8192	$\approx 992/124,0$	≈ 59

Tabelle 3.2: UDP Datendurchsatz im Sendebetrieb mit Checksum-Offloading

Das MB-System erreicht nun bei großen Datenpaketen nahezu GigE. Deutlich ist ebenso der Anstieg des Durchsatzgewinns zu sehen, fast bis zu 300 %. Der geringe Anstieg des Durchsatzes im PPC-System, maximal ca. 118 %, ist mit den Messergebnissen der Ethernet-Messreihe zu erklären (Abbildung 3.9), denn die PPC-CPU ist bei größeren Paketen kaum belastet und hat somit viel Spielraum für die Berechnung der Checksumme in Software gelassen. Der MB hingegen (Abbildung 3.8) ist immer über 70 % ausgelastet und zeigt damit den Gewinn an Rechenzeit deutlicher.

Für das zu realisierende SDR wäre eine MB CPU ausreichend. In zukünftigen SDR-Versionen werden allerdings weitere Datenströme hinzukommen. Dies würde bei dem MB-basierten System schnell dazuführen, dass dieser total ausgelastet ist. Die Alternative ist der PPC, welcher mehr Potenzial für Erweiterungen bietet. Jedoch ist dieser durch Xilinx abgekündigt und könnte in moderneren FPGAs durch einen ARM Prozessor ersetzt werden. Da UDP/IP allerdings plattformunabhängig ist und die Firmware nur lwIP voraussetzt, kann das mit dem PPC realisierte System einfach in ein anderes Embedded System portiert werden. Vielmehr könnte der DDC-Core des bestehenden Systems durch die NPI- und PLB-Schnittstelle inkompatibel werden, da diese in der Zukunft ebenfalls nicht mehr unterstützt werden. Somit steht dem Einsatz des PPC nichts im Weg.

Eine weitere Alternative, aber für diese Arbeit vorerst uninteressant, ist ein Multiprozessor-System. In diesem könnte eine CPU die Verarbeitung des lwIP übernehmen und eine weitere die Administration des DDC-Cores. Dieses Design könnte ermöglichen ein System aus MB zu schaffen, welches auch in Zukunft noch unterstützt wird, da es auf einen Soft-Core basiert.

3.4 Embedded System im FPGA

Ein wesentlicher Stützpfeiler des SDR ist die Übertragung zum PC. Kapitel 3.3 hat bereits gezeigt welcher Übertragungsart für dieses SDR in Frage kommen würde. Aus den dort gewonnenen Erkenntnissen wurde der Entschluss gefasst, UDP/IP over Gigabit-Ethernet (UDP-GigE) als Übertragungsart zu wählen.

Die UDP-Übertragungsrate hängt stark von dem unterliegenden Hardwaresystem ab, sodass für die SDR-Realisierung ein PPC-System zum Einsatz kommen wird. Die Messergebnisse des vorigen Kapitels (3.3) zeigen, dass das PPC-System bei der Übertragung mehr CPU-Zeit zur Verfügung hat. Dies ist für eine spätere Erweiterbarkeit wichtig, denn Erweiterungen bringen immer weiteren Verbrauch an CPU-Rechenzeit mit sich. Zudem werden Folgearbeiten auf dem Xilinx ML507-Board aufbauen und diese sollen nicht mit einem System arbeiten, das auf Grund des MB ausgebremst wird.

Als Basis Embedded-System für die Realisierung wird das PPC-System aus Kapitel 3.3 herangezogen werden. Bei der Analyse wurde bereits darauf geachtet, dass eine Integration des DDC-Cores möglich ist (NPI und PLB werden benötigt). Dieser muss Einzug in das PPC-Hardwaresystem halten. In Abbildung 3.10 ist der *ddc_core* mit hinzugekommen. Dieser wird über den PLB-Bus Memory Mapped gesteuert und liefert seine Daten über die NPI-Schnittstelle direkt an den RAM-Controller. Der Datenfluss kommt vom ADC, geht über das RAM und dem PPC per DMA-Transaktion (LocalLink) zum Ethernet-MAC.

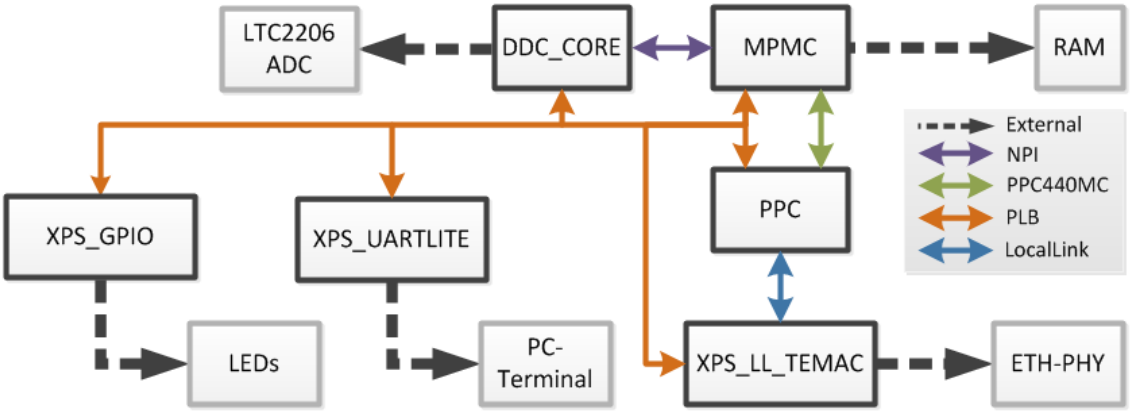

Abbildung 3.10: Embedded System im FPGA

Der DDC-Core im Embedded System wird über Memory-Mapped Register Administriert, die für eine abstrakte Nutzung allerdings nicht vom Vorteil sind. Es wird ein DDC-Treiber für den DDC-Core entstehen müssen, der einen Hardware-Abstraction-Layer (HAL) und Highlevel-Funktionen für den Zugriff und Verwaltung des DDC-Cores bereitstellt.

Ein PC ist für die Steuerung des SDR vorgesehen, was voraussetzt, dass das SDR eine Server-Funktionalität (DDC-Server) erhält, also auf Anfragen vom PC reagieren kann und diese dann ausführt. Der DDC-Server ist einem bestimmten UDP-Endpunkt zugeordnet und empfängt Pakete des Netzwerkes. Diese Pakete sollen ein Format aufweisen, welches dem DDC-Server ermöglicht, die darin kodierten Befehle umzusetzen. Wenn zum Beispiel durch einen UDP-Endpunkt im Netzwerk (PC) Daten von dem SDR angefordert werden, wird der DDC-Server den DDC-Core starten und die Daten über den konfigurierten Port an das UDP-Netzwerk senden.

Das Layer-Modell in Abbildung 3.11 zeigt die Struktur des Embedded-Systems, aufgeteilt in einzelne Schichten. Entgegen der Abbildung 3.10 ist hier ein Frontend-Core hinzugezogen worden. Da es sich hier um ein allgemeines Layer-Modell dieses SDR handelt, stellt dies zunächst nur einen Platzhalter dar, auch wenn kein komplexes Frontend eingesetzt werden wird.

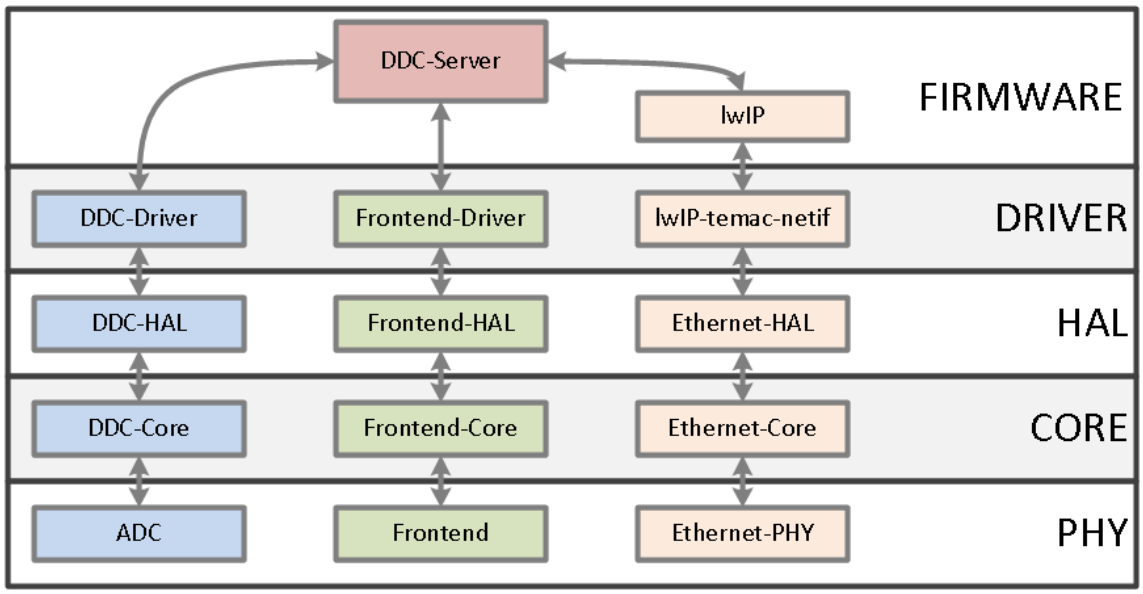

Abbildung 3.11: Layer-Struktur des Embedded-Systems

Das Layer-Modell gliedert sich in fünf Schichten, wobei die unterste am nächsten an der physikalischen Implementierung liegt. Auf der PHY-Ebene befinden sich diejenigen Systemteile die physikalisch andere Systeme steuern. Eine Abstraktionsschicht weiter oben liegen die digitalen Cores im Embedded-System, welche höheren Layern die Nutzung der PHY-Ebene Memory Mapped erlauben. Der HAL-Layer abstrahiert die Register der Cores in Software für den Zugriff. Ein Software-Treiber in der DRIVER-Ebene stellt keinen Register-Zugriff mehr da, sondern abstrahiert den jeweiligen Zweig mit Hilfe von C-Strukturen und Funktionen. Ganz oben liegt die eigentliche Software (Firmware), welche das gesamte System zusammenhält und mit den Treibern zusammenarbeitet.

Der grüne Zweig (Frontend) ist in diesem SDR nicht vorgesehen, soll jedoch für spätere Erweiterungen mit berücksichtigt werden. Somit bleibt der Zweig in dem Layer-Modell enthalten. Ein Frontend-Core wird die digitale Schnittstelle des Frontends steuern, was zum Beispiel die Verstärkung oder das Frequenzband angeht. Der HAL und Treiber der höheren Ebenen ermöglichen dem SDR einen unkomplizierten Zugriff auf die Funktionalität.

In dem PPC-Testsystem aus Kapitel 3.3 ist ein großer Teil des rechten Zweiges (UDP/IP) schon implementiert. Der PHY ist auf dem FPGA-Board bereits vorhanden und der Ethernet-Core wurde in das Embedded System integriert. Die HAL-Layer der Subsysteme (Ethernet-MAC und DMA) sind von Xilinx in den Core-Bibliotheken gegeben. Ein Treiber für den lwIP-Netzwerkadapter (netif) und lwIP selbst liefert Xilinx in seiner lwIP-Bibliothek, wobei das realisierte Patch aus der Analyse (Kapitel 3.3.3) für UDP-Checksum-Offloading auf die lwIP-Bibliothek angewendet werden wird.

Zur Nutzung des ADC von höheren Ebenen aus ist der DDC-Core nötig, welcher zu implementieren ist. Jedoch sind noch ein DDC-HAL für den abstrakteren Register-Zugriff und ein Treiber für den DDC-Core nötig. Der Treiber liefert eine Schnittstelle mit der der DDC umkonfiguriert, gestartet und gestoppt werden kann. Außerdem verfolgt dieser den vom DDC-Core genutzten Speicher, um Zugriff auf den digitalen Datenstrom durch die Firmware zu erlauben.

Der DDC-Server bindet alle Zweige zusammen. Er wartet auf UDP-Steuerpakete aus dem Netzwerk, kommend vom lwIP-Software-Stacks, und dekodiert diese. Nach erfolgreicher Dekodierung wird der jeweilige Befehl an die Treiber des DDC- und Frontend-Cores weitergegeben. Ist zum Beispiel der Start des Datenstroms gewünscht, wird der Server den ausgehenden UDP-Endpunkt einrichten, den DDC- und Frontend-Treiber konfigurieren und die Daten in Echtzeit aus dem DDC-Treiber an den externen UDP-Endpunkt leiten.

Mögliche Befehle an den DDC-Server könnten sein, diesen mit einer Konfiguration zu starten oder zu stoppen. Welche und wie diese Befehle tatsächlich realisiert werden sind dem Kapitel 4 zu entnehmen.

3.5 Analoges Frontend

Ein SDR macht nur dann Sinn, wenn es mit realen Funksignalen von einer Antenne arbeiten kann. Da sich diese Arbeit auf VHF-2 konzentriert, ist wichtig zu wissen, wie ein minimales VHF-2 Frontend auszusehen hat. Dieser Abschnitt soll diese Frage klären, sodass eine aliasingfreie Analog-Digital-Umsetzung mit dem gegebenen ADC LTC2206 (16 Bit @ 80 MHz [13] [12]) für VHF-2 möglich ist.

Reale Signale weisen, bezogen auf das Ziel-Spektrum (hier VHF-2), immer Störungen im Spektrum auf. Dies kann starkes Rauschen sein oder einfach ein nebenliegendes Funkband, das Aliasing beim ADC erzeugen würde. Ein solches Spektrum ist in Abbildung 3.12 zu erkennen. Es handelt sich hier um das VHF-2 Spektrum (87,5 MHz bis 108,0 MHz), welches direkt hinter einer DVB-T Antenne (Hama 44290) mit einem Spektrumanalysator (Advantest R3361) gemessen wurde. Diese Abbildung zeigt deutlich Spektralanteile außerhalb des VHF-2 Bereichs, welche ohne eine Filterung zu Aliasing führen und die Signalqualität stark beeinträchtigen würden.

Um VHF-2 mit dem ADC ohne vorheriges analoges Mischen auf eine Zwischenfrequenz (ZF) in den $\frac{f_s}{2}$-Bereich digitalisieren zu können, ist eine Bandpassunterabtastung nötig (Kapitel 2.3). Das bedeutet bei einem Abtasttakt von 80 MHz, dass sich der Spektralbereich von 80 MHz bis 120 MHz nach der Abtastung auf 0 MHz bis 40 MHz abbildet. Dies geschieht nur dann störungsfrei, wenn der VHF-2 Frequenzbereich vorher ausreichend selektiert wurde. Eine mögliche Filtercharakteristik für diese Selektion könnte jene aus Abbildung 3.13 sein.

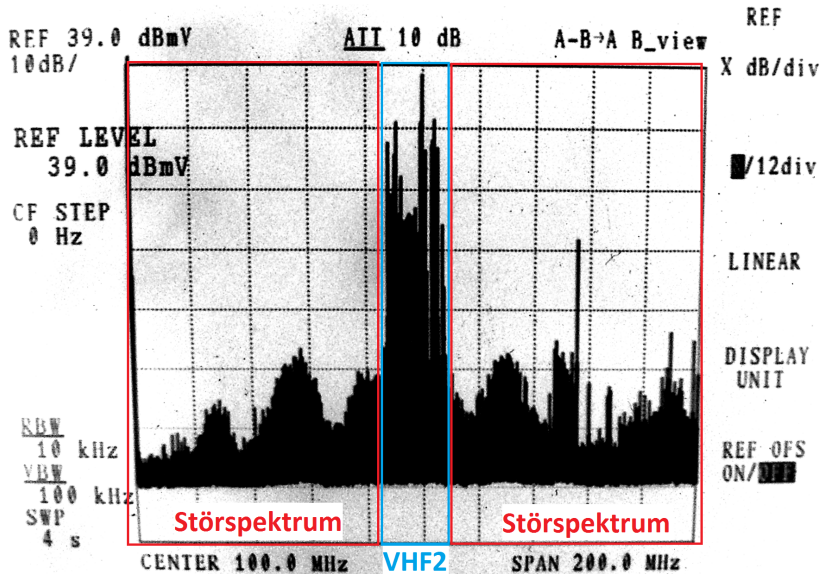

Abbildung 3.12: VHF-2 Spektrum gemessen mit einer Hama 44290 DVB-T Antenne

Hinter dem Filter erscheinen alle Spektralanteile unter 80 MHz und über 120 MHz ausreichend unterdrückt. Für die Filterflanken bedeutet dies einen linken Filterflankenbereich von $(87,5 - 80)\ MHz = 9,5\ MHz$ und einen rechten von $(120 - 108)\ MHz = 12\ MHz$. Es ist somit sinnvoll, die Mittenfrequenz in die VHF-2 Mitte, $\frac{108+87,5}{2}\ MHz = 97,75\ MHz$ zu legen und die Breite der Filterflanken kleiner als 9,5 MHz zu dimensionieren. Bei einem ADC SNR von ca. 75 dB [12] ist eine Flanke steiler als $\frac{75\ dB}{log(9,5)} \approx 76 \frac{dB}{decade}$ nötig, um mit dem ADC eine gute Störunterdrückung zu erreichen.

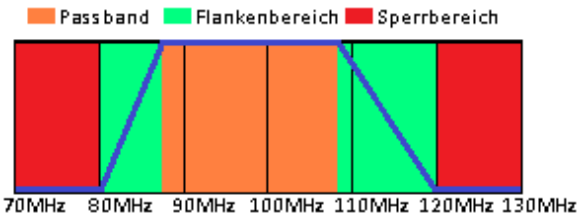

Abbildung 3.13: Analoge Filtercharakteristik (VHF-2-Bandpass)

Der gegebene ADC hat eine theoretische Auflösung von 16 Bit, welche für eine gute Signalqualität auch ausgenutzt werden sollte. Dies wird möglich, wenn das analoge Eingangssignal am ADC Eingang den Spannungshub (Fullscale, hier 2,25 V Peak-Peak oder 1,5 V Peak-Peak bei eingeschaltetem Programmable-Gain-Applifier[29] (PGA)) des ADCs voll ausnutzt. Es wäre ein analoger Breitbandverstärker für den VHF-2-Bereich nötig, der eine Pegelan-

[29] Option des LTC2206 ADC zur Aktivierung eines internen Vorverstärkers.

passung des Antennensignals auf Fullscale übernimmt. Aus dem Spektrum aus Abbildung 3.12 ist ein Signalpegel von 39 dBmV ($Upp = 10^{\frac{39\ dBmV}{20}} \cdot 1\ mV \cdot 2 \cdot \sqrt{2} \approx 252\ mV$) erkennbar. Bei eingeschaltetem PGA wäre eine Verstärkung von $A = 20 \cdot log \frac{1{,}5V}{252\ mV} \approx 15{,}5\ dB$ nötig, um Fullscale zu erreichen. Dieser Wert gilt nur für die verwendete Antenne (intern 39 dB Verstärkung) und der jetzigen Position der Antenne (Hamburg). Ohne diese 15,5 dB würde die theoretischen 16 Bit Auflösung des Signals auf theoretische $log_2(\frac{2^{16} \cdot 252\ mV}{1{,}5V}) = 13{,}4\ Bit$ sinken. Das menschliche Gehör ist jedoch nicht so sensibel diese Minderung an Auflösung zu registrieren. Für die Anforderungen dieser Arbeit ist dieses ausreichend.

Das zu realisierende Frontend wird sich aus den folgenden Komponenten zusammensetzen (Abbildung 3.14):

- **Antenne:** Eine handelsübliche aktive Hama TV- und Radio-Zimmerantenne (Hama 44290) mit einem integrierten regelbaren Verstärker bis 39 dB, zum Einsatz. VHF wird von dieser komplett abgedeckt.
- **Breitbandverstärker:** Auf einen weiteren Breitbandverstärker neben dem regelbaren 39 dB Verstärker der Antenne wird verzichtet, da kein hochqualitativer Audioempfang gefordert ist. Weitere 15 dB Verstärkung könnten mit handelsüblichen DVB-T Verstärkern erreicht werden.
- **Bandselektionsfilter:** Fertige Bandfilter im Handel die genau die gewünschten Charakteristik zeigen, sind schwer auszumachen, zu teuer oder durch eine Mindestabnahme beschränkt. Somit wird ein eigener SMD-Filter mit der Software *Filter Solutions 2011* von Nuhertz Technologies[30] entworfen werden.

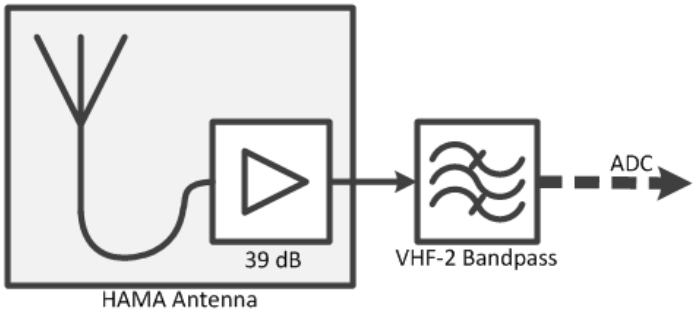

Abbildung 3.14: Übersicht des analog Frontends

[30] http://www.nuhertz.com/

3.6 Signalverarbeitung

Eine FM-Demodulation wird in diesem SDR benötigt, um den Demonstrator, ein FM-Rundfunkradio, zu realisieren. Mögliche Arten, die Demodulation durchzuführen, werden ein Teil dieses Abschnitts sein. Für jede der Demodulationsmöglichten wird das zu demodulierende Signal auf einer Zwischenfrequenz erwartet und es wird angenommen, dass das Signalspektrum vorher über einen Bandfilter selektiert wurde.

3.6.1 FM-Signal

Die Informationen in einem FM-modulierten Signal liegen in seiner momentanen Frequenzdifferenz zu einer Mittenfrequenz. Ist zum Beispiel ein Signal auf eine Frequenz X moduliert, ist ein positiver Pegel des Ursprungssignals in einer Frequenz größer als X abgebildet. Ein negativer Pegel dann in einer Frequenz kleiner als X. Abbildung 3.15 zeigt hierzu deutlich, wie sich das Ursprungssignal im modulierten Signal wiederfindet.

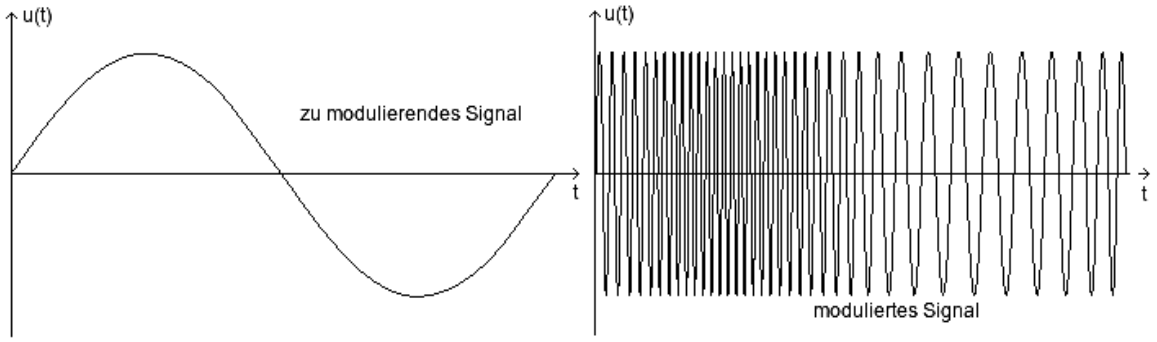

Abbildung 3.15: Signalbeispiel zur FM-Modulation

Mathematisch lässt sich ein FM-moduliertes Signal aus einer sinusförmigen Schwingung mit einer variablen Momentanfrequenz darstellen. Verdeutlicht werden kann dies durch einen rotierenden komplexen Zeiger in der Ebene. Die Geschwindigkeit mit der sich dieser Zeiger bewegt, gibt Auskunft über die Frequenz. Der Phasenwinkel gibt den Weg an, den der Zeiger bei seiner Drehung zurückgelegt hat. Somit kann die Phase des modulierten Signals als Frequenz mal Zeit definiert werden. Für differenziell kleine Frequenzänderungen ist die Phasenänderung, zur Phase der Mittenfrequenz, definiert als $\Delta\omega(t) = k \int_0^t u(\tau)d\tau$, wobei $u(\tau)$ das Ursprungssignal darstellt und k die Modulatorkonstante mit der Einheit $\frac{Hz}{V}$. Die Modulatorkonstante steuert den Modulationshub, also die maximale Frequenzänderung, die im FM-Signal auftritt. Das gesamte modulierte Signal wäre dann $u_{mod}(t) =$

$c \cdot sin(\omega_m(t) + \Delta\omega(t)) = c \cdot sin(\omega_m(t) + k \int_0^t u(\tau)d\tau)$. $\omega_m(t)$ ist hierbei die Phase der Mittenfrequenz und c die maximale Amplitude des Signals.

3.6.2 FM-Demodulation

Es könnten beispielsweise drei verschiedene Demodulationsarten eingesetzt werden, um ein FM-moduliertes Signal wiederzugewinnen:

1. **Differenzieren und AM-Demodulieren**: *Differenzieren und AM-Demodulieren* beruht auf der Tatsache, dass ein höher frequentes Sinussignal eine höhere Steigung im Nulldurchgang durch die x-Achse aufweist. Durch eine Differentiation des modulierten Signals kann ein Signal erzeugt werden, welches der AM-Modulation des Ursprungssignals entspricht. Um aus einem AM-modulierten Signal das Ursprungssignal wieder zu gewinnen, kann eine Hüllkurvendetektion eingesetzt werde (Abbildung 3.16).

Abbildung 3.16: FM-Demodulation mit Differenziation und Hüllkurvendetektion

2. **Demodulieren mit einer Phase Locked Loop**: Für diese Art der Demodulation wird eine Phase-Locked-Loop (PLL) eingesetzt. In einer PLL arbeitet ein Phasenkomparator, der die Phasendifferenz (E) des Eingangssignals (Y) zu einer Rückkopplung des Ausgangssignals (f_{out}) ermittelt. Diese Differenz gibt an, in wieweit die Phasen und somit auch die Frequenzen von Eingangs- und Ausgangssignal abweichen. Das Signal E kann genutzt werden, um mit Hilfe eines Voltage-Controlled-Oscillators (VCO) und eines vorgeschalteten Schleifenfilter, zur Festlegung des Regelverhaltens, eine Regelung der Ausgangsphase zu erreichen. Dieses bewirkt, dass sich vorne am Phasenkomparator eine Phasendifferenz von Null einpendelt. Ist der Regelkreis eingeschwungen und am Eingang (Y) der PLL liegt ein frequenzstabiler Sinus an, würde C zu Null werden. Steigt die Eingangsfrequenz jedoch an, wird eine positive Abweichung detektiert und C wird ebenso positiv. Bei einer niedrigeren Frequenz wird C negativ. Das Signal C ist somit das FM-demodulierte Signal von Y. Um eine solche Demodulation durchführen zu können, ist das FM-modulierte Signal auf eine Zwischenfrequenz zu bringen, die in dem jeweiligen PLL-Arbeitsbereich liegt.

3. **Frequenzzähl-Demodulation**: Durch Zählen von Nulldurchgänge auf der y-Achse im FM-modulierten Signal, kann ebenfalls eine FM-Demodulation erreicht werden (Abbildung 3.18). Ist das modulierte Signal digital, kann durch Abzählen der Samples von

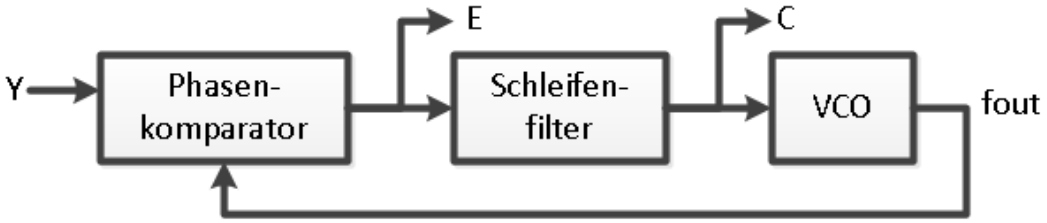

Abbildung 3.17: Aufbau einer PLL

einem Nulldurchgang zum Anderen die Länge der halben Periode ermittelt werden. Entspricht die Momentanfrequenz des FM-modulierte Eingangssignalspektrum genau der definierten Zwischenfrequenz, entspricht die Anzahl der Samples zwischen den Nulldurchgängen dem Nullpegel des demodulierten Signals. Steigt die Frequenz über die Zwischenfrequenz, sinkt der Abstand zwischen den Nulldurchgängen. Bei einer sinkenden Frequenz wird die Sampleanzahl größer. Die Information des Ursprungssignals liegt somit in der Anzahl an Samples zwischen den Nulldurchgängen. Werden die Sampleanzahlen als Signal aufgefasst und von dessen Samples je die Sampleanzahl für die Zwischenfrequenz abgezogen, entspricht dies dem demodulierten Signal.

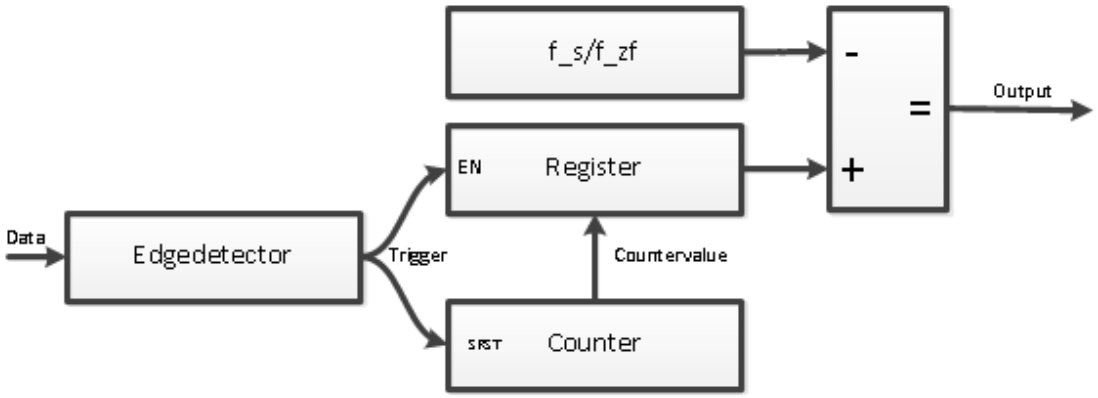

Abbildung 3.18: Aufbau eines Systems mit der Frequenzzähl-Demodulation für digitale Signale

Diese letzte Methode ist einfach und schnell in einer Programmiersprache und auch in digitaler Hardware umsetzbar. Aus diesem Grund wird die *Frequenzzähl-Demodulation* in dieser Arbeit für die FM-Demodulation des Mono-Audiosignals eines FM-Rundfunksenders eingesetzt werden.

3.6.3 Signalverarbeitungseinheit

Für eine Signalverarbeitung ist MATLAB gut geeignet, da dieses Programm bereits alle Hilfsmittel implementiert. Eine Verarbeitung der Daten von MATLAB in Echtzeit wäre wünschenswert. Mit dem *MATLAB RealTime Windows Target*[31] ist laut [19] nur eine Abtastfrequenz von 5 kHz für eine Echtzeitverarbeitung garantiert. Dennoch wurde in C# ein UDP/IP-Server programmiert, der testweise einen 5 MB/s Datenstrom aus Dummy IQ-Daten verschickt (dies entspricht der Geschwindigkeit des bestehenden Datenstroms. Siehe 3.1 oder [29]). Mit MATLAB und Simulink wurde dann versucht, diese Daten in Echtzeit zu empfangen und darzustellen. Es hat sich leider bestätigt, dass der RealTime-Workshop und auch der MATLAB-Embedded Coder[32] nicht für eine Echtzeitverarbeitung von 5 MB/s IQ-Daten geeignet ist.

Ein Programm wie eines in C# geschrieben arbeitet deutlich schneller als es der Fall mit MATLAB ist. Ein Testprogramm mit dieser Programmiersprache hat gezeigt, dass dieses das Potential besitzt, eine FM-Demodulation in Echtzeit auszuführen. Aus diesem Grund wird eine C#-Lösung die Echtzeitkommunikation mit dem SDR übernehmen (Kapitel 3.7).

3.7 SDR-Client

Das eigentliche SDR ist soweit konzeptioniert. Allerdings ist eine UDP-basierte Kommunikation mit dem PC vorgesehen. Hierfür wird der SDR-Client als PC-Software entwickelt werden. Dieser ermöglicht eine Kommunikation mit dem plattformunabhängigen SDR-System über UDP. Realisiert werden wird dieser Client mit der Programmiersprache C# in Verbindung mit der Entwicklungsumgebung *Visual C# 2010 Express* von Microsoft. C# ist eine Objektorientierte Programmiersprache, die zur Ausführung des Bytecode-basierten Kompilats (Common-Language-Infrastructure[33] (CLI) Executable) eine Runtime-Umgebung benötigt. Hiermit ist eine plattformunabhängige Programmierung mit C# möglich. Eine Runtime-Umgebung existiert für Linux Systeme mit dem Mono Projekt[34] und für Windows ist diese im .NET-Framework integriert.

Der SDR-Client wird sich in drei Teile gliedern: SDR-Class, MAT-File-Writer und FM-Radio (Abbildung 3.19).

Der erste Teil (SDR-Class) ist eine C#-Klasse, die das per UDP angeschlossene SDR für den weiteren Quellcode abstrahiert. Diese wird Methoden stellen, die es erlauben, zu prüfen, ob

[31] Ein Teil von MATLAB zur Echtzeit Simulation auf einem Windowssystem.

[32] MATLAB Toolbox zur Generation von nativem Maschinencode aus MATLAB-Skripts oder Simulink-Simulationen.

[33] Ist ein offener Standard von Microsoft der die Ausführbaren Dateien beschreibt, die mit dem .NET-Framework ausgeführt werden.

[34] Mono ist eine Runtime-Umgebung für das Ausführen von CLI-Executables unter Linux.

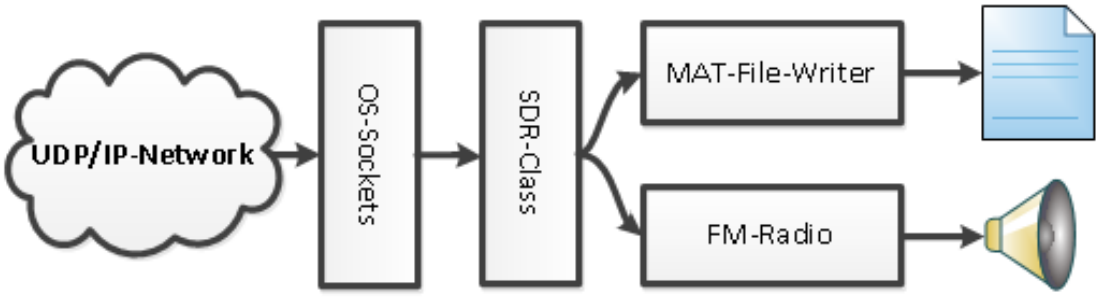

Abbildung 3.19: Übersicht des C# SDR-Clients

das SDR an einem gegebenen UDP-Endpunkt verfügbar ist. Zudem wird eine Administrierung sowie der Echtzeit-Empfang der Daten möglich sein. Hierfür ist der SDR-Klasse die Befehlsstruktur des DDC-Servers bekannt und kann diesen somit einfach über UDP administrieren.

Für eine Verarbeitung von VHF-2 Signalen in MATLAB/Simulink sind Daten vom SDR von Nöten. Da eine direkte Implementierung einer Echtzeit-Schnittstelle in MATLAB nicht möglich ist, werden die vom SDR-Client aufgenommen Daten in einer MAT-Datei[35] abgelegt. Der zweite Teil des SDR-Clients ermöglicht diese Aufnahme der Daten vom SDR. Zur Speicherung in einer MAT-Datei wird die Bibliothek *CSMatIO*[36] genutzt, welche frei erhältlich ist. Im SDR-Client kann eine Konfiguration über eine GUI stattfinden, mit welcher das SDR durch die SDR-Klasse konfiguriert und in Betrieb gesetzt wird. Der SDR-Client sorgt dann für eine zeitbegrenzte Aufnahme der Daten in eine gewünschte MAT-Datei in Echtzeit.

Live FM-Rundfunk zu empfangen und hörbar zu machen, ist die Aufgabe des dritten Teils des SDR-Clients. Er nutzt die SDR-Klasse, um Zugriff auf das SDR zu erhalten. Zur Demodulation wird die Frequenzzähl-Demodulation genutzt (siehe Analyse Kapitel 3.6). Diese ist einfach zu implementieren und bietet trotzdem ausreichende Signalqualität. Zum hören eines FM-Rundfunksenders kann in der GUI ein FM-Sender ausgewählt werden, sodass der SDR-Client das SDR über die SDR-Klasse einstellen kann. Der empfangene Datenstrom wird dann direkt in Echtzeit demoduliert und auf dem PC-Lautsprecher hörbar gemacht. Eine Funktion zum Tunen[37] wird das manuelle Suchen von Sendern ermöglichen.

[35] Ein MATLBA-Dateiformat, welches in MATLAB importiert werden kann, um Variablen dem Workspace hinzu zu fügen

[36] C# Bibliothek zum erzeugen von MAT-Dateien - http://www.mathworks.com/matlabcentral/fileexchange/16319

[37] Schnelles Ändern der Sender-Frequenz, um Sender zu suchen

3.8 Zusammenfassung

Das zu realisierende SDR setzt sich aus den Teilsystemen der vorigen Unterkapitel (3.1, 3.2, 3.3, 3.4, 3.5, 3.6 und 3.7) zusammen und wird im nächsten Kapitel 4 realisiert werden. Ein Gesamtblick auf das System ist hierfür sehr hilfreich. Die folgende Abbildung 3.20 zeigt noch einmal die Teilsysteme angeordnet nach dem Datenfluss, welcher von der Antenne bis zum verarbeitenden System im PC läuft.

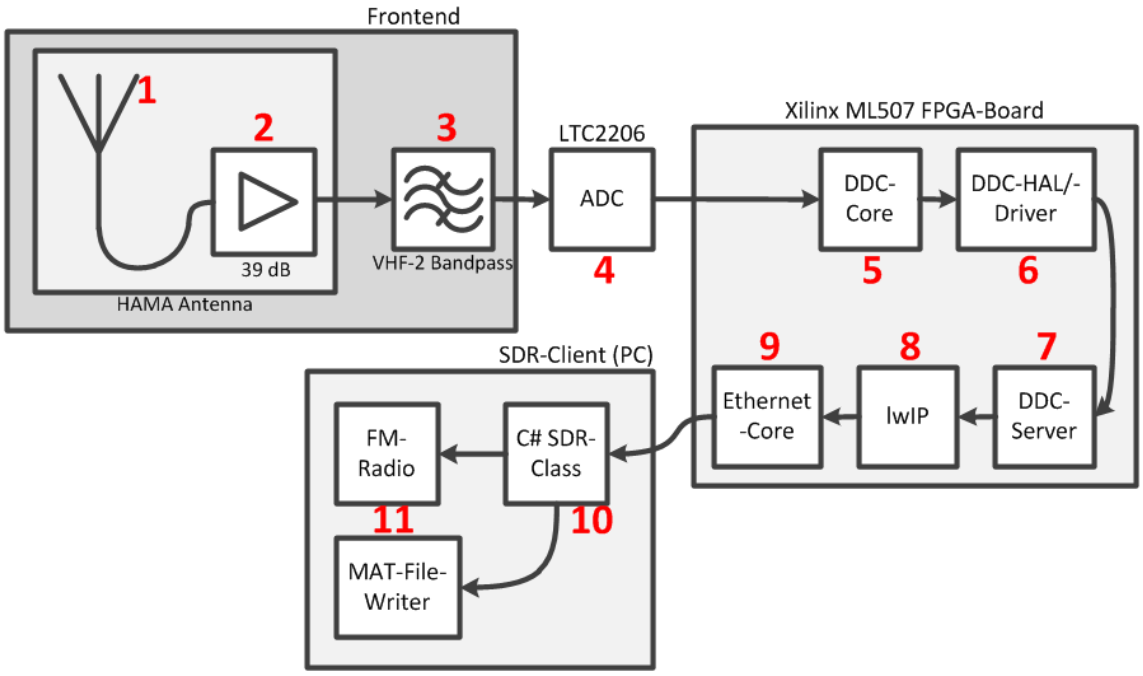

Abbildung 3.20: Überblick des zu realisierenden Gesamtsystems

Die Funktion der einzelnen Teile wurde bereits erklärt. Dennoch soll dieses hier nach dem Fluss der Daten noch einmal grob zur Übersicht geschehen:

1. **Antenne:** Eine Hama DVB-T-Antenne für VHF-2 wandelt die Funksignale in elektrische zur Weiterverarbeitung.
2. **Verstärker:** Es wird der regelbare 39 dB Verstärker der DVB-T-Antenne genutzt, um eine Anpassung des Signalpegels an den ADC zu erreichen.
3. **Bandpass:** Eigenentwurf eines passiven SMD-Bandpasses für den VHF-2-Frequenzbereich. Selektiert das VHF-2-Band zur aliasingfreien Digitalisierung auch bei Unterabtastung.

4. **ADC:** LTC2206 ADC mit 16Bit @80MHz für die Wandlung von der analogen in die digitale Domäne (Unterabtastung).
5. **DDC-Core:** Konvertierung des digitalen VHF-2-Bandes in das Basisband mit Dezimierung des Datenstroms. Der DDC-Core ist Bestandteil (Peripherie) eines Embedded Systems im FPGA.
6. **DDC-HAL/Treiber:** Stellt einen HAL und darauf arbeitenden Treiber, für den DDC-Core im Embedded System. Dient zur einfachen Bedienung durch die Firmware.
7. **DDC-Server:** Firmware, welche es ermöglicht den DDC-Core, beziehungsweise das SDR, von außen zu steuern.
8. **lwIP:** Implementiert einen Software TCP/IP-Stack (lwIP) zur Kommunikation über UDP mit dem PC.
9. **Ethernet-Core:** Verschickt und empfängt Ethernet-Pakete vom lwIP-Stack.
10. **C#-SDR-Class:** Abstrahiert, das über UDP erreichbare SDR, mit einer C# Klasse auf dem PC.
11. **FM-Radio und MAT-File-Writer:** Nutzt die SDR-Klasse, um ein FM-Radio mit Echtzeit-Demodulation und Signalausgabe auf dem PC-Lautsprecher zu realisieren. Weiter kann eine MAT-Datei als Ziel der empfangenen SDR-Rohdaten genutzt werden, sodass diese Daten mit einem Doppelklick auf die Datei in MATLAB importiert werden können. Eine GUI ermöglicht eine benutzerfreundliche Bedienung.

4 Realisierung

Die hier beschriebene Realisierung basiert auf dem konzeptionierten System des vorigen Kapitels 3. Um eine Realisierung des SDR zu erhalten, ist eine schrittweise Vorgehensweise nötig. Jeder dieser abgeschlossenen Schritte wurde getestet, um sicherzustellen, dass später nutzende oder aufbauende Teilsysteme immer mit einem getesteten Teilsystem arbeiten können. Der mögliche Fehlerradius wird hierdurch stark eingegrenzt.

4.1 Verifikation des bestehenden DDC-IP-Cores

Bevor der DDC-Core eingesetzt wird, soll dieser auf seine Funktion geprüft werden. Es soll sichergestellt sein, dass er für spätere Teilsysteme einsatzbereit ist. Für den Test wird das bestehende USB-basierte System (siehe [29] oder Kapitel 3.1) herangezogen.

Für diesen Test wurde im Signalgenerator ein Sinus mit 3,0001 MHz eingestellt. Mit Hilfe des bestehenden Testsystems ([29]) und dem MATLAB-Skript aus Anhang *6* wurde der DDC mit einer Mischerfrequenz von 3 MHz konfiguriert. Dieses bewirkt, dass der 3,0001 MHz Sinus auf 100 Hz heruntergemischt in MATLAB verfügbar ist. 100 Hz sind von einem Menschen hörbar, sodass eine WAV-Datei aus den empfangen Daten generiert wurde. Mit Audacity[1] ergab sich dann das Bild aus Abbildung 4.1. Ein hörbares Sample dieser WAV-Datei kann dem Anhang *7* entnommen werden .

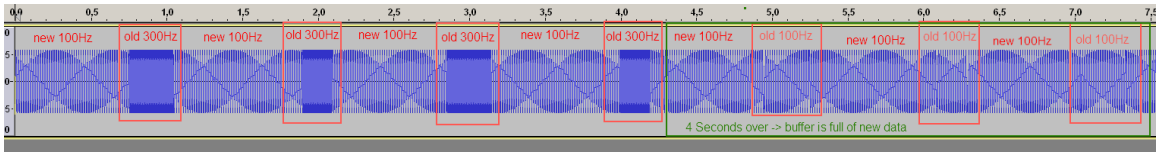

Abbildung 4.1: 30 Sekunden eines 300 Hz Sinus Testsignal, aufgenommen mit dem bestehenden System

Zu erkennen sind Störungen im Sekundenabstand, die nicht dem 100 Hz Sinus entsprechen. Der DDC-Core arbeitet mit vier Speicherblöcken in der FAT-Datei, welche durch den

[1] Programm um Audio-Dateien darzustellen und zu verarbeiten. http://audacity.sourceforge.net/

USB-Massenspeicher zugänglich sind. Ein Speicherblock entspricht einer Sekunde von IQ-Datensätzen. Es liegt somit nahe, dass ein Problem mit dem Auslesen der Daten besteht. Auffällig ist auch, dass sich das sekündliche Fehlmuster nach vier Sekunden auf einen phasenverschobenen Sinus mit 100 Hz ändert. Nach einigen Messungen mit verschiedenen Sinussignalen hat sich bestätigt, dass der bestehende DDC-USB-Reader Fehler bei der Positionierung des Lesezeigers erzeugt und in der Embedded-Firmware ein Segmentzähler beim Start des DDC-Cores nicht auf Null zurück gesetzt wird. Der DDC-USB-Reader liest somit Daten aus einem falschen Datenbereich, der vom DDC-Core noch beschrieben wird. Nach vier Sekunden ist die FAT-Datei einmal im Kreis mit dem 100 Hz Sinus überschrieben worden, sodass Teile des 100 Hz Signals mit Phasensprüngen an den Fragmentgrenzen erscheinen.

Diese Fehler in der bestehenden Software [29] haben dazu geführt, dass die bestehende Firmware und der bestehende DDC-USB-Reader umgeschrieben bzw. neu programmiert wurden. Die neue Firmware ist nun, mit dem zu ISE 12.4 portierten EDK-Projekt, im Anhang *8* zu finden. Der bestehende DDC-USB-Reader wurde komplett neuentwickelt und als CMEX[2]-Implementierung (HawSdrUsbClient) in MATLAB integriert. Dieses ermöglicht nun das Auslesen der DDC-Daten direkt in den MATLAB-Workspace ohne den Umweg einer temporären Datei zu nehmen, wie es vorher der Fall war. Der Quellcode zu dieser CMEX-Lösung ist im Anhang *9* zu finden.

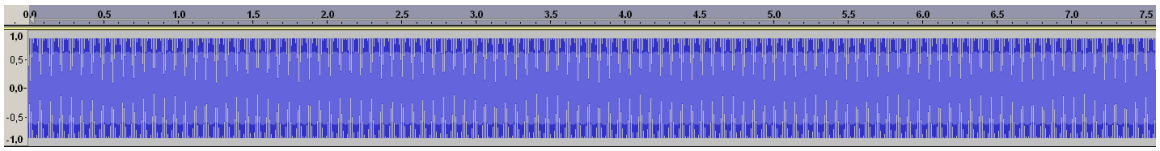

Abbildung 4.2: Sinus Testsignal ohne Fehler, aufgenommen mit modifizierter Firmware und CMEX-Client

Die Ergebnisse aus Abbildung 4.2 zeigen, wie der Sinus nach der Wandlung mit reparierter Firmware und der CMEX-Lösung zur Kommunikation mit dem DDC aussieht. Das MATLAB-Testskript hierzu ist im Anhang *10* einzusehen.

Mit der Anpassung der bestehenden Firmware und der Neuentwicklung des DDC-USB-Readers als CMEX-Implementierung, besteht nun ein verifiziertes System. Mit diesem ist der Zugriff auf den DDC und eine aufbauende Entwicklung fehlerfrei in den Grenzen der USB-Lösung möglich.

[2]CMEX ist ein MATLAB Dateiformat das nativen Code der PC Plattform enthält. Hierdurch ist die Ausführung des Codes deutlich schneller als übliche M-Skripte die interpretiert werden müssen.

4.2 Analoges Frontend

Als Antenne wurde eine handelsübliche Hama DVB-T-Antenne mit integriertem regelbaren 39 dB Verstärker eingesetzt. Die Analyse aus Kapitel 3.5 hat gezeigt, dass eine weitere Verstärkung nicht nötig ist. Somit wird in diesem Abschnitt auf die Realisierung des Bandpassfilters, für eine aliasingfreie Bandpassunterabtastung ($f_s = 80MHz$) des VHF-2 Frequenzbereiches, eingegangen.

4.2.1 Entwurf

Die benötigte Filtercharakteristik (Abbildung 3.13) ist aus der Analyse im Kapitel 3.5 hervorgegangen, welche theoretisch den Anforderung gerecht wird. Mit der Software *Filter Solutions 2011* ist ein Filter entworfen worden, der mit der benötigten, theoretischen, Filtercharakteristik zu vereinbaren ist. Das bedeutet, der Passbandbereich erstreckt sich von 87,5 MHz bis 108 MHz bei einer Passbanddämpfung von maximal 0,6 dB. Die Spektralanteile über 120 MHz und unter 80 MHz, bedingt durch die Unterabtastung mit $f_s = 80MHz$, sind mit einer minimalen Sperrdämpfung von 27 dB dimensioniert. Im Eingangsspektrum (Abbildung 4.3) ist der Spektralanteil bei 80 MHz mit ca. 50 dB Abstand zum Maximum des VHF-2 Spektrums zu erkennen. Mit der konfigurierten Dämpfung von 27 dB steigt dieser Abstand auf ca. 77 dB, welches für eine aliasingfreie Unterabtastung mit $f_s = 80MHz$ und eine theoretischen ADC SNR von ca. 75 dB [12] ausreichend ist. Gleiches gilt für den Spektralanteil bei 120 MHz. Abbildung 4.3 zeigt, welche Spektralbereiche durch das Bandpassfilter eine Dämpfung erfahren werden. Bedingt durch die geplante BNC-Verkabelung, wurden die Ein- und Ausgangswiederstände auf 50Ω dimensioniert.

Ausgelegt wurde der Filter als *Lumped Filter*[3] siebter Ordnung mit einer Butterworth-Charakteristik, so dass dieser diskret aus passiven Bauelementen aufgebaut werden kann und der VHF-2 Bereich (Passband) möglichst wenig Ripple aufweist. Der Schaltplan des mit *Filter Solutions 2011* dimensionierten Filters ist der Abbildung 4.4 zu entnehmen. Alle Bauteile wurden mit Abweichungen (Kapazität und Induktivität) von 5% angegeben, so dass bereits vorher Abweichungen von der Theorie erkennbar sind. Ein solcher Vergleich, von der idealen Charakteristik zur der mit 5% abweichenden Bauteilen (reale Bauteile), ist der Abbildung 4.5 zu entnehmen. Eine größere Version des Bildes ist mit mehr Details und weiteren Ergebnissen von *Filter Solutions 2011* dem Anhang *18* entnehmbar.

Der Amplitudengang (Abbildung 4.5) zeigt einen um ca. 5 MHz verschobenen Frequenzgang des Filters mit realen Bauteilen. Eine Umsetzung des Filters in SMD-Hardware wird zeigen, wie sich dieses auf die Realisierung auswirkt und ob dieses Filter für diesen Zweck der Arbeit nutzbar ist.

[3] Eine passives Filter bestehend aus passiven RCL-Schaltungen.

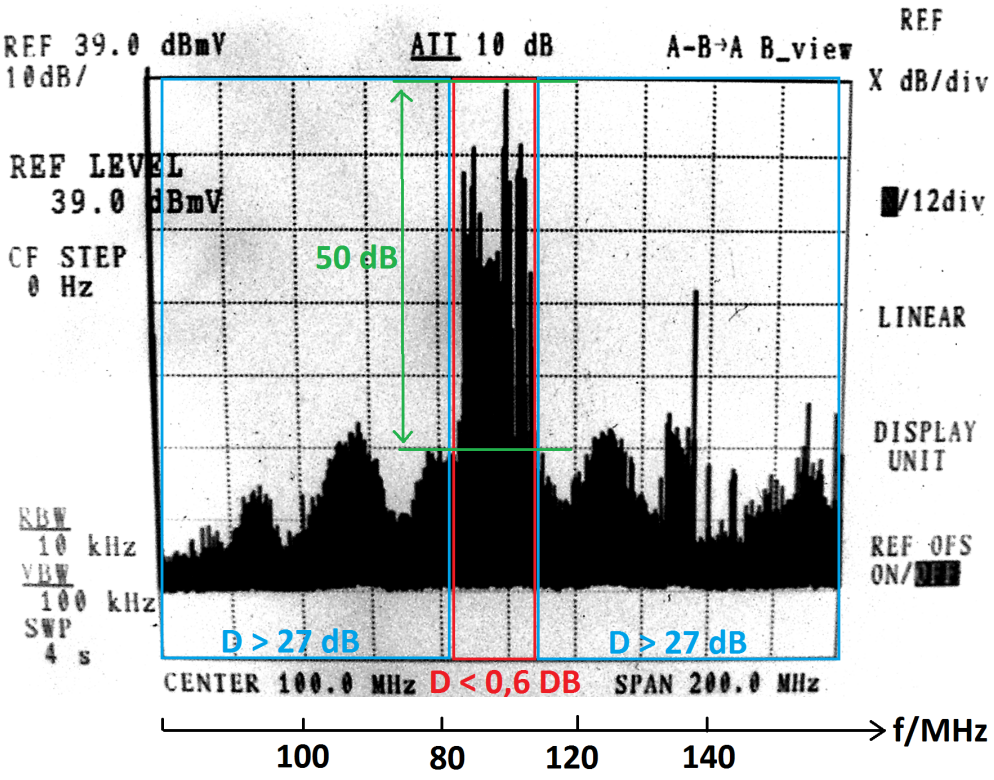

Abbildung 4.3: VHF-2 Spektrum hinter der Antenne mit angaben der gewünschten Dämpfung durch den Bandpassfilter

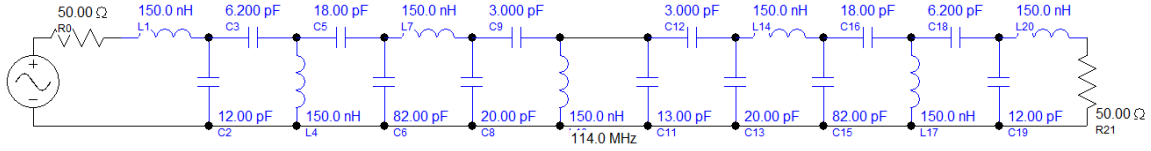

Abbildung 4.4: Schaltplan des realisierten Bandpassfilters

4.2.2 Umsetzung

Eine Vermessung mit einem Vektoranalysator (Rohde & Schwarz Vector Network Analyzer ZVRL) ermöglichte, zu prüfen, wie das, mit SMD-Bauteilen umgesetzte, Filter zu der optimalen Charakteristik passt. Diese Messung zeigte im Amplitudengang einige Abweichungen (Abbildung 4.6 und Tabelle 4.1):

Diese Abweichungen sind nicht zu vernachlässigen, jedoch für diesen Zweck der Arbeit nicht zu groß. Ein Demonstrator soll lediglich zeigen, dass das System echtzeitfähig ist. Hierfür ist nur mindestens ein Rundfunksender nötig. Mit diesem Filter ist der Empfang bis 100 MHz

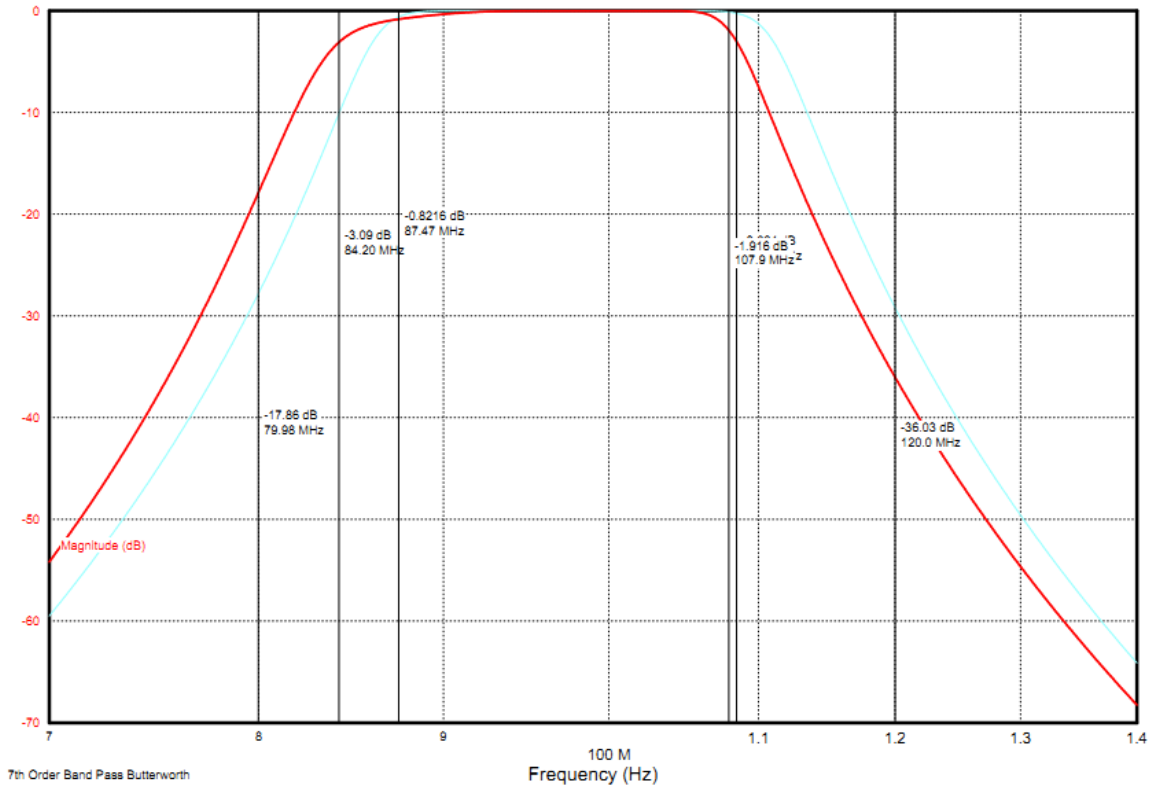

Abbildung 4.5: VHF-2 Spektrum hinter der Antenne. Rot: Bauteile mit Abweichung. Cyan: Ideale Bauteile

möglich. Dieser Bereich enthält ca. 40 % der zuvor gemessenen Sender im VHF-2 Bereich (in Hamburg). NDR Kult auf 99,2 MHz wäre der letzte im Frequenzbereich.

Die gewünschten Stopbandfrequenzen wurden nicht erreicht. Diese sind jedoch essentiell Wichtig, um Aliasing bei der Bandpassunterabtastung zu unterdrücken. Für die Rechteseite im Spektrum (siehe Amplitudengang im Punkt 2) zeigt sich, bei den genannten Abweichungen, kein Problem, da dort die Dämpfung gestiegen ist. Bei 80 MHz (siehe Amplitudengang Punkt 1) sind es, anstatt 27 dB, nur 7 dB. Im Eingangsspektrum (Abbildung 4.3) ist der 80 MHz Spektralanteil mit ca. 50 dB erkennbar, welches mit den erreichten 7 dB, 57 dB ergibt. 57 dB sind bei einem theoretischen ADC SNR von 75 dB [12] jedoch für diesen Demonstrator ausreichend.

Die Vermessung mit dem Vektoranalysator hat zudem eine Abweichung des Filter-Eingangswiderstands gezeigt. Dieser liegt im nutzbaren Bereich (87,5 MHz bis 100 MHz) mit maximal 16 % unter 50Ω. Auch diese Abweichung ist für den demonstrativen Zweck ausreichend. Weitere Details, wie ein Vergleich der Smithdiagramm sind dem Anhang *18* zu entnehmen.

	Gewünscht	Erreicht	Abweichung
Passbandbreite	20 MHz	17 MHz	-3 MHz
Passbandfrequenz (li.)	87,5 MHz	83 MHz	-4,5 MHz
Passbandfrequenz (re.)	108 MHz	100 MHz	-8 MHz
Stopbandfrequenz (li.)	80 MHz	71 MHz	-9 MHz
Stopbandfrequenz (re.)	120 MHz	117 MHz	-3 MHz

Tabelle 4.1: Abweichungen des realisierten Bandpasses

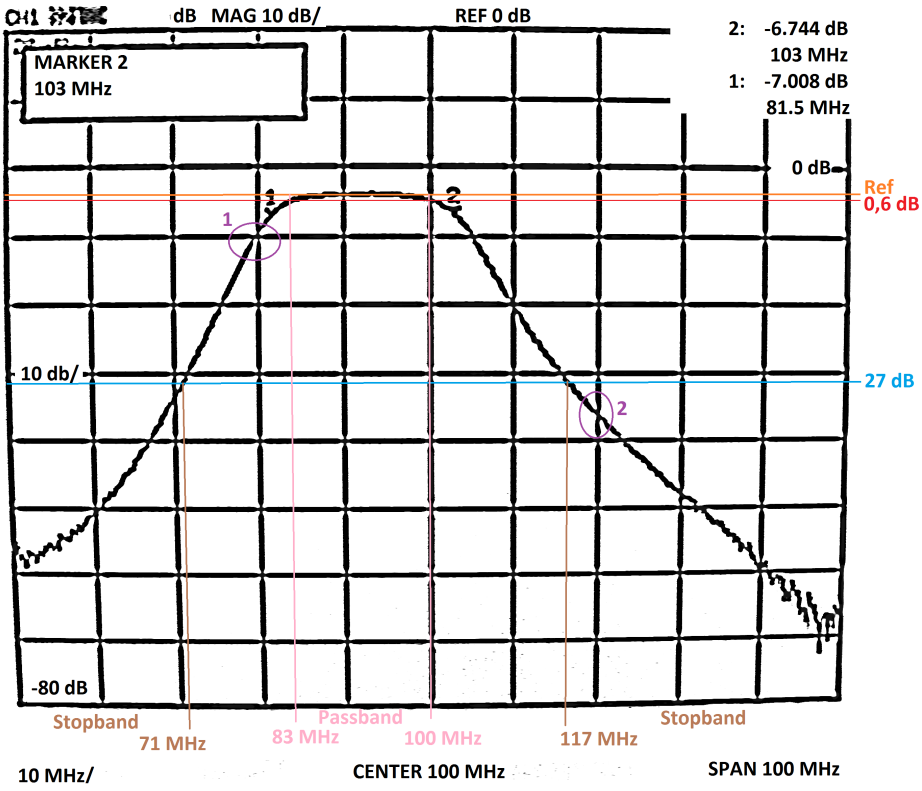

Abbildung 4.6: Amplitudengang des mit SMD-Bauteilen umgesetzten VHF-2 Bandpasses

Mit dieser Filterumsetzung ist das analoge VHF-2 Frontend komplett und kann geschlossen mit dem bestehenden System getestet werden.

4.2.3 Verifikation mit PScope

Das PScope ist eine Software zur Visualisierung der Samples des hier verwendeten ADC (LTC2206). Damit dies möglich ist, wird das in [29] verwendetet LT DC918C Evaluations-

Board mit einem *USB Data Acquisition Controller* (LT DC718C [14]) verbunden und eine USB-Verbindung mit dem PC herstellt. Durch diese Hardware in Verbindung mit dem PScope ist es nun möglich, genau die Daten zu visualisieren, die der DDC-Core für die Verarbeitung erhalten würde. Dieses ist somit die optimale Testumgebung für das analoge Frontend in Verbindung mit dem ADC des SDR. Es wurde je ein FFT-Spektrum ohne und mit Bandpassfilter (Abbildung 4.7) aufgenommen. Im oberen Bild (ohne Filter), ist sehr stark Aliasing zu erkennen. Das untere (mit unterdrückten Störbändern), zeigt die deutliche Verbesserung im Signal-Störabstand des VHF-2-Bereiches um ca. 100 %.

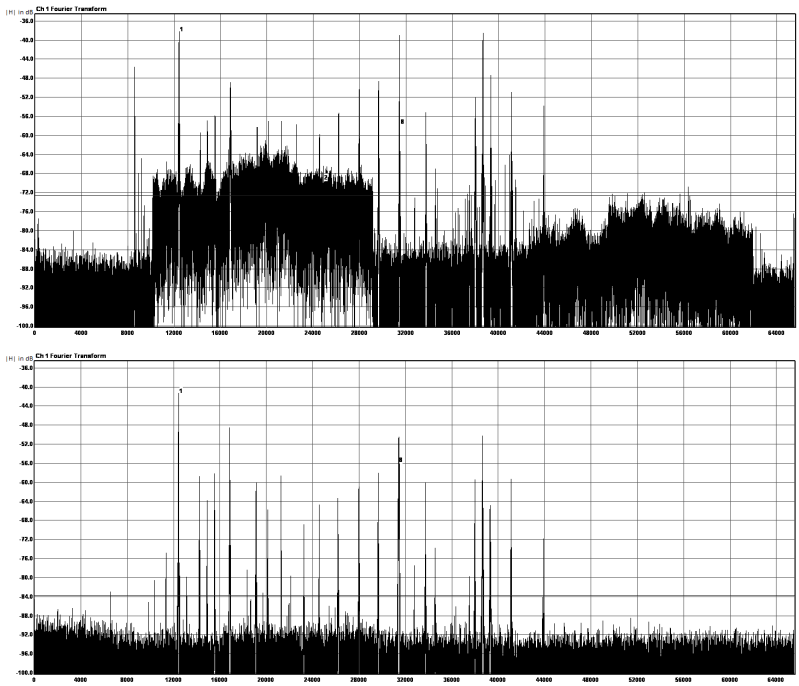

Abbildung 4.7: FFT-Spektren (N=131072 @80 MHz) des empfangenen VHF-2 Frequenzbandes ohne (oben) und mit (unten) Bandselektionsfilter

Abschließend ist in Verbindung mit dem modifizierten DDC-Testsystem des vorigen Kapitels die Antenne mit dem realisierten Filter betrieben worden. NDR90.3 auf 90,3 MHz diente als Testobjekt. Der DDC wurde mit einer Mischerfrequenz von $(90,3 - 80 - 0,1)\ MHz = 10,2\ MHz$ konfiguriert, sodass die Mitte des Rundfunk-Senders auf 100 kHz im Spektrum erscheinen muss. In Abbildung 4.8 ist genau dieses zu erkennen. Das FM-Spektrum liegt in der Mitte des Tiefpassbereiches und könnte so ideal FM demoduliert werden. Das MATLAB-Skript zu diesem Test ist im Anhang *10* zu finden.

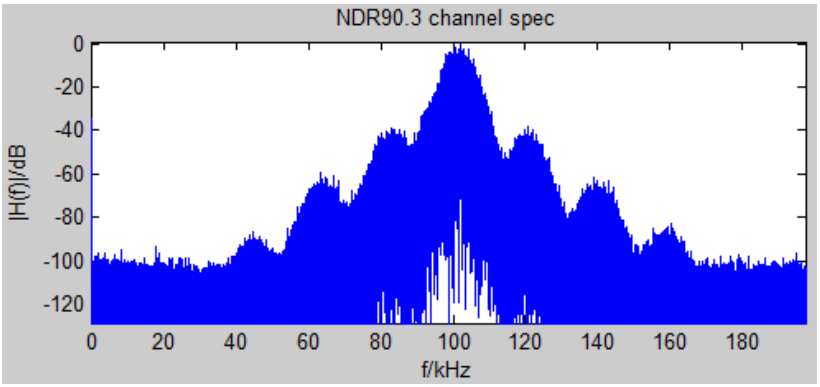

Abbildung 4.8: Mit MATLAB empfangenes FM-Rundfunksignal (90,3 MHz um 100 kHz zentriert)

4.3 FM-Demodulation

Vorgesehen für die zu realisierende FM-Demodulation wird die Frequenzzähl-Demodulation aus Kapitel 3.6. Für diese wird dieses Kapitel zeigen wie ein Software-Algorithmus aussieht und dass dieser funktioniert.

4.3.1 Entwurf des Algorithmus

Wenn f_s die Abtastfrequenz ($f_s = \frac{80\ MHz}{64} = 1,25\ MHz$) und f_m die FM-Spektrum-Mitte ($f_m = 100\ kHz$) ist, dann ist N_m die Anzahl von Samples zwischen zwei Nulldurchgängen die dem Pegel 0 entspricht ($N_m = \frac{1,25\ MHz}{2 \cdot 100\ kHz} = 6,25\ Samples$). Abstände von Nulldurchgängen größer als N_m entsprechen einem positiven Pegel und Werte darunter einem negativen. N_m ist somit der Mittelwert, welcher von jedem ermittelten Abstand abgezogen werden muss, um ein gleichanteilfreies Audiosignal zu erhalten. Der entwickelte Algorithmus zur Realisierung kann dem Flussdiagramm aus Abbildung 4.9 entnommen werden.

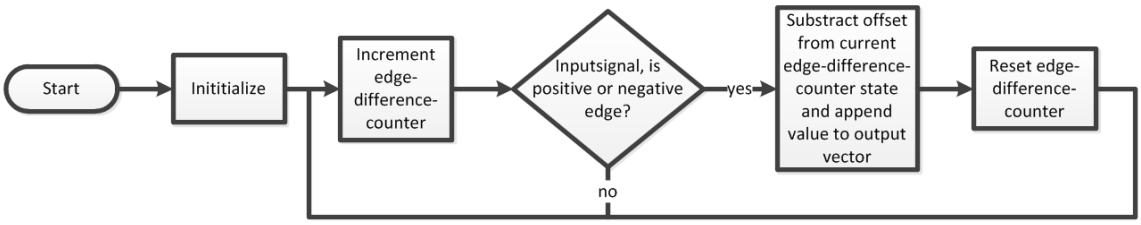

Abbildung 4.9: Flussdiagramm zur FM-Demodulation nach dem Frequenzzähl-Demodulation

4.3.2 Verifikation mit MATLAB

Das zu verarbeitende Spektrum wird für diesen Zähl-Algorithmus auf einer Zwischenfrequenz, welche den Null-Pegel definiert, benötigt. Der DDC-Core implementiert einen Tiefpass mit einer festen Grenzfrequenz von 200 kHz. Um das FM-Senderspektrum maximal effektiv nutzen zu können, wird dieses in die Mitte (100 kHz) des Tiefpassfilters gelegt. So wird möglichst wenig des Signalspektrums an den Kanten verloren gehen. Dies ist genau die Mitte des Tiefpasses. Wenn der DDC nun für einen FM-Rundfunkkanal, zum Beispiel NDR90.3 ($f_{mix} = (90,3 - 80 - 0,1)MHz = 10,2MHz$), konfiguriert ist, kann eine Implementierung des Algorithmus mit dem Testsystem des vorigen Abschnitts getestet werden.

Der demodulierte Datenstrom hat auch nach der Demodulation noch seine Abtastrate von 1,25 MHz. Für das enthaltende Mono-Audiosignal (bis 15 kHz) bedeutet dies eine starke und unnötige Überabtastung. Dies sollte für eine Weiterverarbeitung geändert werden, um den Rechenaufwand weiter zu minimieren.
Das Mono-Audiosignal eines FM-Rundfunkkanals hat eine Bandbreite von 15 kHz. Die minimale Dezimierung des Datenstroms ist somit nach dem Shannon-Nyquist-Abtasttheorem auf 30 kHz möglich. Bei einer Dezimierung des Datenstroms wird das digitale Audiospektrum zusammengepresst. Damit hierbei kein Aliasing auftritt, ist eine ideale Tiefpassfilterung bei 15 kHz nötig. Eine Dezimierung um einen rationalen Faktor ist durch den Zwang, vorher interpolieren zu müssen, zeitaufwendiger, als eine Dezimierung um eine ganze Zahl. Außerdem muss die resultierende Samplefrequenz ebenfalls eine ganze Zahl sein, da Codecs beziehungsweise Multimediabibliotheken (wie DirectX) eine Zahl vom Datentyp Integer voraussetzten. Eine ganze Zahl, die für diesen Datenstrom von 1,25 MHz geeignet ist, ist zum Beispiel 40. Dieser Faktor dezimiert den Datenstrom auf $f_{dez} = \frac{1,25\ MHz}{40} = 31,25\ kHz$ ohne eine rationale Frequenz zu erzeugen. f_{dez} ist nun die Abtastfrequenz des Ausgangssignal und muss dem Shannon-Nyquist-Abtasttheorem genügen. Mit $f_{max,Audio} = 15\ kHz$, $f_{dez} = 31,25 kHz$ ist das Abtasttheorem ($\frac{f_{dez}}{2} > f_{max,Audio} => \frac{31,25\ kHz}{2} > 15\ kHz$) erfüllt.

Zur Verifikation wurde der CMEX-SDR-Client mit der bestehende USB-Lösung und dem analogen Frontend eingesetzt. Hiermit können reale Rundfunksignale empfangen und mit dem Algorithmus getestet werden. Ein MATLAB-Skript, welches diesen Test durchführt und die Resultate in Plots deutlich macht, ist im Anhang *12* zu finden. In Abbildung 4.10 ist das FM-Demodulationsergebnis des Senders NDR 90.3 deutlich im Vergleich zum erwarteten Spektrum zu sehen. Im Anhang *13* ist zudem eine WAV-Datei mit dem FM-demodulierten Audiosignals zu finden.

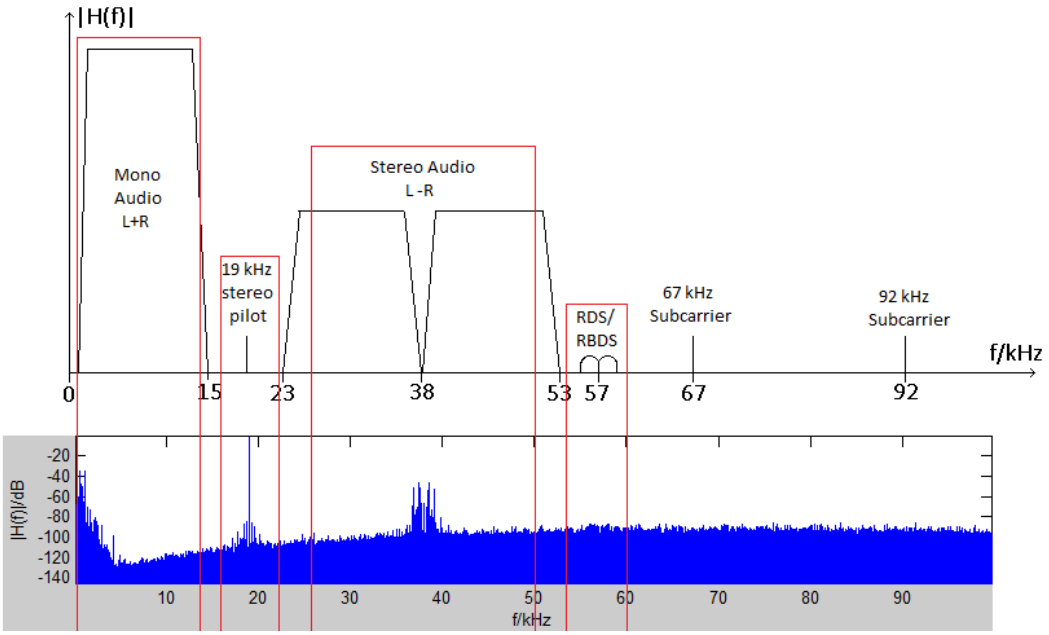

Abbildung 4.10: Demodulierter Rundfunksender NDR 90.3

4.4 Embedded System im FPGA

Die Firmware des SDR setzt auf dem CPU-basierten Embedded-System des FPGA auf. Im Kapitel 3.4 wurde beschrieben, wie dieses System auszusehen hat. Der Abschnitt der Realisierung befasste sich mit der Hardware des Embedded Systems. Aufsetzende Software wie DDC-Treiber und DDC-Server sind nicht Teil dieses Abschnitts.

Für die Analyse im Kapitel 3.3.3 wurde bereits ein PPC-basiertes System mit der Xilinx ISE Design Suite erstellt, welches bis auf dem DDC-Core und einem General-Purpose-Input-Output (GPIO) Core alles Nötige für das zu realisierende System enthält. Für diese Realisierung wurde somit das bestehende System der Analyse um die noch fehlenden Cores erweitert, sodass sich der Ausschnitt der konzeptionierten System-Struktur aus Abbildung 3.10 ergibt.

4.4.1 System-Konfiguration

Detailinformationen zu dem realisierten Embedded-System sind dem EDK-Projekt dem Anhang *11* zu entnehmen. Die wichtigsten Eckpunkte sind jedoch im Folgenden erläutert.

- **Ethernet-Core:** Für das SDR-System ist eine schnelle Übertragung mit UDP vorgesehen, also wird GigE genutzt. Der Ethernet-Core (MAC) arbeitet mit dem Tri-Speed

Ethernet PHY *Marvell Alaska PHY (88E1111)* und unterstützt 10 Mb/s, 100 Mb/s genauso wie 1000 Mb/s (GigE) (das Datenblatt zum PHY ist in [16] zu finden). Zur Kommunikation zwischen PHY und MAC kommt das standardisierte Media-Independent-Interface[4] (MII) zum Einsatz. So ist sichergestellt, dass eine herstellerübergreifende Kommunikation zwischen fremden PHYs und MACs möglich ist. MII ist für Übertragungen bis 100 Mb/s gedacht. GigE wird von Gigabit-MII (GMII) unterstützt, wobei GMII MII integriert, sodass beide Varianten nutzbar sind. Zu MII und GMMI existieren auch die R-Versionen. R steht hier für *reduced*. PHY mit weniger Pins nutzen diese Standards, bei welchen die Anschlussleitungen minimiert wurden.

Für die zu nutzende Konfiguration der Schnittstelle, hier GMII, ist der Jumper *J56* des ML507-Boards entsprechend der Einstellungen zu konfigurieren (Informationen hierzu in [52]). Im EDK-Projekt sind die PHY-Anschlussleitungen entsprechend GMII angeschlossen worden. Mit dieser Konfiguration ist es nun hardwareseitig möglich, den PHY eine *auto-negotiation* durchführen zu lassen, die den angeschlossen Link-Speed erkennt.

Der MAC selber ist als Hard-Core im FPGA vorhanden und implemtiert eigentlich zwei MACs. Für dieses SDR wird ein zweiter nicht benötigt und somit deaktiviert. Für eine schnelle Übertragung ist UDP-Checksum-Offloading hilfreich, somit ist diese Option im MAC aktiviert. Des Weiteren wird der Protokoll-Overhead durch große Ethernet-Pakete (Jumbo-Frames) deutlich minimiert. Der Core unterstützt bereits Jumbo-Frames bis 9 KiB, sodass dies nicht extra aktivert werden muss. Je Tx- und Rx-Pfad existiert ein FIFO, in dem die jeweiligen Bytes zwischengespeichert werden, bis sie zum Beispiel von einer DMA-Transaktion über einen LocalLink-Kanal abgeholt werden. Ein DMA-Transfer ist bei diesem Core nur über LocalLink-Kanäle möglich, welche hierfür von MAC zu RAM-Controller verbunden worden sind. Mit der Option für die FIFO-Größe von 32 KiB je Pfad ist sichergestellt, dass drei komplette Ethernet-Pakete Platz im FIFO finden, bevor eine DMA-Transaktion nötig ist, beziehungsweise diese auf die Leitung gesendet werden. Eine Verbindung zum Interrupt-Controller des Systems stellt sicher, dass die CPU auf den MAC reagieren kann.

Firmwareseitig setzt die Xilinx Treiberbibliothek *lltemac_v3_00_a* auf das Register-File des MACs auf, welche durch die Firmware des nächsten Abschnitts genutzt wird, um den MAC zu steuern. Der DMA-Zugriff wird mit der Treiberbibliothek *lldma_v2_00_a* ermöglicht.

- **GPIO-Core:** GPIO wird durch die Firmware genutzt, um Status-Meldungen mit LEDs des ML507-Boards anzuzeigen. Es werden die Postion- LEDs der Taster des Boards

[4] Ist eine standardisierte Schnittstelle von Ethernet PHYs und MACs, sodass PHYs und MACs von verschiedenen Herstellern ohne Probleme miteinander kooperieren. MII ist speziell für 100 Mb/s spezifiziert. Weitere Standards für eine PHY-MAC-Kommunikation sind RMII, GMII, RGMII, SGMII, XGMII.

(siehe ML507-Datenblatt in [52]) für diese Zwecke genutzt. Hierfür wurde der Core mit fünf IO Leitungen ohne Interrupts konfiguriert. Zur Nutzung des Cores kann in Software die Treiberbibliothek *gpio_v3_00_a* herangezogen werden.

- **UART-Core:** Mit dem UART wird eine Konsole für den Entwickler geschaffen, auf welcher Debug-Informationen erscheinen. Der Core ist nur statisch konfigurierbar, das heißt, die Übertragungseinstellungen sind nur vor der Synthese des FPGA Bitstreams möglich. Es wurde eine Baudrate von 115200 Bd mit 8N1[5] festgelegt. *uartlite_v2_00_a* ist die Treiberbibliothek zum Core.

- **MPMC-Core:** Als Controller für das DDR2-RAM des ML507 FPGA-Boards dient der MPMC. Da der DDC-Core eine NPI-Schnittstelle benötigt, ist eine Alternative nicht möglich. MPMC verwaltet alle Anfragen auf das RAM nach dem Round-Robin Prinzip[6]. Anfragen sind vom PLB-Bus, dem PPC440MC-Interface und NPI möglich. Diese sind hierfür entsprechend angeschlossen worden. Eine Verbindung zum DDR2 PHY ist wie auch beim Ethernet-PHY über eine konfigurierbare Schnittstelle, hier Xilinx-Memory-Interface-Generator[7] (MIG), möglich.

- **DDC-Core:** Zum Herabmischen des Funksignals wird der DDC-Core benötigt. Bis auf die korrekte Verdrahtung zum ADC und der NPI-Schnittstelle des RAM-Controllers (siehe UCF-Datei im EDK-Projekt) ist eine weitere Konfiguration des Cores nicht nötigt. Eine Tiefpassfrequenz von 200 kHz und eine Dezimierung von 64 sind fest implementiert. Für diesen Core existiert keine spezifische Bibliothek. Diese wird in den nächsten Realisierungsschritten entwickelt werden.

- **PowerPC440-Core:** Der zentrale Teil des Systems ist die PowerPC440 (PPC) CPU. Diese ist über die Speicher-Schnittstelle PPC440MC mit dem MPMC verbunden und teilt sich so mit den anderen RAM-Nutzern den RAM-Zugriff. Im PPC sind zwei 32 KiB große Caches, je einen für Instruktionen und einen für Daten, mit 32 Byte großen Cachelines implementiert.

  Die PPC CPU als Hard-Core im FPGA wurde von Xilinx durch den Crossbar erweitert [46]. Dieser ist nötig, da der PPC nur drei PLB-Slaves (Einen zum Lesen von Befehlen, einen zum Lesen von Daten und der letzte zum Schreiben von Daten) implementiert hat. Eine Nutzung von DMA-Kanälen über LocalLink oder den MPMC wäre ohne den Crossbar nicht möglich. Der Crossbar verbindet die drei nativen PLB-Schnittstellen des PPC mit zwei weiteren PLB-Slave Endpunkten, einem PLB-Master, dem MPMC über

[5] Ist eine Kurzschreibweise für die Einstellungen einer UART-basierten seriellen Übertragung. 8N1 = abc; a: Anzahl der Daten-Bits, b: **N**o parity/**P**arity, c: Anzahl an Stop-Bits

[6] Bezeichnet ein Zugriffs-Prinzip auf das RAM, nachdem die verschiedenen Nutzer des RAMs nacheinander einen Zeitschlitz zugewiesen bekommen

[7] Weiterführende Informationen hierzu sind der Xilinx Homepage zu entnehmen. http://www.xilinx.com

PPC440MC und mit vier LocalLink-Kanälen für DMA-Transaktionen. Der Crossbar koordiniert hierbei das *Verbindungskreuz* der Schnittstellen. Einen Überblick über den im Virtex-5 eingebetteten PowerPC mit Crossbar ist der Abbildung 4.11 zu entnehmen.

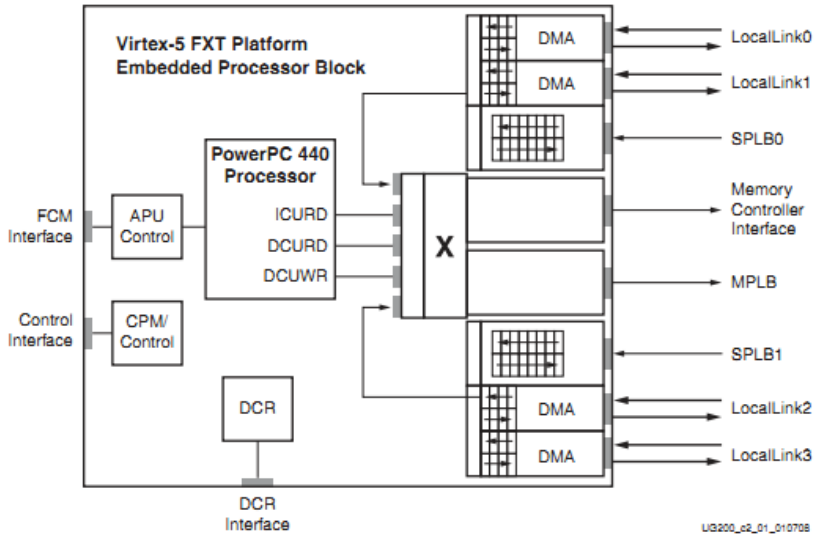

Abbildung 4.11: Überblick des im Virtex-5 eingebetteten PowerPCs. Quelle: Xilinx User Guide UG200 v1.8 vom 24.02.2010 - Figure 2-1

Alternativ zu der Nutzung der DMA-Kanäle am Crossbar, könnten auch die LocalLink-DMA-Kanäle direkt am MPMC-Speicher-Controller genutzt werden. Ein parallel erstelltes Embedded System mit dieser Alternative und einem Ethernet-MAC als zweitem DMA-Endpunkt hat gezeigt, dass diese Alternative geringfügig langsamer arbeitet wie die Lösung mit den DMA-Kanälen am Crossbar. Zudem würde durch die Nutzung der Crossbar DMA-Kanäle, Ports am MPMC frei für andere Verbindungen bleiben. Aus diesen Gründen wurde sich für dieses SDR für den Crossbar entschieden.

- **Sonstige Cores:** Zu *Sonstige Cores* sind diejenigen gezählt worden, welche über das Basis-System generiert worden sind oder unmittelbar aus den obigen Core-Konfiguration hervorgehen. Zum Beispiel die erzeugten Taktsignale des *clock_generators* ergeben sich unmittelbar aus den oben genannten Spezifikationen und der Konfiguration. Weitere Informationen zu den Core-Konfiguration sind den EDK-Projekten aus Anhang *11* entnehmbar.

4.4.2 Erreichbarkeitstest

Ein erster Test des realisierten Embedded-Systems ergibt sich aus den Synthese-Tools und deren Ergebnissen. Die Synthese war ohne Fehler durchführbar, sodass ein Test der Co-

res auf Software-Basis nun möglich ist. Die obig genannten Xilinx Bibliotheken bringen teils Testfunktionen der Cores mit sich. Durch diese ist geprüft worden, ob die jeweiligen Register-Files der Cores über die definierten Basisadressen erreichbar sind. Dies war der Fall, sodass das System nun bereit für eine Firmware bzw. die Entwicklung des DDC-Treibers ist.

4.5 DDC-Treiber

Als erste Software-Schicht über der digitalen Hardware des Embedded-Systems liegt der Handware-Abstraction-Layer (HAL), welcher den Register-Zugriff unter Verwendung von C-Makros abstrahiert. Auf den HAL setzt der eigentliche Treiber für die Verwaltung des Gerätes auf. Jedoch werden HAL und Treiber in dieser überschaubaren Realisierung zusammen unter dem Begriff DDC-Treiber betrachtet und nicht getrennt als HAL und Treiber.

Leider wurde in [29] kein HAL und kein Treiber für den DDC-Core entwickelt. Diese mussten auf Basis des bestehenden Cores entwickelt und getestet werden.

Der erste Abschnitt dieses Kapitels, soll klären wie die Register-Schnittstelle des DDC-Cores aussieht und wie damit zu arbeiten ist. Der zweite Abschnitt zeigt auf, wie der hier entwickelte Treiber auf das Register-File aufgesetzt ist. Ein abschließender Test des Treibers in Verbindung mit dem Core wird den Treiber für die nächsten Schritte der SDR-Realisierung verifizieren.

4.5.1 Core-Funktion

Das Register-File des DDC-Cores beinhaltet drei 32 Bit Register. Im Folgenden werden diese von der Basisadresse an mit REG0 bis REG2 beschrieben. Tabellen 4.2, 4.3 und 4.4 geben an, wie den Bits der Register Namen und Nummern zugeordnet werden. Alle Bits haben einen Resetwert von *0*. Entgegen der Bit-Nummerierung von Xilinx (in älteren Datenblättern) entspricht das 0te Bit dem Least-Significant-Bit (LSB).

REG0

Nummer	16-31	13-15	2-12	1	0
Name	CHUNK_COUNT	DECIMATION	Res.	SYNC	START

Tabelle 4.2: Beschreibung des DDC-Core Registers REG0

CHUNK_COUNT: Definiert mit einem 16 Bit vorzeichenlosen Integer die Anzahl an Chunks, die der Speicherbereich des DDC-Cores umfassen soll. Ein Chunk entspricht hierbei der Größe eines FAT-Sektors von 512 Byte. *Der Zusammenhang von FAT und DDC-Core liegt in der Tatsache, dass durch [29] der Core für die Nutzung mit einer USB-Massenspeicher basierten FAT-Datei-Lösung erstellt wurde. Spätere Core Versionen könnten dieses durch ein extra Register mit einem byteorientierten Wert und einer konfigurierbaren Interruptrate ändern.*
DECIMATION: Nimmt einen vorzeichenlosen 3-Bit-Integer auf, der den Dezimationsfaktor konfiguriert. $2^{DECIMATION}$ ist der Dezimationsfaktor. Der Core ist nur mit einem Dezimationsfaktor von 64 nutzbar, da die integrierten Filter hierfür von [29] fest dimensioniert wurden.
Res.: Ist mit keiner Funktion belegt.
SYNC: 1 gibt an, dass die im Core generierte Mischerfrequenz stabil mit der konfigurierten Frequenz (DDSFREQ) läuft. Durch *0* wird eine nicht stabile Frequenz signalisiert.
START: Mit einer *1* wird der DDC-Core gestartet. Dieser startet dann den DDS und fängt an, Daten vom ADC über NPI in den definierten RAM-Bereich (RAMADDR bis 512xCHUNK_COUNT) zu schreiben. Mit *0* kann der Core wieder gestoppt werden.

REG1

Nummer	0 - 31
Name	RAMADDR

Tabelle 4.3: Beschreibung des DDC-Core Registers REG1

RAMADDR: Gibt mit einem 32 Bit vorzeichenlosen Integer das Byte im Speicher an, ab dem der DDC-Speicherbereich anfangen soll. Ein Alignment des Wertes auf Chunkgröße ist zu empfehlen.

REG2

Nummer	0 - 31
Name	DDSFREQ

Tabelle 4.4: Beschreibung des DDC-Core Registers REG2

DDSFREQ: Erwartet einen vorzeichenlosen 32-Bit-Integer, der der gewünschten DDS-Frequenz in Hertz entspricht.

Der DDC-Core hat eine Interrupt-Leitung, die vor der Synthese durch [29] mit einem festen Interrupt-Intervall definiert wurde. Nach jedem 64ten per NPI geschriebenen Chunk wird ein Interrupt erzeugt (64 Chunks entsprechen $512 \cdot 64 = 32\ KiB => \frac{32 \cdot 1024\ Byte}{2 \cdot 2\ Byte} = 8192\ Samples$), das bedeutet ein Interrupt signalisiert dem Treiber immer 8192 neue IQ-Datenpaare. Bei einem angelegten Abtasttakt von 80 MHz und einer Dezimierung von 64, kommt bei gestartetem Core alle $t_{Int} = \frac{32\ Byte \cdot 1024 \cdot 64}{4\ Byte \cdot 80\ MHz} \approx 6,55\ ms$ ein Interrupt. Dieser Interrupt erlaubt einem nutzenden Software-System zu verfolgen, welche Bereiche des DDC-Speichers beschrieben wurden. Da der DDC wie ein Ringspeicher fungiert und bei Erreichen der Speicherobergrenze unten weiterschreibt, muss die Software auch dieses berücksichtigen.

Eine Interrupt-Kontrolle des DDC-Core Interrupts ist mit den Registern DGIER, IPIER und IPISR [43] möglich. Diese Register wurden automatisch bei der Integration des DDC-Cores in einen PLB-kompatiblen Core generiert [29].

4.5.2 Core-Treiber

Der Treiber verschiebt die Core-Funktion in eine abstraktere Sicht und ermöglicht den einfachen Zugriff durch die Firmware auf den Core. Der realisierte DDC-Treiber, wie er hier in der Programmiersprache C geschrieben wurde, ist dem Anhang *15* (ddc.h und ddc.c) zu entnehmen.

Als ersten Schritt wurde eine C-Struktur (*DdcRegisterFile*) definiert, die über das Core-Register-File gelegt werden kann, um einfache Zugriffe auf die Speicherzellen zu erhalten. Die Core-Basisadresse ist hierfür in den Typ *DdcRegisterFile ** zu casten. Ein weiterer Zugriff auf die Register ist durch die Makros *DDC_REG*(base)* möglich. Diese können in dem C-Code als Variablen genutzt werden, da hinter dem Makro ein dereferenzierter Pointer auf die Register-Adresse steckt. Ein HAL aus C-Makros (fangen mit der Bezeichnung *ddc_ll_* an) kann mit Makro-Konstanten (*DDC_*_OFFSET und DDC_*_MASK*) und dem oben geschilderten genutzt werden, um die DDC-Register abstrakter zu schreiben oder zu lesen.

Auf den HAL setzt sich der eigentliche Treiber auf. Je DDC-Core kann eine Treiberinstanz erstellt werden, die alle instanzbezüglichen Daten enthält. Funktionen (fangen mit *ddc_** an) erlauben das Arbeiten mit dem Core in Verbindung mit der übergebenen Instanz. Die Treiber-Instanz ist eine C-Struktur (*DdcDriver*) die die Basisadresse des Cores, einen Pointer auf das Register-File, DDC-RAM-Adresse, DDC-RAM Größe, den individuellen Interrupt-Identifikator des Cores, einen Pointer auf den zu verwendenden Interrupthandler, Zähler für die DDC-RAM-Leseposition und Statusinformationen über den Treiber enthält.

Der in der Treiberinstanz hinterlegte Interrupthandler ist im Treibermodul (*ddc_handler*) selbst implementiert und sorgt bei gestartetem DDC dafür, dass der Blockzähler in der je-

weiligen Treiberinstanz die geschriebenen Blöcke zählt. Ein Block entspricht hier 8192 Bytes, dieser Wert ist jedoch für eigene Bedürfnisse anpassbar. Grund für diese Dimensionierung ist die Tatsache, dass die Daten per UDP/IP over Ethernet an den PC gesendet werden sollen. 8192 Byte ist vor 9000 Byte (maximale Ethernet-Paketgröße des Ethernet-MAC), die letzte vom Format 2^x. Da mit 2^x-Zahlen im Programm am einfachsten gearbeitet werden kann und eine möglichst große Ethernet-Übertragungsrate erreicht werden soll, wurde 8192 Byte für ein Datenblock im Treiber festgelegt.

Dem Treiber ist ein Fehlernachrichtsystem implementiert worden, mit dem der Rückgabewert einer Treiberfunktion in eine ASCII-basierte Fehlermeldung umgewandelt werden kann. So kann ein Fehler leichter identifiziert und direkt über die Debug-Konsole angezeigt werden.

4.5.3 Core-Treiber Verifikation

Ein Test des Treibers ist essenziell für das weitere Vorgehen. Die folgende Aufzählung zeigt das Vorgehen einer Test-Firmware die alle DDC-Funktionen in einer sinnvollen Reihenfolge aufruft. Während des gesamten Tests werden die jeweiligen Registerzustände und Treiberinstanz-Eigenschaften mit einem JTAG-basierten Debugger schrittweise geprüft.

- CPU-System initialisieren
- Treiberinstanz erstellen
- Treiberinstanz mit *ddc_init* initialisieren
- Interrupthandler des Treibers dem Interrupt-Controller anhängen und Pointer auf die Treiber-Instanz als Interrupthandler-Parameter mit übergeben.
- Initiales Rücksetzen der DDC-Hardware mit *ddc_stop*
- Interrupt-Controller aktivieren
- DDC mit Hilfe der Funktion *ddc_start* starten
- Stoppen und Neustarten des DDC mit *ddc_restart* und *ddc_stop*

Der Test des Treibers hat keine Fehler aufgezeigt. Somit ist eine Realisierung des DDC-Servers, welcher diesen Treiber nutzt, um über UDP den DDC zu bedienen, im nächsten Schritt möglich. Der DDC-Server kann als zweiter Test für den Treiber angesehen werden, da dieser echte Daten in Echtzeit an den PC übermittelt und so große Datenmengen geprüft werden können.

4.6 DDC-Server

Als zentraler Punkt in der Firmware des Embedded-Systems zählt der DDC-Server. Er nimmt Befehle von der Außenwelt entgegen, führt diese aus, sorgt für das Auslesen des DDC-RAMs und schickt die Daten über UDP in Echtzeit an die registrierten SDR-Clients. Der C-Quellcode zu diesem Stück Firmware ist im Anhang *15* in den Dateien *ddc_server.h* und *ddc_server.c* zu finden. Dieser Teil der Realisierung setzt voraus, dass die lwIP Bibliothek (mit angewendetem Patch für das UDP-TX-Offloading) eingebunden ist (siehe *.mss-Datei im EDK-Projekt).

4.6.1 Server-Funktion

Zur Steuerung des SDR über UDP müssen bestimmte UDP-Pakete an einen UDP-Endpunkt versendet werden, standardmäßig ist dies *192.168.2.10:13000*. Diese Einstellungen sind vor der Kompilierung der Firmware konfigurierbar. Für spätere SDR-Versionen könnte über das UART eine Shell implementiert werden, sodass auch eine Konfiguration ohne eine Ethernet-Verbindung möglich ist.

Der DDC-Server registriert die IP des ersten Befehls als *ControlIp*, sodass in Zukunft nur Befehle von dieser IP ausgewertet werden (andere werden verworfen). Ein Rücksetzen dieser IP erfolgt nach dem Neustarten des SDR oder durch einen bestimmten Befehl (siehe unten). Eine Bedienung durch mehrere IPs wäre möglich, jedoch würden sich die Befehle überschneiden und kein sinnvolles Ergebnis erzeugen. Beispielsweise würde ein SDR-Client das SDR in Betrieb setzten und ein anderer zur gleichen Zeit dieses Rückgängig machen. Ist die *ControlIp* einmal hinterlegt, ist eine Kontrolle des SDR nur mit dieser IP möglich. Außerdem trägt der DDC-Server die *ControlIp* automatisch als UDP-Endpunkt ein, also demjenigen an welchem DDC-Daten versandt werden (SDR-Clients). Die ausgehenden UDP-Pakete haben standardmäßig eine Größe von 8192 Byte, dieses ist über C-Makros konfigurierbar.

UDP-Befehle unterliegen nur zwei Konvention. Der Befehl muss komplett in ein UDP-Paket passen und die ersten 32 Bit des Pakets müssen einem definierten Marker entsprechen, der das Format der folgenden Bytes definiert. Es wurden bis jetzt fünf Marker in der Firmware hinterlegt. Im Folgenden sind diese Marker kurz erläutert. In den eckigen Klammern der Aufzählungstitel ist das Befehlsformat gezeigt. Zwischen den Kommata stehen die verschiedenen zu übergebenen Parameter plus das zu nutzende Zahlenformat. Der erste Wert gibt den Marker an.

- **DDC_SERVER_MARKER_START *[0x000000FF (uint32), MIXER_FREQUENCY (uint32), DECIMATION (uint32), WITH_ID (uint32), DATA_PORT (uint32)]*:** Dieser Befehl startet den DDC mit den gegebenen Parametern. Nach dem 32 Bit Marker

erwartet dieser Befehl vier weitere 32 Bit Werte. Sie definieren als vorzeichenlose Integer die Mischer-Frequenz, Dezimation, das Senden einer Paket-ID und den UDP-Port für die DDC-Daten.

Um eine Prüfung der UDP-Übertragung zu erlauben, kann eine 16 Bit ID jedem ausgehenden SDR UDP-Paket hinzugefügt werden, welche mit jedem Paket in der Sendereihenfolge inkrementiert wird. Hiermit ist es möglich, verlorengegangene Pakete zu erkennen oder zu merken, ob die Paketreihenfolge vertauscht wurde. Ist WITH_ID ein Wert größer *0*, wird die ID hinzugefügt und bei *0* nicht.

DATA_PORT definiert den UDP-Port an dem, bei einem gestarteten SDR, die ausgehenden UDP-Pakete versendet werden. Bei einem Port größer gleich 2^{16} oder gleich 0 wird ein Fehler in der Konsole erzeugt.

- **DDC_SERVER_MARKER_STOP *[0x00000FFF (uint32)]*:** Diesem Marker folgenden keine weiteren Parameter. Der DDC wird gestoppt und keine UDP-Pakete werden an die registrierten SDR-Clients gesendet.
- **DDC_SERVER_MARKER_RESTART *[0x000000FF (uint32), MIXER_FREQUENCY (uint32), DECIMATION (uint32), WITH_ID (uint32), DATA_PORT (uint32)]*:** Dieser Befehl nimmt die gleichen Parameter wie DDC_SERVER_MARKER_START entgegen und erlaubt es, das SDR während der Nutzung zu rekonfigurieren. Diese Funktion kann zum Beispiel für ein schnelles Tuning durch die Frequenzbänder genutzt werden.
- **DDC_SERVER_MARKER_CLEAR_CONTROL_IP *[0x0FFFFFFF (uint32)]*:** Es werden keine weiteren Parameter nach dem Marker erwartet. Dieser Befehl löscht alle SDR-Client UDP-Endpunkte, stoppt das SDR und setzt die *Controllp* zurück. Nach diesem Befehl ist das SDR nahezu im Urzustand und ein nächster Befehl würde dem DDC-Server eine neue *Controllp* liefern.
- **DDC_SERVER_MARKER_PING *0xFFFFFFFF (uint32), RANDOM_VALUE (uint32)]*:** Eine Prüfung der Erreichbarkeit kann mit diesem Befehl erreicht werden. Wenn das SDR im Urzustand ist, also keine *Controllp* festgelegt ist, würde dieser Befehl das SDR dazu veranlassen, eine Antwort über den UDP-Datenkanal zu versenden. Standardmäßig ist das der UDP-Endpunkt *Controllp:13001*. Das Antwort-Paket ist das Selbe was empfangen wurde, jedoch mit dem um eins erhöhten RANDOM_VALUE. Erreicht diese Antwort den Befehlssender weiß dieser, dass das SDR erreichbar ist.

Alle Befehle (außer die letzten drei) sind für die Nutzung des SDR von einem PC aus nötig. Weitere Befehle sind optional und würden die Arbeit mit dem SDR weiter erleichtern. Eine Erweiterung um weitere Befehle ist ohne Weiteres möglich (siehe nächster Abschnitt).

4.6.2 DDC-Server

Wie auch beim DDC-Treiber wurde für den DDC-Server eine C-Struktur (*DdcServer*) angelegt, die alle nötigen Informationen zu dem aktuellen Zustand des Servers speichert. Hierzu gehören ein Pointer auf die zu verwendende DDC-Treiber-Instanz, DDC-Core Parameter für den nächsten Start des SDR und UDP-Verbindungs-Konfigurationen. Die Verbindungs-Konfigurationen umfassen die *Controllp*, die SDR-Client UDP-Endpunkte und den UDP-Server Endpunkt zum Empfangen von Befehlen.

Die Nutzung des DDC-Server-Moduls umfasst die Kenntnis der öffentlichen Funktion (fangen mit ddc_* an). Hierzu gehören die drei folgenden Funktionen.

- ***ddc_server_init*:** Initialisiert die Server-Struktur *DdcServer*, registriert sich beim lwIP-Stack, erstellt den DDC-Server UDP-Endpunkt und prüft ob der DDC-Treiber konfiguriert und einsatzbereit ist.
- ***ddc_server_rx_callback:*** Diese Funktion wurde bei der Initialisierung des DDC-Server-Moduls beim lwIP-Stack als Rx-Callback-Funktion registriert. Diese wird immer dann aufgerufen, wenn ein neues UDP-Paket den lwIP-Stack durchlaufen hat und als gültig gilt. Innerhalb des Callbacks wird geprüft, ob das Paket von einer gültigen *Controllp* kommt. Falls keine *Controllp* registriert ist, wird diese hinzugefügt und ein UDP-Endpunkt in der SDR-Client Liste erzeugt.

  Eine Dekodierung des Befehls geschieht anhand des 32 Bit Markers, welcher extrahiert und gespeichert wird. Anhand einer Switch-Case-Anweisung wird dann der jeweilig hinterlegte Code (Funktionen fangen mit _* an) für den Marker ausgeführt. Der jeweilige Code dekodiert die Parameter des jeweiligen Pakets und führt den Befehl mit diesen Parametern aus.

  Wenn ein neuer Befehl hinzugefügt werden soll, ist dies einfach durch eine Erweiterung der Switch-Case-Anweisung und Erzeugen des Befehlscodes möglich. Als Skeleton[8] kann einfach ein bestehender Befehlszweig kopiert werden.
- ***ddc_server_poll:*** Damit das DDC-Server-Modul am Leben erhalten wird, also eine Reaktion auf Ereignisse möglich ist, gibt es diese Funktion. Sie soll zum Polling des DDC-Servers genutzt werden. Sollte eine Thread-basierte Firmware eingesetzt werden, sollte diese Funktion in einer Endlosschleife im Thread laufen.

  Während eines *poll*s werden in Abhängigkeit des DDC-Server-Zustandes und des DDC-Treibers die Daten aus dem DDC-RAM geladen und als UDP-Pakete verpackt verschickt. Sollte der DDC-RAM zu langsam ausgelesen werden (keine Echtzeit) wird dies als *dropped block* im DDC-Treiber vermerkt.

[8] Ist eine Vorlage für ein Quellcode-Stück.

4.6.3 DDC-Server Verifikation

Die Analyse aus Kapitel 3.6 hat gezeigt, dass eine Echtzeit-Verarbeitung mit MATLAB und einer Abtastrate von 1,25 MHz nicht möglich ist. Somit fällt ein Echtzeit-Test mit MATLAB aus. Allerdings wurde mit der dsp-Toolbox von MATLAB und den darin enthaltenden UDP-Sender geprüft, ob die Befehle vom Server angenommen und umgesetzt wurden. Der JTAG-Debugger in Verbindung mit Wireshark[9] hat gezeigt, dass die implementierten Befehle angenommen und umgesetzt werden.

Dass ein Echtzeit-UDP-Datenstrom mit einem Basissystem möglich ist, hat das Analyse-Kapitel 3.3 gezeigt. Dass der DDC-Server mit der darunterliegenden Hardware funktioniert, wurde hier gezeigt. Ob eine Echtzeitfähigkeit mit dem DDC-Server erreicht werden kann, kann erst nach der Realisierung des SDR-Client im Kapitel 5 gezeigt werden. Denn dieser Test ist ohne den mit C# programmierten SDR-Client nicht möglich.

4.7 SDR-Client

Eine Kommunikation vom PC zum SDR ist über dem im PC-Betriebssystem enthaltenden TCP/IP-Stack unter Kenntnis der UDP-Steuerbefehle zum SDR möglich. Mit dem SDR-Client soll dieses weiter abstrahiert werden, sodass der SDR-Nutzer kaum Kenntnisse über die UDP-Kommunikation haben muss. Der SDR-Client wird ein MAT-File-Writer enthalten, der erlaubt, die empfangenen Daten des SDR in eine MAT-Datei zu schreiben. Weiter wird ein live FM-Radio implementiert, um in Echtzeit Rundfunksender zu hören. Dieser Abschnitt beschreibt, wie der SDR-Client auf Basis von C# und dem *Microsoft Visual Studio 2010 Express* umgesetzt wurde.

4.7.1 SDR-Klasse

Als Basis für den MAT-File-Writer und dem FM-Radio dient die in dieser Arbeit entwickelte SDR-Klasse. Sie enthält statische Methoden, die die einzelnen UDP-Befehle repräsentieren. Durch den Aufruf einer solchen Funktion wird ein Internet-Socket zum gegebenen UDP-Endpunkt (Remote IP und Control Port des SDR) erstellt und der Befehl an das SDR gesendet. Nach dem Methodenaufruf ist der Socket wieder geschlossen, sodass diese Funktionen auch statisch nutzbar sind. Zum Beispiel ist mit den statischen Methoden *Sdr.Start(...)* und *Sdr.Stop(...)* ein Starten und Stoppen des SDRs über UDP möglich.

[9] Ist ein PC-Programm zum Verfolgen der Daten, die über einen Ethernetadapter gehen.

Sollen mit Hilfe der SDR-Klasse Daten in Echtzeit aus dem SDR gelesen werden, sind die Klassen-Eigenschaften (Dezimation, Mischerfrequenz, Ports, usw.) entsprechend zu setzten. Durch den Aufruf der Methode *Connect* würde ein UDP-Daten-Socket erstellt und das SDR mit den Klassen-Eigenschaften synchronisiert werden. Über die blockende Methode *ReceiveData* können nun die Daten aus dem Socket gelesen werden. Die Daten werden von der Remote IP aus an den PC (Local IP und Local Port) versendet, der vom SDR als Client eingetragen wurde. Während einer Verbindung können die Klassen-Eigenschaften immer geändert werden, eine Synchronisation mit dem SDR erfolgt allerdings erst durch einen expliziten Aufruf der Methode *Sync*. Dieser startet das SDR mit den neuen Klassen-Eigenschaften neu und lässt den Daten-Socket bestehen. Ein Tuning durch das Frequenzband ist somit leicht möglich.

ReceiveData kann die Daten wahlweise als Array vom Typ *Complex* oder als Liste mit zwei Arrays vom Typ *Int16* (ein *Int16*-Array jeweils für die I- und Q-Signalanteile) liefern. Sind keine Daten am Socket verfügbar, würde die Funktion nach Ablauf des definierten Timeouts eine Exception auslösen.

4.7.2 MAT-File-Writer

Damit SDR-Daten in eine MAT-Datei gespeichert werden können und MATLAB so Zugriff auf diese hat, wurde die SDR-Klasse um die statische Methode *ToMatFile* erweitert. Mit dieser Methode und einem Objekt der SDR-Klasse können über ein zeitbegrenztes Intervall Daten in die angegebene MAT-Datei gespeichert werden. Es kann hierbei ausgewählt werden, ob Paket IDs vom SDR mit den Paketen gesendet werden sollen, die MAT-Datei komprimiert werden soll und ob Status/Konfigurations-Informationen als Variablen mit in die Datei geschrieben werden. Die zusätzlichen Variablen können einem MATLAB-Skript helfen, die Daten besser zu interpretieren oder für den Nutzer darzustellen.

Eine MAT-Datei binär kompatibel zu MATLAB zuschreiben, muss in Anlehnung an die Spezifikation [17] von Mathworks[10] geschehen. Es gibt allerdings eine C#-basierte Bibliothek (*CSMatIO*[11]), die frei von der MATLAB-Homepage zugänglich und nutzbar ist. CSMatIO wurde in der Methode *ToMatFile* genutzt, um eine MAT-Datei zu erzeugen.

Eine Bedienung über die Konsole des Betriebssystems ist für die tägliche Nutzung sehr unhandlich. Mit der Software *HawSdrClient* aus dem Anhang *16SdrClient* wurde eine C#-basierte grafische Oberfläche (GUI) (Bildschirmfoto der GUI siehe in Abbildung 4.12) geschaffen, die mit Hilfe der SDR-Klasse und ihren statischen Methoden eine benutzerfreundliche Handhabung des SDRs ermöglicht.

[10] Hersteller der Software MATLAB/Simulink.
[11] http://www.mathworks.com/matlabcentral/fileexchange/16319

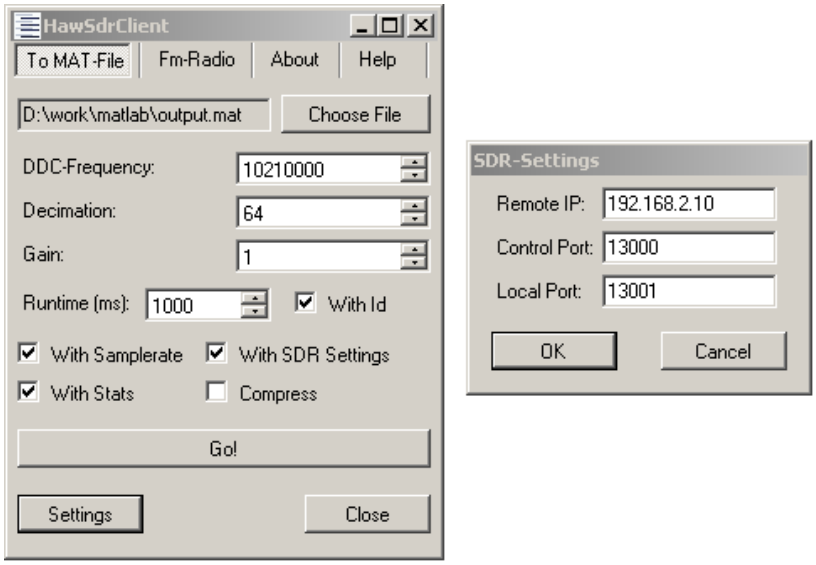

Abbildung 4.12: Bildschirmfoto des MAT-File-Writers im HawSdrClient

4.7.3 FM-Radio

Im Reiter *Fm-Radio* von *HawSdrClient* ist das live FM-Radio nutzbar (Abbildung 4.13). Um eine Echtzeit Demodulation zu erreichen, muss die Signalverarbeitung mit C# schneller sein als die Daten ankommen. Um dieses bei der Verarbeitung festzustellen, wird, wenn die Option *Stats* aktiviert ist, die Zeit gemessen, die sich der verarbeitende Thread im blockierten Zustand befindet. Ist diese Zeit nahe zu Null, sollte eine Überarbeitung der Signalverarbeitung durchgeführt werden.

Der verarbeitende Thread wartet bis ein UDP-Paket beim Socket angekommen ist und verarbeitet dieses direkt als Vektor von Daten. Für die Verarbeitung wird der Algorithmus aus Kapitel 4.3 eingesetzt. Er wurde jedoch um eine Dezimierung mit voriger Tiefpassfilterung und eine Filterung des dezimierten Signals in der Hauptschleife erweitert. Der erste Tiefpass ist auf 30 kHz bezüglich eines Dezimationsfaktors von 20 dimensioniert, um als Anti-Aliasfilter zu fungieren. Der zweite Tiefpass entfernt Audiosignalanteile über 15 kHz, damit die Audioqualität steigt und der kontinuierliche Pilotton bei 19 kHz nicht hörbar ist. Abbildung 4.14 zeigt die blockweise Verarbeitung des Signals im Programm.

Eine Ausgabe des generierten Audiostroms erfolgt mit der DirectSound API von DirectX[12]. Math .NET[13] diente mit Neodym[14] als Grundlage für die Signalverarbeitung. Neodym bietet

[12] Multimediabibliothek für Windows-System.

[13] Freie C# Bibliothek für mathematische Anwendungen. http://www.mathdotnet.com/

[14] Teil von Math .NET, welcher speziell für die digitale Signalverarbeitung gedacht ist. http://www.mathdotnet.com/Neodym.aspx

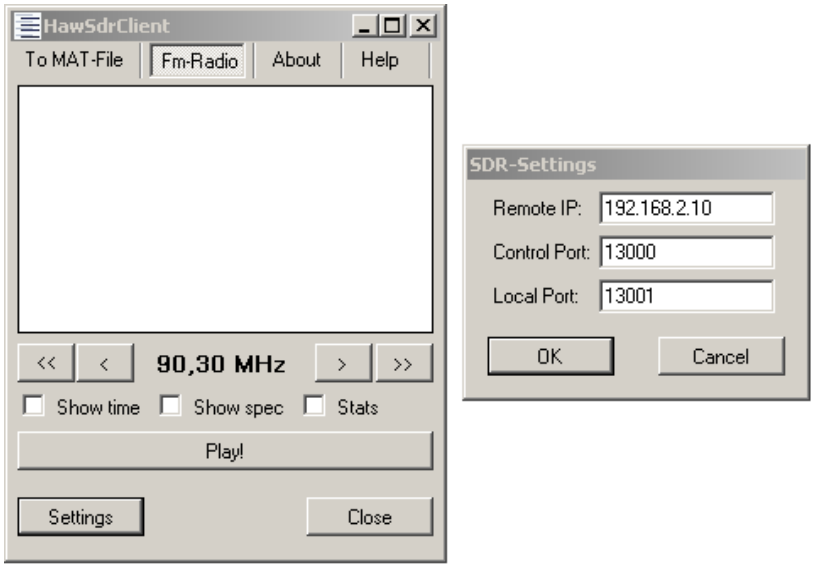

Abbildung 4.13: Bildschirmfoto des FM-Radios im HawSdrClient

FIR-Filter-Klassen, die während der Laufzeit dimensioniert werden und dann im Programm für die Verarbeitung nutzbar sind.

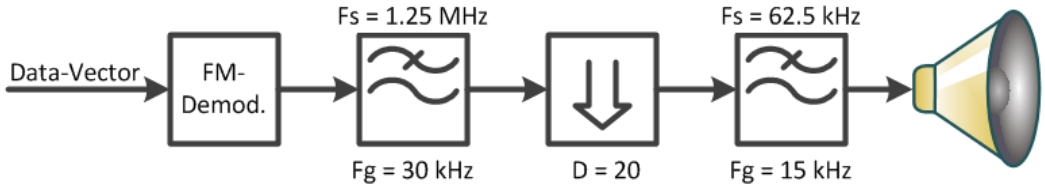

Abbildung 4.14: Übersicht der Signalverarbeitung im SDR-Client

5 Systemtest

Ein kompletter Test des Systems wird Teil dieses Kapitels sein. Er soll zeigen, dass die Realisierung in Anlehnung an das Konzept erfolgreich ist. Da die jeweiligen Teilsysteme bereits nach ihrem jeweiligen Abschluss getestet worden sind, beschränkt sich dieser Test auf *die Sicht als Ganzes*.

Nun besteht eine verifizierte Möglichkeit, das SDR vom PC aus zu bedienten. Ob die Datenübertragung auch in Echtzeit möglich ist, soll jetzt gezeigt werden. Damit im UDP/IP-Netzwerk keine weiteren Datenpakete den Datenverkehr zwischen SDR und PC stören oder eventuelle Bandbreite belegen, ist das SDR direkt mit einem zweiten Ethernetadapter im PC verbunden worden. Die Option für Jumbo-Frames im Adapter (*Intel Gigabit CT Desktop Adapter*) wurde für diesen Test aktiviert, da UDP-Pakete mit 8192 Byte ankommen werden.

Der SDR-Client bietet die Möglichkeit, Paket-IDs vom SDR in die UDP-Daten-Pakete hinzufügen zu lassen. Diese Option wurde im Reiter *To MAT-File* und in Verbindung mit den Optionen *With Stats*, *With Samplerate* und *With SDR Settings* genutzt, um ein sinusförmiges Generatorsignal (3,0001 MHz) am SDR-Eingang (ohne Frontend) mit dem SDR aufzunehmen und in eine MAT-Datei zu schreiben. Durch Einlesen der MAT-Datei in MATLAB sind alle Daten im Workspace für MATLAB zugänglich (Abbildung 5.1).

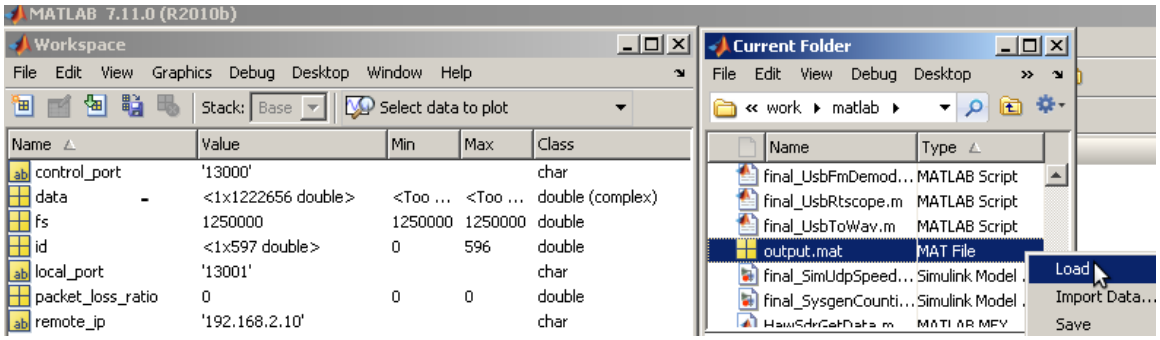

Abbildung 5.1: Bildschirmfoto, einlesen der MAT-Datei

Die MATLAB-Skripte aus dem Anhang *17* wurden benutzt, um die nun in MATLAB verfügbaren Daten grafisch darzustellen. Abbildung 5.2 zeigt ein Teil des sinusförmigen Signals des I-Signalanteils. Die Sprünge im Signal sind die mitgesendeten IDs (siehe Option-*With Id*).

In Abbildung 5.3 wurden die IDs aus allen Datenpaketen extrahiert und über den Index der Pakete dargestellt. Es ist eine lineare Funktion ohne Sprungstellen zu sehen. Des weiteren ist die Nummer des letzten Pakets und dessen Paket-ID gleich. Das heißt, es sind keine Pakete verloren gegangen oder in keiner falschen Reihenfolge angekommen. Ein Test über ein längeres Zeitintervall mit der Stats-Ausgabe des SDR-Clients zeigte das gleiche Verhalten (Abbildung 5.4).

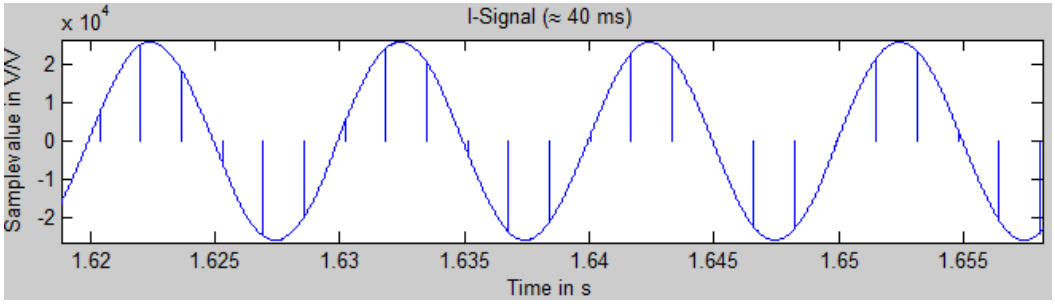

Abbildung 5.2: I-Signalanteil eines Sinus-Testsignals mit IDs

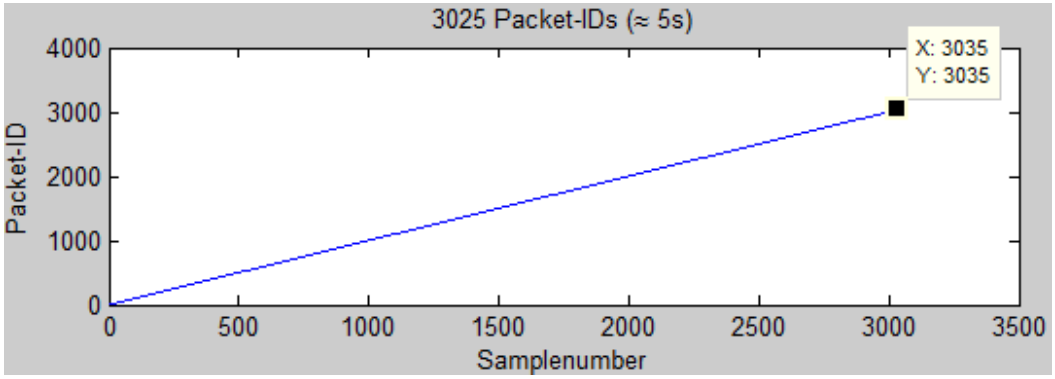

Abbildung 5.3: IDs der UDP-Datenpakete, aufgetragen über die Paketindizes

Abbildung 5.4: Stats-Ausgabe des SDR-Clients für eine lange Messung

Ein anschauliches Bild für die I und Q Signale des sinusformigen Eingangssignals ist durch Abbildung 5.5 gegeben. Es zeigt eine komplexe Ebene (I+jQ) über die Zeit (z-Achse), wobei I auf der x-Achse und Q auf der y-Achse aufgetragen ist. Zu erkennen ist deutlich der

rotierende komplexe Zeiger im 3D-Raum, der die Betrags- und Phaseninformationen trägt. Die Spitze des komplexen Zeigers definiert die sichtbaren Punkte in der Grafik. Das Resultat der Rotation des Zeigers in Abhängigkeit der Zeit, lässt die sichtbare Spirale (Blau) um die Zeitachse entstehen. Im Ursprung ($t = 0\ s$) ist diese Spirale nicht zu erkennen (siehe Punkt 2 in Abbildung 5.5). Grund hierfür ist der Einschwingvorgang der Filter im DDC-Core. Abbildung 5.6 zeigt dieses deutlicher, in einer 2D-Grafik des Signals I(t). Punkt 1 in der Abbildung 5.5 sind die Packet-IDs und sagen in diesem Bild nichts weiter aus.

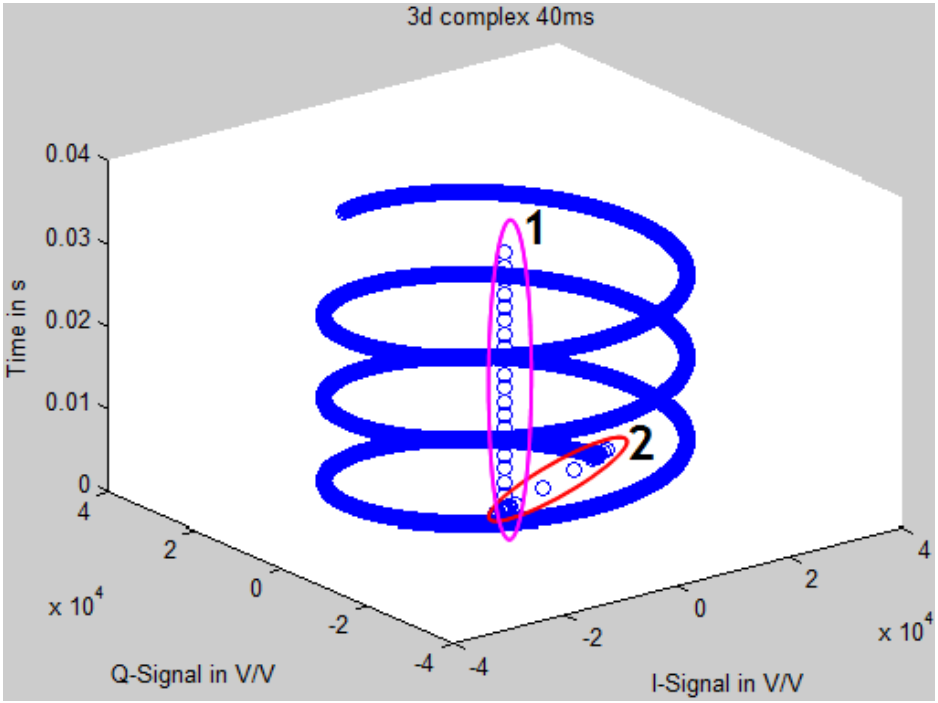

Abbildung 5.5: Komplexe I+jQ-Ebene aufgetragen über die Zeit

Das der Quellcode des FM-Radios, des zweiten Reiters im SDR-Client, funktioniert wurde bereits durch den Debugger im *Microsoft Visual Studio 2010 Express* festgestellt. Ob allerdings die Audioausgabe erfolgreich funktioniert, ist nur durch Hören während der Nutzung festzustellen. Es konnte direkt ein Rundfunksender eingestellt und gehört werden. Ein Sweep durch den VHF-2 Frequenzbereich, mit der Tuning-Funktion des FM-Radios, war ebenfalls zur vollsten Zufriedenheit möglich. Zudem ist das hörbare Signal mit einem Online-Internetradio des gleichen Rundfunksenders verglichen worden. Eine einwandfreie Nutzung als FM-Rundfunkradio in Echtzeit ist gegeben.

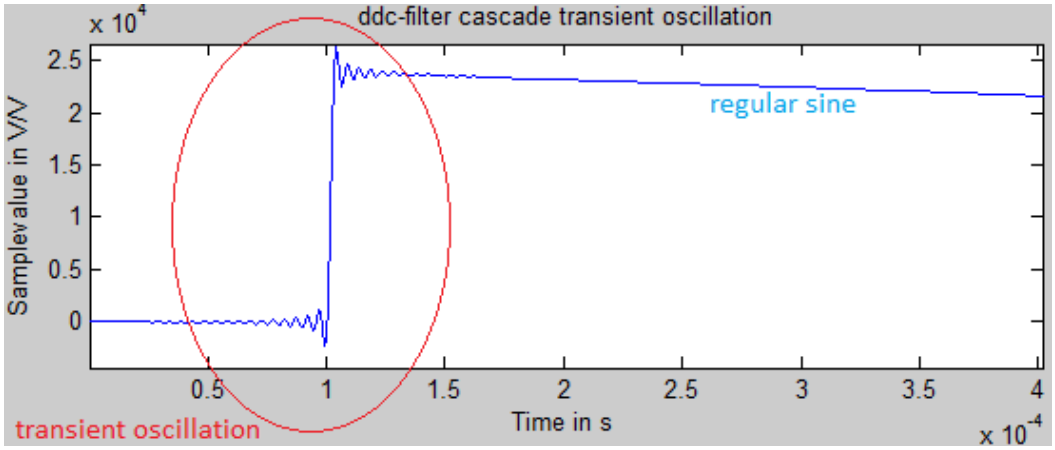

Abbildung 5.6: Einfluss des Einschwingvorgangs der DDC-Filter auf das komplexe Signal

6 Zusammenfassung und Ausblick

SDR-Lösungen gibt es bereits viele. Dennoch sollte durch eine SDR-Lösung der HAW-Hamburg eine eigene Plattform entstehen, die es Studierenden ermöglicht, einfach mit realen Funksignalen in einer gewohnten Umgebung (MATLAB) zu arbeiten.

Zusammenfassung

Im Rahmen dieser Arbeit wurde von mir ein SDR auf der Basis des bestehenden DDC-Cores und von *UDP/IP over Gigabit-Ethernet* geschaffen, das eine schnelle und plattformunabhängige Datenübertragung zum PC erlaubt. Durch die Port-basierte Struktur des UDP Protokolls können mehrere virtuelle Kanäle für Datenströme oder Steuerströme, auch nachträglich, einfach erstellt werden. Durch IP ist das System auch in einem LAN nutzbar und somit mobil.
Damit all dies möglich ist, wurde für den bestehenden DDC-Core [29] einen Treiber mit HAL entwickelt, der den Core abstrahiert. Desweiteren ist eine DDC-Server entwicklet worden, der im SDR-System und dem gegebenen lwIP TCP/IP-Stack für die Steuerbarkeit des SDRs über UDP sorgt.
Eine benutzerfreundliche Bedienung ist über das echtzeitfähige, in C# entwickelte, PC-Programm *HawSdrClient* möglich gemacht worden, welches zudem eine Schnittstelle für eine zeitbegrenzte Datenenge zu MATLAB bietet (MAT-Dateien). Im Kern des *HawSdrClient* steckt die speziell für dieses SDR realisierte SDR-Klasse, die das SDR abstrahiert und auch später noch einfach erweitert werden kann.
Zur Verifikation des SDR-Systems ist ein einfaches VHF-2 Frontend entworfen und realisiert worden, das dem ADC im System ermöglicht, Rundfunksignale mit dem SDR aliasingfrei über eine Bandpassunterabtastung umzusetzen. Ein FM-Radio im entwickelten *HawSdrClient* dient als Demonstrator, das ein vollwertiges Echtzeit FM-Rundfunkradio in Software darstellt.
Für die initiale Verifikation des DDC-Cores und der ersten Teilmodule dieses Systems, musste die bereits bestehende Testumgebung [29], bestehend aus Firmware und PC C++-Programm, umprogrammieren bzw. auf Basis einer MATLAB-CMEX-Datei neu entwickelt werden.

Das von mir realisierte SDR ist für die gegebenen Anforderungen der Arbeit ausgelegt und hat diese voll erfüllt. Ein Empfang von realen VHF-2 Signalen mit dem bestehenden DDC-Core [29] ist über eine echtzeitfähige, plattformunabhängige und erweiterbare Schnittstelle zum PC gegeben. Ein echtzeitfähiges FM-Rundfunkradio als Demonstrator zeigt zudem anschaulich die Funktion des Systems.

Ausblick

Über die Anforderungen dieser Arbeit hinaus könnte weiterhin an einigen Modifikationen und Erweiterungen des Systems gearbeitet werden. Diese würden das SDR-System noch flexibler machen und mit mehr Funktionen versehen. Somit würde dieses SDR auch für andere Labore der HAW-Hamburg oder anderen Hochschulen interessanter sein.

Im Folgenden wird aufgezeigt, welche Teile des SDR modifiziert oder erweitert werden könnten und wie weitere Abschlussarbeiten hieraus hervorgehen könnten, um ein volles SDR mit Sende- und Empfangszweig für einen universellen Gebrauch zu erhalten.

1. **Bedienung - Weitere Befehle:** Im Zuge einer Erweiterung des SDR werden immer mehr Befehle hinzukommen. Somit ist dieser Punkt zu jeder Erweiterung zu zählen. Dennoch können Befehle wie für eine Job-Verarbeitung oder zur Systemkonfiguration (zum Beispiel Puffergrößen oder Offset-Korrektur) eingepflegt werden.
2. **Bedienung - Uart-Shell:** Sollte eine Verbindung zum SDR über UDP und ein Transport ins Labor zum Programmer nicht möglich sein, muss eine Konfiguration des SDR dennoch erfolgen können. Hierfür könnte eine Shell mit Hilfe eines UARTs realisiert werden die es erlaubt, das ganze System zu steuern, zu konfigurieren und Fehlermeldungen, sowie Debug-Informationen, zu lesen. Eine Wartung des Systems ist so jederzeit möglich.
3. **Bedienung - Direkt am SDR:** Eine Bedienung des SDR könnte am SDR selbst mit Hilfe eines Displays und Tasten realisiert werden. Wichtig könnte dies für die Konfiguration des UDP-Endpunkts des Steuerkanals sein, da nicht alle IP-Netze mit der Standard-IP (Klasse-C Netz) des SDR umgehen können und diese vorher umkonfiguriert werden muss.
4. **DDC-Core - Konfigurierbare Tiefpass-Kaskade:** Im DDC-Core wird je Datenpfad eine Tiefpassfilter-Kaskade genutzt, um hochfrequente Reste des Mischens zu entfernen und gleichzeitig eine Dezimation des Datenstroms zu erreichen. Diese Filter sind jedoch fest auf 200 kHz Eckfrequenz dimensioniert. Sollte ein anderes Frequenzband wie VHF-2 genutzt werden, kann es zu Problemen kommen, da das Filter zu schmal-

beziehungsweise zu breitbandig ist. Es sollte dafür gesorgt werden, diese Filter per Software konfigurieren zu lassen.

5. **DDC-Core - Konfigurierbare Dezimation**: Durch eine feste Tiefpassfilter-Kaskade ist die minimale Samplefrequenz gegeben und es besteht kein Bedarf, die Dezimation ändern zu wollen. Bei einer variablen Filterstruktur muss jedoch eine geänderte Dezimation erfolgen. Somit ist eine konfigurierbare Dezimation für spätere SDR Versionen vorzusehen.

6. **DDC-Core - Konfigurierbare Interruptrate:** Sollte der DDC-Core eine bestimmte Datenmenge (hier fest bei 32 KiB) in das RAM geschrieben haben wird ein Interrupt ausgelöst. Diese 32 KiB sind nicht für alle Systeme sinnvoll. Zum Beispiel Systeme mit kleinerer Taktrate würden mit einer langsameren Interruptrate besser umgehen können. Sollte die NPI-Schnittstelle zum RAM bestehen bleiben, könnte eine einstellbare Interruptrate diesen Core auch für andere Embedded Systeme interessant machen.

7. **DDC-Core - 32-Bit Speicheradresse für RAM-Zugriff:** In [29] wurde im Register-File des DDC-Cores eine Speicherkapazitätsangabe implementiert, die auf 512 Byte großen Chunks basiert. Für eine universelle Nutzung sollte diese Kapazität auf eine 32-Bit Speicheradresse geändert werden. Hiermit ist eine byteweise Angabe des DDC-Core Speicherbereichs möglich.

8. **DDC-Core - Schnittstelle für Signalverarbeitung:** Die Signalverarbeitung auf dem PC, wie sie jetzt vorgesehen ist, ist sehr flexibel, aber auch sehr rechenintensiv. Sollte ein bestimmter Verarbeitungs-Algortithmus fertig entwickelt worden sein, wäre eine Schnittstelle des DDC-Core für eine FPGA-basierte Signalverarbeitung nötig. Diese würde erlauben, den Datenstrom nicht über UDP sondern durch einen zweiten Core zu schicken, der die Signalverarbeitung übernimmt und hinterher die Daten über UDP versendet.

9. **DDC-Core - Allgemeine ADC-Schnittstelle:** Die Schnittstelle zum verwendeten ADC ist überschaubar und einfach. Es gibt jedoch andere ADCs, die eine andere Schnittstelle stellen. Der DDC-Core wurde so entworfen, dass dieser nur mit dem hier verwendeten ADC funktioniert oder solchen mit einer gleichen Schnittstelle. Eine allgemeine Schnittstelle könnte ermöglichen, auch anderen ADCs mit dem DDC-Core zu arbeiten.

10. **Frontend - Konfigurierbares Bandselektionsfilter:** Das hier realisierte Frontend dient nur zum demonstrativen Zweck. Eine SDR-Plattform für Ausbildungszwecke sollte jedoch mehr Potential für weitere Frequenzbänder bieten, nicht nur VHF-2. Hier ist allerdings ein komplexes Frontend nötig, bei welchem das SDR das Frontend digital steuern kann, um dieses für den jeweiligen gewünschten Frequenzbereich zu konfigurieren. Eine Einstellungsmöglichkeit sollte hier der analoge Bandselektionsfilter sein.

11. **Frontend - Konfigurierbare Verstärkung:** Das analoge Frontend kann mit verschiedenen Antennen versehen werden, sodass am ADC die verschiedensten Signalpegel ankommen können. Eine digital steuerbare Verstärkung in einem komplexen Frontend könnte genutzt werden, um den ADC stets voll auszusteuern.

12. **System - Implementierung digitaler Demodulationen für Testzwecke**: Diese Arbeit hat sich mit der Demodulation des Mono-Audiosignals eines FM-Rundfunksender als Demonstrator beschäftigt. Mit dem realisierten VHF-2 Frontend sind jedoch auch die digitalen Signale eines FM-Rundfunksenders zugänglich (zum Beispiel RDS mit TMC). Es könnte also eine Demodulation für diese digitalen Signale auf dem PC oder auch im FPGA durch einen extra Core realisiert werden.

13. **System - Automatische Frontendkonfiguration:** Sollte ein komplexes Frontend mit einstellbarer Verstärkung, wählbaren Filtern, usw. eingesetzt, werden ist eine automatische Frontendkonfiguration denkbar. Das heißt ein Treiber oder Subsystem im Embedded System könnte anhand des gewünschten Frequenzbereiches die Filter einstellen und mit dem gelieferten Pegel des ADC die Verstärkung auf möglichst Fullscale erreichen.

14. **System - Sendezweig im SDR:** Mit einem Empfänger-SDR für die HAW kann bereits ein Gewinn erzielt werden, um zum Beispiel bestimmte Demodulation-Algorithmen zu testen. Dennoch sollte die Möglichkeit bestehen, auch Daten in die andere Richtung zu befördern. Wenn das System um einen Sendezweig erweitert werden würde, könnte dann auch eine gesamte Übertragungsstrecke realisiert werden.

15. **System - UDP/IP aus digitaler Logik:** In dem realisierten System wird ein großer Teil der Prozessorleistung von der Arbeit des softwarebasierten TCP/IP-Stacks verbraucht. Eine Einsparung an Prozessorressource könnte dem SDR ermöglichen, noch mehr Funktionalitäten unterzubringen. Eine Lösung könnte sein, den UDP/IP-Zweig eines TCP/IP-Stacks in digitaler Logik abzubilden. Hiermit könnte dem Prozessor dann immer nur ein neues UDP-Pakete signalisiert werden anstatt ein Ethernet-Paket, welches erst noch durch Software zu einem UDP-Paket verarbeitet werden muss.

16. **System - Eigene Platine für das SDR:** Es werden bis jetzt nur Hardware-Evaluations-Kits genutzt, welche zu groß sind und unnötige Funktionen mit sich bringen. Warum also nicht eine eigene Hardwarebasis erstellen? Ein komplexes Frontend wäre schon ein erster Schritt. Dieser Gedanke macht nur dann Sinn, wenn ein definierter Plan für eine nächste SDR-Version besteht und keine einzelnen Prototypen die Version bereits gefestigt haben.

Weitere Abschlussarbeiten sollten individuell zusammengestellt werden, wobei die obige Liste als Stütze dienen kann. Als weitere Grundlage hierfür folgt eine Liste mit möglichen Abschlussarbeiten die einige der oben genannten Themen aufgreifen.

1. **Konzeption einer SDR-Endversion:** Das *ultimative SDR für Bildungs- und Entwicklungszwecke* existiert nicht, auch wenn Projekte wie SDR4ALL bereits in diese Richtung gehen. Aber was genau kann ein *ultimatives SDR für Bildungs- und Entwicklungszwecke* sein? Hierfür könnte eine Abschlussarbeit erstellt werden, die genau dieses klärt. Hierzu gehören Fragen wie, welche Funktionen sind für ein solches System wünschenswert? Oder wie könnte es zusammengesetzt sein? Diese Arbeit beziehungsweise das entstandene Konzept könnte dann als Planungsgrundlage für spätere SDR-Versionen dienen, sodass abgeleitet daraus neue Prototypen entstehen könnten.
2. **Realisierung eines komplexen analogen Frontends:** Ein komplexes Frontend soll dem realisierten SDR ermöglichen, mit den verschiedensten Frequenzbereichen zu arbeiten. Ein digital konfigurierbares universelles Frontend könnte dem SDR dieses ermöglichen. Konfiguriert werden könnten zum Beispiel Bandselektionsfilter oder Breitbandverstärker-Blöcke für den jeweiligen Frequenzbereich. Ein spezielle Frontend-Core köntne Memory Mapped genutzt werden das Frontend automatisch zu konfigurieren. Hierfür würde dem Core nur der gewünschte Frequenzbereich mitgeteilt werden, dieser sorgt dann selber für die Wahl des richtigen Filters und stellt die Verstärker für eine Fullscale Aussteuerung des ADCs ein. Auch denkbar wäre eine allgemeine Schnittstelle zum FPGA, die es erlaubt, einfach verschiedenste Frontends anzubinden.
3. **Erweiterung des DDC-Cores für eine allgemeinere Einsatzmöglichkeit:** Der DDC-Core kann nicht ohne weiteres in anderen System genutzt werden, denn er ist nicht konfigurierbar. Er hat zum Beispiel eine feste Filterstruktur, Dezimierung, Interruptrate und ADC-Schnitttslle. Es könnte in einer weiteren Arbeit eine Analyse stattfinden welche zeigt, was ein solcher DDC-Core für Funktionalitäten benötigt, die es erlauben, ihn flexibel in den verschiedensten Systemen einzusetzen. Eine Implementierung der Ideen in den bestehenden Core würden die Arbeit abschließen. Die Anregungen der obigen Aufzählung könnten hierzu Denkanstöße liefern.

Literaturverzeichnis

[1] AMCC. *PPC440 Processor Users Manual - UM2013*. http://www.phxmicro.com/CourseNotes/AMCC_UM/PPC440_UM2013.pdf, 1.09 edition, März 2008.

[2] Bjarne Mjelde. *The Perseus SDR*. http://www.kongsfjord.no/bm/Perseus%20SDR.pdf, Januar 2008.

[3] Clemens Seidenberg. *SDR der nächsten Generation: der PERSEUS von Nico Palermo*. http://www.funkempfang.de/funkempfang/8service/pdf/PERSEUS.pdf, Dezember 2007.

[4] Comedi. Webseite. http://www.comedi.org/. 18. Juli 2011.

[5] Commgenuity. Webseite. http://www.coolusbradio.com/. 22. Juli 2011.

[6] Daniel Ch. von Grüningen. *Digitale Signalverarbeitung*. Carl Hanser Verlag, 2008.

[7] Ed Hallett. *Reference System: XPS LL Tri-Mode Ethernet MAC Embedded Systems for MicroBlaze and PowerPC Processors - XAPP1041*. http://www.xilinx.com/support/documentation/application_notes/xapp1041.pdf, 2.0 edition, September 2008.

[8] Ettus Research. Webseite. http://www.ettus.com/. 07. Juli 2011.

[9] GNU Radio. Webseite. http://gnuradio.org/. 18. Juli 2011.

[10] John G. Proakis, Masoud Salehi. *Grundlagen der Kommunikationstechnik*. Pearson, 2003.

[11] Jürgen Reichardt. *Lehrbuch Digitaltechnik*. Oldenbourg Wissenschaftsverlag GmbH, 2009.

[12] Linear Technology. *LTC2207/LTC2206 16-Bit, 105Msps/80Msps ADCs*. http://cds.linear.com/docs/Datasheet/22076fc.pdf, August 2009.

[13] Linear Technology. *QUICK START GUIDE FOR DEMONSTRATION CIRCUIT 918 - 16/14 BIT 40 TO 105 MSPS ADC*. http://cds.linear.com/docs/Demo%20Board%20Manual/dc918C_A-L.pdf, Juni 2009.

[14] Linear Technology. *QUICK START GUIDE FOR DEMONSTRATION CIRCUIT 718 - QUICKEVAL-II*. http://cds.linear.com/docs/Demo%20Board%20Manual/dc718C.pdf, Juli 2011.

[15] Martin Werner. *Nachrichtentechnik: Eine Einführung für alle Studiengänge*. Viweg&Teubner Verlag, 2010.

[16] Marvell. *Alaska Single-Port Gigabit Ethernet Transceiver 88E1111*. http://www.xilinx.com/products/boards/ml505/datasheets/M88E1111.pdf, 88E1111-00 edition, Oktober 2002.

[17] Mathworks. MATLAB® 7 MAT-File Format. PDF-Dokument. http://www.mathworks.com/help/pdf_doc/matlab/matfile_format.pdf. 18. Juli 2011.

[18] Mathworks. MatWorks Deutschland - MATLAB. Webseite. http://www.mathworks.de/products/matlab/?s_cid=global_nav. 18. Juli 2011.

[19] Mathworks. *Real-Time Windows Target 3.7*. http://www.mathworks.com/products/datasheets/pdf/real-time-windows-target.pdf, 3.7 edition, Juni 2011.

[20] microtelecom. Webseite. http://www.microtelecom.it. 07. Juli 2011.

[21] microtelecom. Perseues SDR Home Page. Webseite. http://www.microtelecom.it/perseus/. 07. Juli 2011.

[22] National Instruments. NI LabView. Webseite. http://sine.ni.com/np/app/flex/p/ap/global/lang/de/pg/1/docid/nav-77/. 18. Juli 2011.

[23] Nils Schiffhauer. *PERSEUS - eine Kurzwellen-Revolution, die begeistert!* http://www.ssb.de/pdfs/Perseus_SDR_1107.pdf, November 2007.

[24] SDR4ALL. Webseite. http://www.sdr4all.org/. 18. Juli 2011.

[25] Software Radio Laboratory LLC. SRL-LLC QS1R versus microtelecom.it Perseus Feature Comparison. http://www.srl-llc.com/QS1R-vs-perseus.pdf. 07. Juli 2011.

[26] Software Radio Raboratory LLC. Quicksilver. Webseite. http://www.srl-llc.com/. 07. Juli 2011.

[27] Stephen MacMahon, Nan Zang, Anirudha Sarang. *LightWeight IP (lwIP) Application Examples - XAPP1026*. http://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf, 3.1 edition, April 2011.

[28] Supélec. Webseite. http://www.supelec.fr/. 18. Juli 2011.

[29] Valentin Stanev. Abtastratenumsetzung nach dem Direct-DownConversion Prinzip als FPGA-IP-Core für nachfolgende Auswertung durch Matlab. Master's thesis, Hochschule für Angewandte Wissenschaften Hamburg, http://opus.haw-hamburg.de/volltexte/2011/1243/pdf/VS_Masterthesis.pdf, Februar 2011.

[30] Wikipedia. Bandpassunterabtastung. Webseite. http://de.wikipedia.org/w/index.php?title=Bandpassunterabtastung&oldid=77542890. 7. August 2010.

[31] Wikipedia. DirectBand. Webseite. http://en.wikipedia.org/w/index.php?title=DirectBand&oldid=439810701. 16. Juli 2011.

[32] Wikipedia. FM-Stereo. Webseite. http://de.wikipedia.org/w/index.php?title=FM-Stereo&oldid=84110539. 22. Januar 2011.

[33] Wikipedia. PCI-Express. Webseite. http://de.wikipedia.org/w/index.php?title=PCI_Express&oldid=89199610. 31. Mai 2011.

[34] Wikipedia. Quadraturamplitudenmodulation. Webseite.

[35] Wikipedia. Radio Data System. Webseite. http://en.wikipedia.org/w/index.php?title=Radio_Data_System&oldid=439759560. 16. Juli 2011.

[36] Wikipedia. Serial ATA. Webseite. http://de.wikipedia.org/w/index.php?title=Serial_ATA&oldid=89441929. 30. Mai 2011.

[37] Wikipedia. UKW-Rundfunk. Webseite. http://de.wikipedia.org/w/index.php?title=UKW-Rundfunk&oldid=89759781. 7. Juni 2011.

[38] Wikipedia. Universal Serial Bus. Webseite. http://de.wikipedia.org/w/index.php?title=Universal_Serial_Bus&oldid=89866379. 10. Juni 2011.

[39] Wikipedia. USB-Massenspeicher. Webseite. http://de.wikipedia.org/w/index.php?title=USB-Massenspeicher&oldid=89775958. 9. Juni 2011.

[40] Winrad. Webseite. http://www.winrad.org/. 18. Juli 2011.

[41] Wireless Innovation Forum. The Wireless Innovation Forum. Webseite. http://www.wirelessinnovation.org/. 18. Juli 2011.

[42] Xilinx. Webseite. http://www.xilinx.com/company/gettingstarted/. 18. Juli 2011.

[43] Xilinx. *PLB IPIF - DS448*. http://www.xilinx.com/support/documentation/ip_documentation/plb_ipif.pdf, v2.02 edition, April 2005.

[44] Xilinx. *Virtex-5 FPGA Integrated Endpoint Block for PCI Express Designs - User Guide - UG197*. http://www.xilinx.com/support/documentation/user_guides/ug197.pdf, 1.5 edition, Juli 2009.

[45] Xilinx. *XPS UART Lite (v1.01a) - DS571*. http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/mb_ref_guide.pdf, 1.01a edition, Dezember 2009.

[46] Xilinx. *Embedded Processor Block in Virtex-5 FPGAs Reference Guide - UG200*. http://www.xilinx.com/support/documentation/user_guides/ug200.pdf, 1.8 edition, Februar 2010.

[47] Xilinx. *LogiCORE IP Processor Local Bus (PLB) v4.6 (v1.05a) - DS531*. http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf, 1.05a edition, September 2010.

[48] Xilinx. *MicroBlaze Processor Reference Guide Embedded Development Kit EDK 12.4 - UG081*. http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/mb_ref_guide.pdf, 11.4 edition, November 2010.

[49] Xilinx. *LogiCORE IP DDR2 Memory Controller for PowerPC 440 Processors DS567*. http://www.xilinx.com/support/documentation/ip_documentation/ppc440mc_ddr2.pdf, 3.00c edition, März 2011.

[50] Xilinx. *LogiCORE IP Multi-Port Memory Controller (MPMC) (v6.03.a) - DS643*. http://www.xilinx.com/support/documentation/ip_documentation/mpmc.pdf, v6.03.a edition, März 2011.

[51] Xilinx. *LogiCORE IP XPS LL TEMAC (v2.03a) - DS537*. http://www.xilinx.com/support/documentation/ip_documentation/xps_ll_temac.pdf, 2.03a edition, Dezember 2011.

[52] Xilinx. *ML505/ML506/ML507 Evaluation Platform - User Guide - UG347*. http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf, 3.1.2 edition, Mai 2011.

Anhang

Alle unten aufgeführten Anhänge sind in elektronischer Form auf einer CD im Unterordner *anhang* abgelegt. Diese ist beim Prüfer Prof. Dr. rer nat Jürgen Reichardt einzusehen.

1. **iPerf** (anhang/iperf-2.0.5.zip): Quellcode zum Programm iPerf (Linux oder Cygwin).
2. **jPerf** (anhang/jperf-2.0.2.zip) Kompilierte Win32 Version von iPerf (siehe *./bin*) mit einer Java-basierten GUI für iPerf.
3. **EDK-PRJ: PPC-System** (anhang/IsePpc400.zip) Xilinx ISE Design Suite EDK und SDK Projektdateien zum PPC440-Testsystem für die Ermittlung der praktisch erreichbaren Ethernet und UDP/IP Übertragungsraten.
   ./ -> ISE-Projekt
   ./edk_ppc_400_mpmc -> EDK-Projekt.
   ./edk_ppc_400_mpmc/SDK/SDK_Workspace -> SDK-Workspace
4. **EDK-PRJ: MB-System** (anhang/IseMb125.zip) Xilinx ISE Design Suite EDK und SDK Projektdateien zum MB-Testsystem für die Ermittlung der praktisch erreichbaren Ethernet und UDP/IP Übertragungsraten.
   ./ -> ISE-Projekt
   ./edk_mb_125 -> EDK-Projekt
   ./edk_mb_125/SDK/SDK_Workspace -> SDK-Workspace
5. **C-PATCH: Csum-Offload** (anhang/lwip130UdpTxCsumOffloading.zip) Mit TortoiseSVN erstelltes Patch, welches der Xilinx lwIP130-Bibliothek v3.00.a erlaubt Checksum-Offloading auch für UDP im Sendebetrieb zu nutzen. Dieses Patch ist kompatibel zu den Programmen *diff* und *patch* von Unix/Linux Systemen.
6. **M: Aquire from Stanev Temp** (anhang/StanevTempFileDemod.zip) MATLAB-Skript, welches mit der bestehenden, USB-basierten, Lösung [29] Daten akquiriert und anschließend eine FM-Demodulation mit einer MATLAB-Funktion durchführt.
7. **WAV: Aquire with USB** (anhang/WavSampleStanevSystem.zip) WAV-Datei mit dem Demodulationsergebnis des Anhangs *M: Aquire from Stanev Temp*.

8. **EDK-PRJ: Fixed USB** (anhang/FixedStanevEdk124.zip) Xilinx ISE Design Suite EDK und SDR Projektdateien der bestehenden, USB-basierten, Lösung [29] mit reparierter Firmware und portiert von der ISE Version 11.4 zu der Version 12.4.
9. **CMEX: USB-Client** (anhang/HawSdrUsbClient.zip) Als CMEX-Lösung portierte und reparierte Version des DdcUSbReaders der bestehenden Lösung [29].
10. **M: Aquire with CMEX** (anhang/UsbToWav.zip) MATLAB-Skript, welches mit der erstellten CMEX-Lösung (Anhang *CMEX: USB-Client*) und der reparierten Firmware (Anhang *EDK-PRJ: Fixed USB*) Daten akquiriert.
11. **EDK-PRJ: Main-System** (anhang/MainEdkPrjAndApp.zip) Xilinx ISE Design Suite EDK und SDR Projektdateien des realisierten SDR Endsystems.
    ./ ISE Projekt
    ./edk_ppc_400_mpmc EDK-Projekt
    ./edk_ppc_400_mpmc/SDK/SDK_Workspace SDK-Workspace
12. **M: CMEX FM-Demod** (anhang/UsbFmDemodCounting.zip) MATLAB-Skript für den Test der Frequenzzähl-Demodulations Methode.
13. **WAV: FM Demod Sample** (anhang/WavSampleCountingFmDemod.zip) WAV-Datei mit dem Demodulationsergebnis des MATLAB-Skriptes aus Anhang *M: CMEX FM-Demod*.
14. **M: CMEX Frontendtest** (anhang/UsbToWav.zip) Beinhaltet den gleichen Inhalt wie der Anhang *M: Aquire with CMEX*.
15. **C: Main-App** (anhang/MainEdkPrjAndApp.zip) Beinhaltet den gleichen Inhalt wie der Anhang *EDK-PRJ: Main-System*.
16. **C#: HawSdrClient** (anhang/HawSdrClient.zip) Microsoft Visual Studio C# Express Projektdateien des HawSdrClients.
17. **M: Display SDR MAT** (anhang/MatDisplayData.zip) MATLAB-Skript zum auslesen der Daten der SDR-Endlösung über UDP und zur Darstellung in mehreren Grafiken.
18. **MISC: Lumped Filter** (anhang/LumpedFilter.zip) Sämtliche Design, Analyse und Messergebnisse zum realisierten, SMD-basierten, Bandpass-Filter.

Glossar

ADC Analog Digital Converter

AMBA Advanced Microcontroller Bus Architecture

CLI Common Language Infrastructure

CPLD Complex Programmable Logic Device

CPU Central Processing Unit

DAC Digital Analog Converter

DAQ Data Acquisition

DCM Digital Clock Manager

DDC Digital Down Conversion

DDS Direct Digital Synthesis

DFT Discret Fourier-Transformation

DLL Delay Locked Loop

DLL Dynamic Link Library

DMA Direct Memory Access

DSP Digital Signal Processor

DUC Digital Up Converter

EPP Extensible Processing Platforms

FFT Fast Fourier-Transformation

FPGA Field Programmable Gate Array

HAL Hardware Abstraction Layer

HAW Hochschule für angewandte Wissenschaften

HDL Hardware Description Language

HTML Hypertex Markup Language

IOB Input Output Block

IP Internet Protocol

IQ Inphase- und Quatraturteil

ISM Industrial, Scientific and Medical Band

KiB Einheit entspricht $1024 \cdot 1\ Byte$

Kib Einheit entspricht $1024 \cdot 1\ Bit$

LB Logic Block

LL LocalLink

LSB Least Significant Bit

MAC Media Access Control

MB Microblaze

MiB Einheit entspricht $1024 \cdot 1024 \cdot 1\ Byte$

Mib Einheit entspricht $1024 \cdot 1024 \cdot 1\ Bit$

MPMC Multi-Port Memory Controller

MSS Massenspeicher

OMAP Open Multimedia Application Platform

PCI Peripheral Component Interconnect

PGA Programmble Gain Amplifier

PLA Programmable Logic Array

PLB Peripheral Local Bus

PLL Phase Locked Loop

PPC PowerPC440

RDS Radio Data System

RF Radio Frequency

RFID Radio Frequency Identification

RT Realtime

Rx Receive

SDK Software Development Kit

SDR Software Defined Radio

SoC System-on-Chip

TCP Transmission Control Protocol

TI Texas Instruments

TMC Traffic Message Channel

TTM Time To Market

Tx Transmit

uC Mikrocontroller

UDP User Datagram Protocol

USB Universal Serial Bus

VCO Voltage Controlled Oscillator

VHDL Very High Speed Integrated Circuit HDL

VHF Very High Frequency

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 11. August 2011	
Ort, Datum	Unterschrift